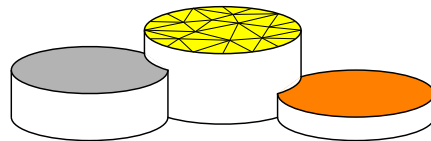


# PODIUM



A suite of software utilities for

**P**reparation **O**f **D**ata for **I**nversion on **U**nstructured **M**eshes

Peter G. Lelièvre

January 23, 2026

This document is a work-in-progress. Please contact me at [plelievre@mta.ca](mailto:plelievre@mta.ca) if something needs updating, you need something explained further, or for any suggestions or requests.

# Contents

<b>1</b>	<b>PODIUM preliminaries</b>	<b>5</b>
1.1	About PODIUM	5
1.2	A word on file formats	5
1.3	About this document	5
1.4	Summary of all utility programs	5
<b>2</b>	<b>File formats</b>	<b>10</b>
2.1	Unstructured meshes	10
2.2	Rectilinear meshes	10
2.2.1	Rectilinear mesh files	10
2.2.2	Rectilinear model files	11
2.3	Model discretization specification files	11
2.4	Physical property files	12
2.4.1	Specifying the physical property model	13
2.4.2	Specifying depth/distance/sensitivity weighting for inversion	13
2.4.3	Inversion regularization parameters	15
2.4.4	Constraint options	15
2.5	Joint coupling measure files	15
2.5.1	Coupling options (parameter coupling)	16
2.5.2	Weighting the coupling	18
2.5.3	Defining clusters	18
2.5.4	Single property inversion	20
2.6	Regions files	20
2.7	Rock-units files	20
2.8	Unstructured block files	21
<b>3</b>	<b>Utilities for file format conversion</b>	<b>22</b>
3.1	Unstructured mesh file formats	22
3.1.1	cells2nodes	22
3.1.2	ele2ele	22
3.1.3	ele2vtu	23
3.1.4	mesh2neigh	23
3.1.5	mesh2poly	23
3.1.6	mesh2tikz	24
3.1.7	mesh2vtu	24
3.1.8	node2d	25
3.1.9	node3d	25
3.1.10	node2node	25
3.1.11	node2vtu	25
3.1.12	poly2mesh	26
3.1.13	poly2vtu	26
3.2	Peter's file formats	27
3.2.1	blocks2poly	27
3.2.2	blocks2vtu	27
3.2.3	rockunits2ele	27
3.2.4	rockunits2node	28
3.3	UBC-GIF file formats	28
3.3.1	node2xyz	28
3.3.2	reorder_model	28

3.3.3	ubcgif2mesh	29
3.3.4	ubcgif2vtr	29
3.3.5	xyz2csv	30
3.3.6	xyz2node	30
3.3.7	xyz2xyz	31
3.4	Industry file formats	31
3.4.1	convert_format	31
3.5	Combining multiple files of the same format	34
3.5.1	combine_files	34
<b>4</b>	<b>Data processing</b>	<b>35</b>
4.1	Observation creation and data manipulation	35
4.1.1	add_noise	35
4.1.2	break_lines	36
4.1.3	decimate	37
4.1.4	decimate_by_mesh	38
4.1.5	decimate_by_nodes	38
4.1.6	drill_spline	38
4.1.7	ll2utm	39
4.1.8	make_obs	39
4.1.9	mag_dipole	40
4.1.10	remove_duplicates	40
4.1.11	remove_nodes	41
4.1.12	remove_range	41
4.1.13	remove_trend	42
4.1.14	reorder_attributes	43
4.1.15	smooth_node	43
4.1.16	transform_coordinates	44
4.2	Data visualization	45
4.2.1	drillcore2node	45
4.2.2	drillcores2mesh	45
4.3	Data query and calculations	46
4.3.1	calc_cor	46
4.3.2	calculate_correlation	46
4.3.3	cluster_analysis	47
4.3.4	data_addition	48
4.3.5	data_difference	49
4.3.6	data_histogram	50
4.3.7	lin_reg	50
4.3.8	linear_regression	51
4.3.9	matrix_sums	51
4.3.10	mean_coordinates	52
4.3.11	print_coordinates	52
4.3.12	sum_node	53
4.3.13	terrain_correction	53
4.4	Interpolation and projection	53
4.4.1	interpolate_data	53
4.4.2	project_geomag	54
4.5	Utilities for seismic and muon tomography data	55
4.5.1	check_combos	55
4.5.2	pierce_points	55
4.5.3	split_tobs	56
<b>5</b>	<b>Model processing</b>	<b>58</b>
5.1	Model creation and manipulation	58
5.1.1	add_blocks	58
5.1.2	boundary_outline	59
5.1.3	build_tetmesh	60
5.1.4	clean_surface	60
5.1.5	coarsen_model	60

5.1.6	conform_topography	61
5.1.7	count_cells	62
5.1.8	mark_model	62
5.1.9	mesh_core	62
5.1.10	mesh_faces	63
5.1.11	outlines2plc	63
5.1.12	pad_mesh	63
5.1.13	populate_model	64
5.1.14	remove_cells	65
5.1.15	remove_unused_nodes	65
5.1.16	reorder_regions	65
5.1.17	scoop_model	66
5.1.18	sew_surfaces	66
5.1.19	smooth_model	67
5.1.20	snap_surface	67
5.1.21	subdivide	68
5.1.22	surface_normals	69
5.1.23	threshold_attribute	70
5.1.24	threshold_edge_length	70
5.1.25	toposplit	71
5.2	Model visualization	72
5.2.1	make_scatter_plot	72
5.2.2	outlines2poly	72
5.3	Model query and calculations	73
5.3.1	bulk_magnetization	73
5.3.2	calculate_crossgrad	73
5.3.3	calculate_fcm	74
5.3.4	calculate_gradients	75
5.3.5	calculate_tvar	75
5.3.6	cell_dimensions	76
5.3.7	check_bounds	76
5.3.8	closest_nodes	77
5.3.9	check_coll	77
5.3.10	dyno_difference	77
5.3.11	find_groups	78
5.3.12	find_holes	78
5.3.13	model_difference	78
5.3.14	mvi_difference	79
5.3.15	pun2nun	79
5.3.16	query_model	80
5.3.17	sum_models	80
5.3.18	surf_volume	81
5.3.19	tetgen_intersections	81
5.4	Creation of inversion input files	82
5.4.1	check_dyno_bounds	82
5.4.2	make_face_weights	82
5.4.3	make_cell_weights	83
5.4.4	robust_weights	83
5.5	Interpolation	84
5.5.1	interpolate_mesh	84
5.5.2	interp_rect_mesh	85
<b>6</b>	<b>Miscellaneous utilities</b>	<b>87</b>
6.1	fit_mag_dipole	87
6.2	log2aux	87
6.3	rel2abs	87
6.4	remove_comments	88
<b>7</b>	<b>Change Log</b>	<b>89</b>

# Chapter 1

## PODIUM preliminaries

### 1.1 About PODIUM

PODIUM is a suite of software utilities for Preparation Of Data for Inversion on Unstructured Meshes. Despite the name, many utilities also exist in this package for working with rectilinear meshes. PODIUM does not include any forward or inverse modelling programs; for those programs, see the MAGNUM package. The software is provided to assist users in preparing data and models for running forward and inverse modelling programs, and for assessing the results of such modelling. The package has grown from the needs of many researchers working with real and synthetic data examples. While it meets our needs, your needs may differ. Therefore, we encourage users to suggest additional functionality or other general improvements.

### 1.2 A word on file formats

Because I primarily work with unstructured meshes, most of these utility programs have been designed for working with the file formats used by the [Triangle](#) and [TetGen](#) meshing programs. This means you may have to convert your data or model files into those formats before using some of the PODIUM utilities, followed by conversion back to the original format. I provide many file format utilities for performing such conversions in [Chapter 3](#).

### 1.3 About this document

[Section 2](#) provides reference material to explain file formats used by various PODIUM programs. Documentation on the PODIUM programs starts in [Section 3](#).

### 1.4 Summary of all utility programs

Below I provide a brief summary of all utility programs in the PODIUM package. The program names below are hyperlinked to sections that provide further information.

#### Utilities for file format conversion

##### Unstructured mesh file formats

- [cells2nodes](#): Moves a model on an unstructured mesh of cells to a cloud of nodes.
- [ele2ele](#): Reads a `.ele` file and re-writes it.
- [ele2vtu](#): Converts a `.ele` file to a 2D `.vtu` file for viewing in ParaView.
- [mesh2neigh](#): Reads information about an unstructured mesh defined by `.node` and `.ele` files, and writes a `.neigh` file containing cell neighbour information.
- [mesh2poly](#): Converts an unstructured mesh defined by `.node` and `.ele` files to a `.poly` file for use with the TetGen and Triangle meshing programs.
- [mesh2tikz](#): Converts `.node` and `.edge` files to a `.ltx` file containing plotting information for typesetting with L<sup>A</sup>T<sub>E</sub>X using the TikZ package.
- [mesh2vtu](#): Converts an unstructured mesh defined by a pair of `.node` and `.ele` files to a `.vtu` file for viewing in ParaView.
- [node2d](#): Changes the header of a 3D `.node` file to make it specify a 2D object.
- [node3d](#): Changes the header of a 2D `.node` file to make it specify a 3D object.
- [node2node](#): Reads a `.node` file and re-writes it.

- [node2vtu](#): Converts a `.node` file to a `.vtu` file for viewing in ParaView.
- [poly2mesh](#): Converts a `.poly` file to a pair of `.node` and `.ele` files.
- [poly2vtu](#): Converts a `.poly` file to a `.vtu` file for viewing in ParaView.

### Peter's file formats

- [blocks2poly](#): Writes a `.poly` file containing a number of different rectangular or prismatic blocks.
- [blocks2vtu](#): Writes a `.vtu` file containing a number of different rectangular or prismatic blocks.
- [rockunits2ele](#): Reads rock unit information from a file and adds that rock unit information to the attributes columns in a `.ele` file or UBC-GIF format model file.
- [rockunits2node](#): Reads rock unit information from a file and adds that rock unit information to the attributes columns in a `.node` file.

### UBC-GIF file formats

- [node2xyz](#): Converts a `.node` file to a simple `.xyz` format file.
- [reorder\\_model](#): Reorders a model on a rectilinear mesh from one ordering system to another.
- [ubcgif2mesh](#): Converts a UBC-GIF model on a rectilinear mesh to unstructured mesh format.
- [ubcgif2vtr](#): Converts UBC-GIF format mesh and model files to a `.vtr` file for viewing in ParaView.
- [xyz2csv](#): Reads a file and replaces sequences of spaces with commas.
- [xyz2node](#): Converts a column-based data file (e.g. `.xyz`) to a `.node` format file.
- [xyz2xyz](#): Extracts a single column out of a multiple-column data file and writes it to a single-column data file.

### Industry file formats

- [convert\\_format](#): Converts from various industry file formats to unstructured mesh file formats, or the reverse.

### Combining multiple files of the same format

- [combine\\_files](#): Combines the information in several sets of `.node`, `.ele`, `.neigh`, or `.poly` files.

## Data processing

### Observation creation and data manipulation

- [add\\_noise](#): Adds noise and/or standard deviations to data in a `.node` or `.ele` file.
- [break\\_lines](#): Breaks the data observation locations in a `.node` file into lines based on data separation distances and line deviation angles.
- [decimate](#): Decimates (down-samples) data rows from a `.node` or `.ele` file.
- [decimate\\_by\\_mesh](#): Decimates data based on a UBC-GIF mesh file such that there is only one observation point above each vertical column of cells.
- [decimate\\_by\\_nodes](#): Decimates data so there are no observation points within some distance from a second set of points.
- [drill\\_spline](#): Fits a Catmull-Rom spline through down-hole observation locations and writes interpolated locations to a new file.
- [ll2utm](#): Converts the coordinates from a 3D `.node` file from longitude/latitude to UTM (WGS84, units in meters).
- [make\\_obs](#): Creates gridded observation locations and writes them to a `.node` or `.ele` file.
- [mag\\_dipole](#): Generates the TMI response of a magnetic dipole.
- [remove\\_duplicates](#): Removes duplicate or closely spaced observation points from a `.node` file.
- [remove\\_nodes](#): Removes any nodes in a nodes file, or in an unstructured mesh, with a specified elevation or attribute value.
- [remove\\_range](#): Removes nodes from a particular range in a `.node` file or masks those nodes by a polygon.

- [remove\\_trend](#): Removes a polynomial trend from  $(x, y, d)$  data in a `.node` file.
- [reorder\\_attributes](#): Reorders the attributes in a `.node` or `.ele` file.
- [smooth\\_node](#): Smooths the coordinates in a `.node` file using a simple neighbourhood-averaging method.
- [transform\\_coordinates](#): Coordinate transformation of data or a model.

## Data visualization

- See programs [node2vtu](#) and [ele2vtu](#).
- [drillcore2node](#): Converts drillcore information from several `.csv` files to a `.node` file.
- [drillcores2mesh](#): Converts drillcore information (at the moment only one attribute) from a `.csv` formatted drillcore intervals file to a `.node` and `.ele` file pair.

## Data query and calculations

- [calc\\_cor](#): Calculates the correlation between two sets of numbers.
- [calculate\\_correlation](#): Calculates the correlation between two sets of numbers.
- [cluster\\_analysis](#): Performs cluster analysis.
- [data\\_addition](#): Adds attributes from two datasets together.
- [data\\_difference](#): Provides statistics on the difference between two data sets.
- [data\\_histogram](#): Reads data from a file and prints histogram information for a specified attribute column.
- [lin\\_reg](#): Performs linear regression on two sets of numbers.
- [linear\\_regression](#): Performs linear regression on two sets of numbers.
- [matrix\\_sums](#): Reads a matrix file and outputs various information about the matrix.
- [mean\\_coordinates](#): Calculates the averages of the coordinate columns in two or more node files.
- [print\\_coordinates](#): Reads data from a file and prints coordinate and attribute range information (minimum, maximum values, etc.).
- [sum\\_node](#): Sums the attribute column(s) in one or more node files.
- [terrain\\_correction](#): Performs terrain correction for vertical component gravity data.

## Interpolation and projection

- See also program [interpolate\\_mesh](#).
- [interpolate\\_data](#): Data interpolation at specified points.
- [project\\_geomag](#): Projects a geomagnetic reference field vector onto a vertical cross-section.

## Utilities for seismic and muon tomography data

- [check\\_combos](#): Takes three files defining seismic sources, receivers and source-receiver combinations and generates a `.vtu` file showing the connections.
- [pierce\\_points](#): Takes files defining a 3D unstructured mesh, tomography sources, receivers and source-receiver combinations and generates a `.node` file containing the pierce points.
- [split\\_tobs](#): Splits a traveltime data file into three files containing sources, receivers and traveltime information.

## Model processing

### Model creation and manipulation

- [add\\_blocks](#): Adds rectangular prisms, spheres and horizontal discs inside models on rectilinear or unstructured meshes.
- [boundary\\_outline](#): Determines the boundary outline of a meshed surface.
- [build\\_tetmesh](#): Allows the user to build tetrahedral meshes following typical forward and inverse modelling requirements.

- [clean\\_surface](#) Performs several cleaning operations on a surface model.
- [coarsen\\_model](#): Coarsens a mesh-based model using a thresholding, grouping, and averaging method.
- [conform\\_topography](#): Cuts the lateral sides of a single 3D boundary block by a topography surface.
- [count\\_cells](#): Counts the cells and calculates a volume or area sum for a collection of triangular or tetrahedral cells.
- [mark\\_model](#): This program has been absorbed into program [populate\\_model](#) and is now obsolete.
- [mesh\\_core](#): Extracts the core (removes padding cells) from a UBC-GIF mesh.
- [mesh\\_faces](#): Converts an unstructured mesh of triangular (2D) or tetrahedral (3D) cells to one containing line element (2D) or triangular (3D) cell face information.
- [outlines2plc](#): Generates a 3D poly file containing outlines specified in 2D input files. Designed for micromagnetic work on thin sections of drillcore.
- [pad\\_mesh](#): Adds padding cells to the sides and bottom of a UBC-GIF mesh using a user-defined expansion ratio.
- [populate\\_model](#): Populates a voxel model with values at specified points.
- [remove\\_cells](#): Removes any cells in an unstructured mesh with a specified attribute value.
- [remove\\_unused\\_nodes](#): Removes unused nodes from a `.nodes` file and adjusts the cell definitions accordingly in a companion `.ele` file.
- [reorder\\_regions](#) Reorders regions of cells in a `.ele` file.
- [scoop\\_model](#): This program is for setting particular regions of a model to constant values.
- [sew\\_surfaces](#): Takes two surfaces and sews their lateral sides together.
- [sew\\_surfaces\\_to\\_boundary](#): This program is obsolete. Use programs [snap\\_surface](#), [sew\\_surfaces](#) and [combine\\_files](#) instead.
- [smooth\\_model](#): Smooths a mesh-based model using a simple neighbour-averaging method.
- [snap\\_surface](#): Snaps the
- [subdivide](#): Performs a surface subdivision on a 2D or 3D wireframe surface model or a 2D triangular mesh.
- [surface\\_normals](#): Orders the node indices in the facet definitions for a 2D polygon/polyline or 3D triangulated surface such that the normals for each facet (2D edge or 3D triangular face) follow some specified rules.
- [threshold\\_attribute](#): Thresholds a tetrahedral mesh by cell attribute and extracts the bounding triangulated surface for the resulting body.
- [threshold\\_edge\\_length](#): Thresholds a tetrahedral mesh by cell edge length and extracts the bounding triangulated surface for the resulting body.
- [toposplit](#): Reads files specifying a mesh (rectilinear or unstructured) and wireframe topography (unstructured), and writes a model file specifying the cells above and below the topography surface.

## Model visualization

- [make\\_scatter\\_plot](#): Creates a scatter-plot (cross-plot) from two `.ele` files.
- [outlines2poly](#): Generates a 2D poly file containing several outlines specified in an input file.
- See also programs [ele2vtu](#), [mesh2vtu](#), [poly2vtu](#) and [ubcgif2vtr](#).

## Model query and calculations

- [bulk\\_magnetization](#): Calculates the bulk magnetization in a model with option to mask the calculation.
- [calculate\\_crossgrad](#): Calculates the cross-gradient measure.
- [calculate\\_fcm](#): Calculates the fuzzy c-means (FCM) measure.
- [calculate\\_gradients](#): Calculates spatial model gradients.
- [calculate\\_tvar](#): Calculates the total variation (total gradient) for a model on a rectilinear or unstructured mesh.
- [cell\\_dimensions](#): Calculates cell dimension information for an unstructured mesh.



- [check\\_bounds](#): Checks if a model is on bounds (checks if the bounds are “active”) and calculates the distance of the model to the bounds in physical property space.
- [check\\_bounds](#): Checks a surface mesh for any intersections of its facets.
- [closest\\_nodes](#): Finds the indices of a set of nodes closest to another set of nodes.
- [dyno\\_difference](#): Calculates and displays statistical information between a surface model inverted with DyNo, and its corresponding true model. Used in comparing synthetic modelling results.
- [find\\_groups](#): Reads unstructured mesh files and assigns new “CellGroup” attributes to each physically connected group of cells.
- [find\\_holes](#): Finds holes in a 3D surface of polygonal facets.
- [model\\_difference](#): Provides statistics on the difference between two UBC-GIF format models or any two column-based data files with the same number of values on every line.
- [mvi\\_difference](#): Calculates the difference between two magnetic vector inversion (MVI) recovered models.
- [pun2nun](#): Creates two surface meshes that represent  $\pm 1$  standard deviation of the nodes’ positions, the uncertainty resulting from an uncertainty in the model’s physical properties. Used on the resulting inversion model from DyNo. Only set for the magnetic susceptibility physical property, not yet known if functional for others.
- [query\\_model](#): Interpolates model values in a mesh (a voxel model) at specified points. Does essentially the opposite of program [populate\\_model](#).
- [sum\\_models](#): Sums two models.
- [surf\\_volume](#): Calculates the volume of each closed region of a surface mesh model.
- [tetgen\\_intersections](#): Takes the output of TetGen with the -d flag and writes a .poly file containing intersecting facets.

### Creation of inversion input files

- [make\\_face\\_weights](#): Sets face-based weights based on the values in an unstructured or rectilinear model.
- [make\\_cell\\_weights](#): Sets cell-based weights based on the values in an unstructured or rectilinear model.
- [robust\\_weights](#): Calculates the weights for the robust modelling iterative re-weighting scheme of [Günther & Rücker \(2006\)](#).
- [check\\_dyno\\_bounds](#): Studies a surface mesh with given node constraints from a `nodeunitsfile`, and displays the pairs of facet groups that could intersect during an inversion. Used in the writing of a `collfile`.

### Interpolation

- [interpolate\\_mesh](#): Interpolates values from one mesh to another.
- [interp\\_rect\\_mesh](#): Interpolates values from one rectilinear mesh to another.

### Miscellaneous utilities

- [fit\\_mag\\_dipole](#): Least-squares fitting of magnetic data by a sphere.
- [log2aux](#): Takes numbers from a DYNO .log file and places them into a .aux file.
- [rel2abs](#): Reads a .node file and converts relative node location bounds to absolute.
- [remove\\_comments](#): Removes any commented lines in a file.

# Chapter 2

## File formats

This chapter provides information about many file formats used by the software in this documentation. Later chapters in this documentation link back to this chapter to help you organize your input files.

**Avoid tab characters in all input files! Use spaces instead.**

**When specifying path/file names in an input file, place double quotes around them (e.g. `".././data.txt"`). Single quotes should also work but this is machine dependent.**

**Often, input files specify the locations of other files. In these cases, you can use an absolute or relative path name. A relative path name should be relative to the location of the input file in which it is specified. For example, if file `main_input_file.inp` contains the line**

```
meshinp "../mesh.inp"
```

**then the file `mesh.inp` should reside in the directory above the one where `main_input_file.inp` lives.**

### 2.1 Unstructured meshes

These file formats ...

- `.node`
- `.ele`
- `.neigh`
- `.poly`

... are explained in more detail on these webpages:

- Triangle: 2D tetrahedral meshing program  
<http://www.cs.cmu.edu/~quake/triangle.html>
- TetGen: 3D tetrahedral meshing program  
<http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual006.html>

I allow an extension to those formats: attribute names can be included in the first line of the `.node` and `.ele` files as follows:

```
[standard information] "name1","name2",...,"nameN"
```

where N is the number of attributes. The names MUST be contained within double quotes and separated by commas. I cannot guarantee that other software that works with `.node` and `.ele` files will accept this format extension.

### 2.2 Rectilinear meshes

#### 2.2.1 Rectilinear mesh files

UBC-GIF format mesh and model files are used for rectilinear meshes, which are explained further in various places here:

<http://gif.eos.ubc.ca>

The 3D mesh file format is as follows:

```
nE nN nV
EO NO VO
dE1 dE2 ... dEnE
dN1 dN2 ... dNnN
dV1 dV2 ... dVnV
```

where

- **nE**, **nN** and **nV** define the number of cells in the Easting, Northing and Vertical directions respectively
- point (EO,NO,VO) defines the coordinates of the top SW corner of the mesh
- the last three lines define the cell dimensions along each Cartesian direction.

On the cell dimension specification lines (the last three lines) you can indicate groups of equal cell dimensions using the syntax **n\*d** which indicates **n** repetitions of the dimension **d**. For example, The following lines are equivalent:

```
100 100 100 50 25 25 25 25 25 50 100 100 100
3*100 50 5*25 50 3*100
```

## 2.2.2 Rectilinear model files

The standard UBC-GIF model file format is a single column of model values. I allow two extensions to that format. A file can contain multiple columns, each representing a different mesh cell attribute; each row contains all attributes for a particular cell. A file can contain an optional commented header line that specifies the attribute names:

```
# "name1", "name2", ..., "nameN"
```

where N is the number of attributes. The names MUST be contained within double quotes and separated by commas. I cannot guarantee that the UBC-GIF utilities will accept these format extensions.

## 2.3 Model discretization specification files

These files tell the forward modelling and inversion programs how to discretize the model.

Each line of the input file should be of the format

```
name value
```

where **name** is the name of some modelling parameter and **value** is the value for that parameter. The possible parameters and default values are listed below. If a parameter is not found in the input file then the default value is used for that parameter. Note that some of the parameters are for use in inversion and are ignored for forward modelling purposes.

Lines in the input file beginning with the **#** or **!** character are ignored and can be used as comments for your own reference.

Name	Default	Brief description
<b>meshtype</b>	<b>"unstructured"</b>	the type of mesh (the other option is <b>"rectilinear"</b> )
<b>meshfile</b>	<b>" "</b>	file containing mesh information
<b>modelfile</b>	<b>" "</b>	file containing model information
<b>neighfile</b>	<b>" "</b>	another file containing mesh information (unstructured meshes only)
<b>split</b>	<b>0</b>	how to convert from rectilinear to unstructured mesh
<b>zdir</b>	<b>1</b>	specifies the coordinate system
<b>regionsfile</b>	<b>" "</b>	a list of regions that can be used with a surface model
<b>nodeunitsfile</b>	<b>" "</b>	a list of node-related rockunits that can be used with a surface model
<b>facetunitsfile</b>	<b>" "</b>	a list of facet-related rockunits that can be used with a surface model
<b>regionunitsfile</b>	<b>" "</b>	a list of region-related rockunits; can be used with voxel or surface model

- The model may be a 2D or 3D rectilinear or unstructured (triangular or tetrahedral) mesh.
- The unstructured mesh can define a voxel mesh or tessellated surface.
- An unstructured voxel mesh should contain triangular cells in 2D and tetrahedral cells in 3D.

- An unstructured tessellated surface should contain line element facets in 2D and polygonal (e.g. triangular) facets in 3D.
- For a rectilinear mesh, the `meshfile` and `modelfile` should be UBC-GIF format files.
- For an unstructured mesh, the `meshfile` and `modelfile` should be `.node` and `.ele` files respectively and the optional `neighfile` a `.neigh` file (if `neighfile` is set to "null" then the neighbour information is calculated automatically, which may be slow for large problems).

`zdir`

- For 3D, set `zdir > 0` to specify  $+z$  up,  $+x$  East,  $+y$  North;  
set `zdir < 0` to specify  $+z$  down,  $+x$  North,  $+y$  East.  
For 2D, the  $+x$  axis is always to the right and `zdir` only specifies the  $+z$  direction. The  $y$  axis is along-strike.
- Output mesh/model `.node` files have  $+z$  as specified.
- Output mesh/model `.vtu` files have  $+z$  up, regardless of the value supplied for `zdir`.

## 2.4 Physical property files

These files tell the forward modelling and inversion programs how to convert *models* to *physical properties*.

Each line of the input file should be of the format

`name value`

where `name` is the name of some modelling parameter and `value` is the value for that parameter. The possible parameters and default values are listed below. If a parameter is not found in the input file then the default value is used for that parameter. Note that some of the parameters are for use in inversion and are ignored for forward modelling purposes.

Lines in the input file beginning with the `#` or `!` character are ignored and can be used as comments for your own reference.

The capitalized headings in the table below link to sections where the parameters are discussed in more detail.

Name	Default	Brief description
<code>proptype</code>	<code>""</code>	e.g. set to <code>"den"</code> , <code>"sus"</code> etc. for density, mag. susceptibility, etc.
<a href="#">SCALING PARAMETERS</a>		
<code>mmul</code>	1.0	multiplicative scalar to convert model values to physical property values
<code>madd</code>	0.0	additive scalar to convert model values to physical property values
<code>mtrend</code>	0.0	background physical property depth trend
<code>uselog</code>	<code>t</code>	if true then log(conductivity) is used; only applies to conductivity models
<code>inputscaled</code>	<code>t</code>	specify if input model quantities are scaled (true) or actual (false) physical properties
<a href="#">INVERSION WEIGHTING PARAMETERS</a>		
<code>wmode</code>	<code>"none"</code>	defines what type of weighting is used in an inversion
<code>wbeta</code>	1.0	distance/sensitivity weighting strength
<code>wnorm</code>	2.0	distance/sensitivity weighting norm
<code>wpower</code>	0.0	depth/distance weighting power
<code>wzero</code>	0.0	depth/distance weighting $z_0/r_0$
<code>obsfile</code>	<code>""</code>	file containing observation locations for use with distance weighting
<code>zdir</code>	<code>""</code>	defines coordinate system for <code>obsfile</code> and <code>wzero</code> (if used)
<code>datatype</code>	<code>""</code>	specifies the data type (e.g. <code>"gz"</code> ) from which information is taken from for sensitivity weighting
<a href="#">INVERSION REGULARIZATION OPTIONS</a>		
<code>initfile</code>	<code>"null"</code>	file containing an initial model
<code>initindex</code>	0	attribute index for the initial model file
<code>initvalue</code>	<code>*</code>	initial model value
<code>reffile</code>	<code>"null"</code>	file containing a reference model
<code>refindex</code>	0	attribute index for the reference model file
<code>refvalue</code>	<code>*</code>	reference model value
<code>wsfile</code>	<code>"null"</code>	file containing smallness weights
<code>wsindex</code>	0	attribute index for the <code>wsfile</code> smallness weights file
<code>alphas</code>	0.0	multiplier on the smallness regularization
<code>alphab</code>	1.0	multiplier on both the smallness and smoothness regularization terms
<a href="#">INVERSION CONSTRAINT OPTIONS</a>		

Name	Default	Brief description
<code>boundsfile</code>	<code>"null"</code>	file containing model bounds
<code>lowerindex</code>	0	attribute index to use for the lower bound in a bounds file
<code>upperindex</code>	0	attribute index to use for the upper bound in a bounds file
<code>lowervalue</code>	*	lower bound value for the entire mesh
<code>uppervalue</code>	$-10^6$	upper bound value for the entire mesh

\* Defaults for these parameters depend on the physical property type. See more information below.

### 2.4.1 Specifying the physical property model

`proptype`

- The following are the most commonly used properties:
  - `den` density; required for vertical gravity and gravity gradiometry data
  - `slo` slowness; required for seismic first arrival travel times
  - `sus` magnetic susceptibility; used for inverting magnetic data with the `"sus"` model type if using the `"amp"` data type then this is the effective magnetic susceptibility (total magnetization amplitude divided by Earth's field strength)
- The following properties are used for inverting magnetic data for with the Cartesian `"pst"` magnetization vector inversion (MVI) model type:
  - `mvp` total magnetization vector  $p$  component (generally specified parallel to direction of Earth's field)
  - `mvs` total magnetization vector  $s$  component (generally specified perpendicular to direction of Earth's field)
  - `mvt` total magnetization vector  $t$  component (generally specified perpendicular to direction of Earth's field)
- The following properties are used for inverting magnetic data with the spherical `"atp"` MVI model type:
  - `sus` effective magnetic susceptibility
  - `inc` total magnetization vector inclination (phi angle)
  - `dec` total magnetization vector declination (theta angle)

`mmul`, `madd`, `mtrend`

- If  $m$  is the model value in a particular cell then the physical property value,  $p$ , used for that cell is calculated as

$$p = \text{mmul} * m + \text{madd} + \text{mtrend} * (z - z_0)$$

where  $z$  is the depth of the cell and  $z_0$  is the depth of the top of the modelling mesh.

- Rearranging the equation above gives:  $m = (p - \text{madd} - \text{mtrend} * (z - z_0)) / \text{mmul}$

**which can help you set bounds, initial values, reference values, and joint coupling cluster coordinates, which must all be set in terms of the scaled physical property  $m$ .**

- Typically, one has some background physical property value and is interested in the anomalous property relative to that value: set `madd` to the background value.
- If you want to rescale the physical property values then `mmul` can be used.

### 2.4.2 Specifying depth/distance/sensitivity weighting for inversion

`wmode`

- Depth/distance/sensitivity weighting model.
- Set to `"none"`, `"depth"`, `"distance"` or `"sensitivity"`.

`wbeta`, `wnorm`, `wpower`, `wzero`

- The depth weighting for the  $j^{th}$  cell is like

$$w_j = (d_j + h_0)^{-\text{wpower}}$$

where  $d_j$  is the *depth* (+ down) of the cell centroid and  $h_0$  is, for example, the average survey *height* (+ up). That is the description given by [Li & Oldenburg \(1998a\)](#) and I find it a bit complicated given that one quantity is defined positive-down and another positive-up. Thinking of it another way, the depth weighting should be

$$w_j = |z_j - z_0|^{-\text{wpower}}$$

where  $z_j$  is the  $z$  coordinate of the cell centroid and  $z_0$  is, for example, the average  $z$  coordinate of the survey observation locations. Looking at the depth weighting in this latter way, you should see that subtracting  $z_0$  calculates the required distance  $|z_j - z_0|$  between the cell centroid and the average survey elevation.

- For depth weighting, **wzero** should define the average survey elevation in the input coordinate system. For example, if the topography surface is flat and lies at  $z = 0$ , and the survey data are all above the topography surface (i.e. airborne or ground data, no downhole data), then:
  - if the input coordinate system has  $+z$  up then the **wzero** value specified in the file should be positive
  - if the input coordinate system has  $+z$  down then the **wzero** value specified in the file should be negative.
- The distance weighting for the  $j^{th}$  cell is like

$$w_j = \left( \sum_{i=1}^N |r_i + \text{wzero}|^{(-\text{wpower} * \text{wnorm})} \right)^{(\text{wbeta} / \text{wnorm})}$$

where the sum is over all data observations, and  $r_i$  is the distance between the cell centroid and the  $i^{th}$  observation location. See [Li & Oldenburg \(2000b\)](#) for more information, although I do things slightly differently (I do not integrate over the volume of the cell - testing has shown this to be of minimal importance).

- For distance weighting, **wzero** should be some small value, such as half the smallest cell dimension.
- The sensitivity weighting for the  $j^{th}$  cell is like

$$w_j = \left( \sum_{i=1}^N \left| \frac{G_{ij}}{v_j} \right|^{\text{wnorm}} \right)^{(\text{wbeta} / \text{wnorm})} = v_j^{-\text{wbeta}} \left( \sum_{i=1}^N |G_{ij}|^{\text{wnorm}} \right)^{(\text{wbeta} / \text{wnorm})}$$

where the sum is over all data observations (all elements of the  $j^{th}$  column of the sensitivity matrix),  $G_{ij}$  is an element in the sensitivity matrix and  $v_j$  is the cell volume. See [Li & Oldenburg \(2000b\)](#) for more information, although I do things slightly differently (I normalize by cell volume - testing has shown this to be extremely important when cell volumes are not constant across the mesh).

- When weights are applied to mesh faces, the above equations for depth and distance weighting hold but cell centroids are replaced by face centroids. For sensitivity weighting, if the  $j^{th}$  face is between cells  $a$  and  $b$  then the weight used for the cell is:

$$w_j = \left( \frac{1}{2} \sum_{i=1}^N \left| \frac{G_{ia}}{v_a} \right|^{\text{wnorm}} + \frac{1}{2} \sum_{i=1}^N \left| \frac{G_{ib}}{v_b} \right|^{\text{wnorm}} \right)^{(\text{wbeta} / \text{wnorm})}$$

#### obsfile

- The observation locations from this **.node** file are used if **wmode** specifies distance weighting.
- **obsfile** should typically specify the data file used in the inversion but I'm providing you the flexibility here to do whatever crazy things you like.

#### datatype

- The sensitivity matrix for this data type is used if **wmode** specifies sensitivity weighting.
- **datatype** should typically be consistent with **proptype**, e.g. gravity data combined with density, but I'm providing you the flexibility here to do whatever crazy things you like.

#### zdir

- The **zdir** parameter only relates to (defines the coordinate system of) the **obsfile** and **wzero** parameters. These are only relevant if distance or depth weighting is used.
- For 3D, set **zdir** > 0 to specify  $+z$  up,  $+x$  East,  $+y$  North;  
 set **zdir** < 0 to specify  $+z$  down,  $+x$  North,  $+y$  East.  
 For 2D, the  $+x$  axis is always to the right and **zdir** only specifies the  $+z$  direction. The  $y$  axis is along-strike.

### 2.4.3 Inversion regularization parameters

**initfile, initindex, initvalue**

- The **initindex** column in the **initfile** (.ele or UBC-GIF file) is used as the initial model.
- If **initfile** is specified as "null" or **initindex** ≤ 0 then the initial model value is **initvalue** for the entire mesh.
- If the initial model specified does not honour all bound constraints then the model is projected such that it honours the bounds.
- If not specified in the physical property input file, the default for **initvalue** is  $10^{-8}$  for conductivity and 0.0 for all other physical property models.

**reffile, reffindex, refvalue**

- The **reffindex** column in the **reffile** (.ele or UBC-GIF file) is used as the reference model.
- If **reffile** is specified as "null" then the reference model value is **refvalue** for the entire mesh.
- If not specified in the physical property input file, the default for **refvalue** is  $10^{-8}$  for conductivity and 0.0 for all other physical property models.

**alphas, wsfile, wsindex**

- The **alphas** value and the weights in any **wsfile** are combined/compounded (one does not override the other).
- The **wsindex** column in the **wsfile** (.ele or UBC-GIF file) is used for the smallness weights.
- If **wsfile** is specified as "null" then the smallness weights are 1.0 for the entire mesh.

### 2.4.4 Constraint options

**boundsfile, lowerindex, upperindex, lowervalue, uppervalue**

- The **lowerindex** and **upperindex** columns in the **boundsfile** (.ele or UBC-GIF file) are used for the lower and upper bounds.
- If **boundsfile** is specified as "null" then the lower and upper bounds are **lowervalue** and **uppervalue** for the entire mesh.
- If a **boundsfile** is specified (any file name other than "null") and **lowerindex** or **upperindex** is non-positive then the specified values of **lowervalue** and **uppervalue**, respectively, are used for the entire mesh. In this way, you can have, for example, a heterogeneous upper bound specified in a **boundsfile** but the lower bound specified as some constant value for the entire mesh (or vice versa).
- If not specified in the physical property input file, the default for **lowervalue** is  $10^{-8}$  for conductivity and 0.0 for all other physical property models.

## 2.5 Joint coupling measure files

These files tell the inversion program VIDI (formerly VINV) how to couple the multiple physical property models in a joint inversion. They are also required if you want to add some additional regularization, based on a joint coupling measure, to a single physical property inversion.

Each line of the input file should be of the format

**name value**

where **name** is the name of some modelling parameter and **value** is the value for that parameter. The possible parameters and default values are listed below. If a parameter is not found in the input file then the default value is used for that parameter. Note that some of the parameters are for use in inversion and are ignored for forward modelling purposes.

Lines in the input file beginning with the # or ! character are ignored and can be used as comments for your own reference.

Name	Default	Brief description
<b>coupling</b>	"null"	specifies the type of coupling
<b>proptype1</b>	"null"	specifies a property type for the first model in the coupling
<b>proptype2</b>	"null"	specifies a property type for the second model in the coupling
<b>proptype3</b>	"null"	specifies a property type for the third model (only used for FCM clustering)

Name	Default	Brief description
<code>rho</code>	0.0	final multiplier value for the joint measure
<code>nsteps</code>	1	number of lambda steps over which to heat the rho value
<code>wjvalue</code>	1.0	constant coupling weight value (applied across entire mesh)
<code>wjfile</code>	"null"	optional file containing cell-centred coupling weights
<code>wjindex</code>	1	attribute index for the <code>wjfile</code> for the coupling weights
OPTIONS FOR LINEAR COUPLING		
<code>lina1</code>	1.0	see explanation below
<code>linb1</code>	0.0	see explanation below
<code>lina2</code>	1.0	see explanation below
<code>linb2</code>	0.0	see explanation below
OPTIONS FOR CORRELATION COUPLING		
<code>issqr</code>	"t"	set to false ("f") if you want to specify a positive or negative correlation
<code>pn</code>	0	the sign specifies a positive or negative correlation (only used if <code>issqr</code> is false)
OPTIONS FOR FUZZY C-MEANS COUPLING		
<code>fcmf</code>	2.0	an exponential power used in the fuzzy c-means (FCM) joint measure
<code>fcmd</code>	"c"	specifies the distance measure to use for the FCM joint measure
<code>slopes</code>	""	linear model slope parameters
<code>intercepts</code>	""	linear model intercept parameters
OPTIONS FOR FUZZY C-MEANS OR GAUSSIAN PDF COUPLING		
<code>wjconstant</code>	"f"	if true then the weights in the <code>wjfile</code> are treated as constants
<code>wj10</code>	"t"	determines how the weights in the <code>wjfile</code> are interpreted
<code>nclusters</code>	0	number of clusters
<code>centres1</code>	""	cluster centres for first physical property
<code>centres2</code>	""	cluster centres for second physical property
<code>centres3</code>	""	cluster centres for third physical property (only used for FCM clustering)
<code>spreads1</code>	""	cluster spreads for first physical property
<code>spreads2</code>	""	cluster spreads for second physical property
<code>rotations</code>	""	cluster rotations in degrees
<code>variances1</code>	""	cluster variances for first physical property
<code>variances2</code>	""	cluster variances for second physical property
<code>covariances</code>	""	cluster covariances for the two physical properties
OPTIONS FOR SUMMATIVE GRADIENT COUPLING		
<code>epsilon</code>	1.0E-6	small value to help avoid division by zero
OPTIONS FOR JOINT TOTAL VARIATION (JTV) COUPLING		
<code>epsilon</code>	1.0E-6	small value to help avoid division by zero
<code>power</code>	1.0	exponent in the JTV coupling

### 2.5.1 Coupling options (parameter coupling)

#### eq, equal

The equal coupling option specifies that the two models should be equal:

$$\mathbf{m}_1 = \mathbf{m}_2 \quad (2.1)$$

#### eqgrad, equalgrad

The equal gradient coupling option specifies that the spatial gradients of the two models should be equal:

$$\mathbf{G}\mathbf{m}_1 = \mathbf{G}\mathbf{m}_2 \quad (2.2)$$

where  $\mathbf{G}$  is the spatial gradient operator.

#### lin, linear

The linear coupling option specifies that there is a known linear relationship between the two models:

$$a_1\mathbf{m}_1 + b_1 = a_2\mathbf{m}_2 + b_2 \quad (2.3)$$



### lingrad, lineargrad

The linear gradient coupling option specifies that there is a known linear relationship between the spatial gradients of the two models:

$$a_1 \mathbf{Gm}_1 + b_1 = a_2 \mathbf{Gm}_2 + b_2 \quad (2.4)$$

### corr, correlation

The correlation coupling option specifies that there is some linear relationship between the two models but the linear parameters of that relationship are unknown. See [Lelièvre et al. \(2012\)](#) for the mathematics. In this case, if you don't know anything about the relationship then you will set parameter `issqr` to "t" (true). If you know that the linear relationship should have a positive or negative slope then set `issqr` to "f" (false) and use parameter `pn` to specify a positive or negative correlation.

### corrgrad

The gradient correlation coupling option specifies that there is some linear relationship between the spatial gradients of the two models but the linear parameters of that relationship are unknown.

### cross, crossgrad

The cross-gradient structural coupling measure compares the spatial gradients of the two models. See [Gallardo & Meju \(2004\)](#) and [Fregoso & Gallardo \(2009\)](#) for more information. At any specific location in the modelling mesh, a zero cross-gradient measure can indicate that the two models have gradients in the same direction, either parallel or antiparallel, or it can indicate that one or both models has zero gradient. As such, this measure suffers from the issue of multiple local minima and it should be used with care to avoid local minima entrapment. This measure is currently only implemented in 2D.

### fcm, fuzzy

The fuzzy c-means (FCM) measure is used to specify different clusters in a physical property cross-plot. See [Lelièvre et al. \(2012\)](#), [Carter-McAuslan et al. \(2015\)](#) and [Sun & Li \(2017\)](#) for the mathematics and usage suggestions. The FCM measure also suffers from the issue of multiple local minima and should be used with care to avoid local minima entrapment.

There are three different options for the distance measure used in the FCM coupling. Set parameter `fcmd` as follows:

- "c" for circular clusters (Euclidean distance); cluster centres must be specified (see [Section 2.5.3](#) below)
- "e" for elliptical clusters (Mahalanobis distance); cluster centres, spreads and rotations must be specified, or centres, variances and covariances (see [Section 2.5.3](#) below)
- "l" for a linear regression relationship; the linear regression parameters must be specified (see [Section 2.5.3](#) below).

### gaus, gauss, gaussian

This coupling measure is formed from a combination of Gaussian functions. This is a similar measure to the fuzzy c-means option but is not as "fuzzy", meaning it suffers even more from the issue of multiple local minima and should be used with even more care to avoid local minima entrapment.

### sumgrad

The summative gradient measure is effectively a normalized version of the equal gradient measure:

$$\frac{\mathbf{Gm}_1}{|\mathbf{Gm}_1|^2 + \text{epsilon}^2} = \frac{\mathbf{Gm}_2}{|\mathbf{Gm}_2|^2 + \text{epsilon}^2} \quad (2.5)$$

### jtv, tvar, totvar

The joint total variation minimizes the following constraint vector:

$$\left( (\mathbf{Gm}_1)^2 + (\mathbf{Gm}_2)^2 + \text{epsilon}^2 \right)^{\text{power}/2} \quad (2.6)$$

## 2.5.2 Weighting the coupling

The **wjvalue** is always applied, in addition to any other weights specified through the **wjfile**. This can be useful if, for example, the values in the coupling constraint vector are very small (as can happen with the cross-gradient measure) but you'd rather work with a scaled version of those. This scaling weight is applied inside the joint coupling calculation, before the **rho** value is applied.

If a **wjfile** is not specified, or equal to "null", then the joint coupling measure will be applied across the entire modelling volume with equal weight throughout.

If the mesh is unstructured then the **wjfile** can be a **.ele** file or a column-based data file. If the mesh is rectilinear then the **wjfile** must be a column-based data file. The number of cells in the **.ele** file, or the number of rows in the column-based data file, must equal the number of cells in the inversion mesh.

For the **equal**, **lin** and **cross** coupling options, there is only a single weight required for each cell. The **wjindex** parameter specifies an attribute index for the **wjfile** and the coupling weights are taken from that attribute column. The weights are applied to the coupling measurement for each cell. For example, the **equal** measure becomes

$$\Psi_{equal} = \sum_i w_i (m_{2,i} - m_{1,i}) \quad (2.7)$$

where the sum is over every cell in the mesh.

For the **corr** coupling option, there is also only a single weight required for each cell but the weights are used differently. The weights are used to specify which region of the mesh should be included in the coupling. Cells with  $w_i = 0$  are ignored. Any non-zero weighting value  $w_i$  can be used to indicate that the  $i^{th}$  cell should be included in the coupling. If you want different implicit linear relationships in different regions of your model then you will need different coupling files and associated weighting files for each region.

For the **fcm** coupling option there can be a single weight specified for each cell or multiple weights for each cell.

- If **wjconstant** is "t" (true) then the supplied weights are treated as constant membership weights and parameter **wj10** is not used. The **wjfile** and **wjindex** parameters should specify the weights  $w_{ij}$  for each cell and each cluster. Weight  $w_{ij}$  is the membership of the  $i^{th}$  cell for the  $j^{th}$  cluster.  $N_c$  attributes are required in the **wjfile**, where  $N_c$  is the number of clusters defined in the [physical property file\(s\)](#). The **wjindex** parameter specifies the attribute column to start taking the attributes from: attributes are taken from columns **wjindex** to **wjindex**+ $N_c$ -1. Each attribute column contains the information for a different cluster; the order of the attributes in the **wjfile** should be consistent with the order of the clusters defined in the [physical property file\(s\)](#).
- If **wjconstant** is "f" (false, the default) then the supplied weights are used to specify which region of the mesh should be included in the coupling. The values in the **wjindex** file are not actually treated as coupling weights but simply determine which cells are attached to which clusters. The actual membership weights are determined automatically as part of the joint inversion process. Parameter **wj10** is treated as follows.
  - If **wj10** is "t" (true, the default) then the **wjfile** and **wjindex** parameters should be specified as above for **wjconstant** equal to "t" (true). However, cells with  $w_{ij} = 0$  are ignored. Any non-zero value  $w_{ij}$  can be used to indicate that the  $i^{th}$  cell should be coupled with the  $j^{th}$  cluster.
  - If **w10** is "f" (false) then only a single attribute column is required in the **wjfile**. The **wjindex** parameter should specify the attribute column to use. The single attribute column should contain integer indices on  $[1, N_c]$  that specify which cells should correspond to which cluster.

For the **gauss** coupling option, the measure is a Gaussian mixture,

$$\Psi_{gauss} = \sum_i \sum_j w_{ij} N(m_{1,i}, m_{2,i}, \dots | \mu_j, \sigma_j), \quad (2.8)$$

and the **wjfile** and **wjindex** parameters should be specified as described above for the **fcm** measure with **wjconstant** equal to "t" (true).

## 2.5.3 Defining clusters

**centres1**, **centres2**, **centres3**, **spreads1**, **spreads2**, **rotations**, **variances1**, **variances2**, **covariances**

These are string buffers specifying the cluster information. For example, if on a density-vs-susceptibility plot there are three clusters at (0.0,0.0), (1.0,0.3) and (2.0,0.5) then you will have the following lines in the coupling measure input file:

```

proptype1 den
proptype2 sus
nclusters 3
centres1 "0.0 1.0 2.0"
centres2 "0.0 0.3 0.5"

```

The cluster centres, spreads and variances/covariances should be specified in terms of the scaled physical property: see the physical property input file parameters `mmul`, `madd` and `mtrend` explained in [Section 2.4.1](#).

Below, two options are explained for specifying elliptical clusters rotated with respect to the two physical property axes. Option 1 is always used if the `rotations` parameter is specified, in which case the `variances1`, `variances2` and `covariances` information is ignored.

### Option 1

You can specify the spreads and rotations (`spreads1`, `spreads2`, `rotations`) of the clusters. A positive rotation moves the first physical property axis towards the second. That is, counter-clockwise if the first and second properties are plotted on the  $x$  and  $y$  axes respectively, with  $x$  right and  $y$  up the page. The “spreads” are standard deviations for the two axes of the rotated ellipses. A matrix is calculated as follows:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \quad (2.9a)$$

$$a_{11} = \frac{\cos^2 \theta}{\sigma_1^2} + \frac{\sin^2 \theta}{\sigma_2^2} \quad (2.9b)$$

$$a_{22} = \frac{\sin^2 \theta}{\sigma_1^2} + \frac{\cos^2 \theta}{\sigma_2^2} \quad (2.9c)$$

$$a_{12} = \frac{\sin 2\theta}{2\sigma_1^2} - \frac{\sin 2\theta}{2\sigma_2^2} \quad (2.9d)$$

where  $\sigma_1$ ,  $\sigma_2$  and  $\theta$  are values pulled from the `spreads1`, `spreads2` and `rotations` lists respectively. That matrix is used to calculate the Mahalanobis distance:

$$\begin{aligned} & (\mathbf{p} - \boldsymbol{\mu})^T \mathbf{A} (\mathbf{p} - \boldsymbol{\mu}) \\ &= \begin{bmatrix} p_1 - \mu_1 & p_2 - \mu_2 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} p_1 - \mu_1 \\ p_2 - \mu_2 \end{bmatrix} \\ &= a_{11} (p_1 - \mu_1)^2 + 2a_{12} (p_1 - \mu_1) (p_2 - \mu_2) + a_{22} (p_2 - \mu_2)^2 \end{aligned} \quad (2.10)$$

where  $\mathbf{p}$  is a vector holding the two physical property values in a mesh cell and  $\boldsymbol{\mu}$  is a vector holding the means for the two physical properties. Those means are pulled from the `centres1` and `centres2` lists. The Mahalanobis distance appears in a Gaussian distribution  $f$  as follows:

$$f(\mathbf{p}) = \exp \left( -\frac{1}{2} (\mathbf{p} - \boldsymbol{\mu})^T \mathbf{C}^{-1} (\mathbf{p} - \boldsymbol{\mu}) \right). \quad (2.11)$$

See [https://en.wikipedia.org/wiki/Gaussian\\_function#Two-dimensional\\_Gaussian\\_function](https://en.wikipedia.org/wiki/Gaussian_function#Two-dimensional_Gaussian_function) and [Sun & Li \(2017\)](#) for more information on the mathematics for the `gauss` and `fcm` coupling options respectively.

### Option 2

If you specify the variances and covariances (`variances1`, `variances2`, `covariances`) of the clusters then a covariance matrix is generated for each cluster:

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} \\ c_{12} & c_{22} \end{bmatrix} \quad (2.12)$$

where  $c_{11}$ ,  $c_{22}$  and  $c_{12}$  are values pulled from the `variances1`, `variances2` and `covariances` lists respectively. The covariance matrix is inverted and used to calculate the Mahalanobis distance:

$$(\mathbf{p} - \boldsymbol{\mu})^T \mathbf{C}^{-1} (\mathbf{p} - \boldsymbol{\mu}). \quad (2.13)$$

See [https://en.wikipedia.org/wiki/Mahalanobis\\_distance](https://en.wikipedia.org/wiki/Mahalanobis_distance) and [https://en.wikipedia.org/wiki/Multivariate\\_normal\\_distribution#Density\\_function](https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Density_function) for more information on the mathematics.

### 2.5.4 Single property inversion

If you want to apply some additional regularization to a single physical property inversion, for example to apply some clustering to the recovered single physical property, then use `proptype1` to specify the single property and leave `proptype2` set to `null` (the default if absent).

## 2.6 Regions files

These files specify different regions within a wireframe model. The format is like at the end of a `.poly` file:

```
nreg ndim nat nvol
1 x1 [y1] z1 [attribute] [volume]
2 x2 [y2] z2 [attribute] [volume]
...
n xn [yn] zn [attribute] [volume]
```

where

- `nreg=n` is the number of regions (the number of lines that follow in the file)
- `ndim` is the number of dimensions (2 or 3)
- `nat` is the number of attributes (0 or 1)
- `nvol` is the number of volumes (0 or 1)
- the `(xi,[yi],zi)` coordinates specify points within each region.

Refer to the [TetGen](#) or [Triangle](#) documentation for further information on the `.poly` file format.

Any line beginning with `#` is a comment line and is ignored. Fully commented lines may not appear within the list of regions but comments may appear at the end of any data line.

The attribute and volume columns are only required if `nat=1` and `nvol=1` respectively. The volumes are used for specifying minimum volume constraints when meshing. The file format only allows one attribute. If you want more than one attribute it would be better to use the attributes to specify rock-unit ID values that can then get linked to the attribute values in a rock-units file: see program [rockunits2ele](#).

## 2.7 Rock-units files

These files specify different attributes (e.g. physical property values) for different rock units. The format is as follows:

```
nunits nat
ID1 a11 a12 ...
ID2 a21 a22 ...
...
name1
name2
...
```

where

- `nunits` is the number of units
- `nat` is the number of attributes
- each row after the first specifies the unit ID and the attribute values for that unit:
  - `IDi` is the  $i^{th}$  unit ID (a unique integer)
  - `aij` is the  $j^{th}$  attribute for the  $i^{th}$  unit (e.g. a real value)
- the final several lines contain the attribute names:
  - `namej` is the name for the  $j^{th}$  attribute.

Any unique integer values can be used for the unit ID's. However, some programs will run faster if they are sequential from 1.

The names specified for the rock unit names may need to be set to exactly those required for a specific purpose.

## 2.8 Unstructured block files

These files specify a number of blocks in a simple block model, destined for use in an unstructured mesh. The format is as follows:

```
n nbb ndim nat nvol
xc1 [yc1] zt1/zc1 dx1 [dy1] dz1 [phi1] theta1 [psi1] [a1] [v1]
xc2 [yc2] zt1/zc2 dx2 [dy2] dz2 [phi2] theta2 [psi2] [a2] [v2]
...
xcn [ycn] ztn/zcn dxn [dyn] dzn [phin] thetan [psin] [an] [vn]
```

where

- **n** is the number of blocks (the number of lines that follow in the file)
- **nbb** is the number of boundary blocks (usually 1 for simple models)
- **ndim** is the number of dimensions (2 or 3)
- **nat** is the number of attributes (0 or 1)
- **nvol** is the number of volumes (0 or 1)
- (**xc1**, **[yc1]**, **zt1**) specifies the LATERAL CENTRE OF THE TOP of any boundary blocks (*y* coordinates are not used in 2D)
- (**xc1**, **[yc1]**, **zci**) specifies the CENTROID of any non-boundary blocks
- the **dx**, **dy**, **dz** quantities define the dimensions of the block
- **phi**, **theta** and **psi** are the strike, dip and tilt angles respectively, in degrees, defined as in [Li & Oldenburg \(2000a\)](#) and [Lelièvre & Oldenburg \(2009\)](#) (only dip, **theta**, is required for 2D problems)
- **ai** is the attribute value for the  $i^{th}$  block
- **vi** is a volume for the  $i^{th}$  block (used as a maximum cell volume constraint for meshing programs [Triangle](#) and [TetGen](#)).

Alternatively, the block specification lines can be in this format:

```
A x1 x2 [y1 y2] z1 z2 [phi] theta [psi] [attribute] [volume]
```

where that “A” character must exist at the start of the line.

All boundary blocks should be specified first, before any non-boundary blocks. Hence, for the first **nbb** blocks, the *z* value supplied is assumed to be the lateral centre of the top of the boundary blocks; for the other blocks, the *z* value supplied is assumed to be that of the centroid of the non-boundary blocks.

The coordinates in the blocks file are assumed to have +*x* East, +*y* North, +*z* up for 3D and +*x* right, +*z* up for 2D.

# Chapter 3

## Utilities for file format conversion

These are simple programs that convert from one file format to another. Note that some file conversions may require running multiple programs in succession. For example, to convert a UBC-GIF format gravity data observations file to a `.vtu` file for opening in [ParaView](#), you must use program `xyz2node` followed by `node2vtu`.

### 3.1 Unstructured mesh file formats

These programs work with file formats used by the TetGen and Triangle meshing programs, and by ParaView. These file formats are explained in more detail on these webpages:

- Triangle:  
<http://www.cs.cmu.edu/~quake/triangle.html>
- TetGen:  
<http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual006.html#sec69>
- VTK file formats  
<http://www.vtk.org/VTK/img/file-formats.pdf>

#### 3.1.1 cells2nodes

Moves a model on an unstructured mesh of cells to a cloud of nodes. Reads unstructured mesh information from a pair of `.node` and `.ele` files and writes a `.node` file with node attributes taken from the mesh cell attributes. This program is useful for creating contour and vector plots in ParaView or GMT. To use a model on a rectilinear mesh instead, first use program `ubcgif2mesh` to convert to an unstructured mesh.

Command line usage:

```
cells2nodes noderoot eleroot outroot [centred [zrev]]
```

Command line parameters:

- Mesh information is read from `noderoot.node` and `eleroot.ele`.
- The output file is named `outroot.node`.
- The optional `centred` parameter determines where the nodes are:  
if `t` (true) then the nodes in the output files are at the cell centres and the cell attributes are simply copied over;  
if `f` (false, default) then the nodes in the output files are at the original node positions and the cell attributes are interpolated using distance-weighted averages from the surrounding cells (distance between node and cell centroids).
- Set the optional `zrev` parameter to `t` (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.

#### 3.1.2 ele2ele

Reads a `.ele` file and re-writes it. Use this program when you want to have the same precision in files generated from a meshing program and from other PODIUM programs.

Command line usages:

```
ele2ele inroot  
ele2ele inroot outroot
```

Command line parameters:

- Reads cell information from `inroot.ele`.
- Writes cell information to `outroot.ele`. If the optional `outroot` parameter is absent then file `inroot.ele` is overwritten.

### 3.1.3 ele2vtu

Converts a `.ele` file to a 2D `.vtu` file for viewing in ParaView. The coordinates for the `.vtu` file are taken from the first and second columns of indices in the `.ele` file. Use this program for viewing traveltime data in ParaView.

Command line usages:

```
ele2vtu inroot
ele2vtu inroot outroot
```

Command line parameters:

- Reads information from `inroot.ele`.
- Writes information to `outroot.vtu`. If the optional `outroot` parameter is absent then file `inroot.vtu` is overwritten.

### 3.1.4 mesh2neigh

Reads information about an unstructured mesh defined by `.node` and `.ele` files, and writes a `.neigh` file containing cell neighbour information.

Command line usages:

```
mesh2neigh noderoot eleroot neighroot
mesh2neigh meshroot
```

Command line parameters:

- Mesh information is read from `noderoot.node` and `eleroot.ele` for the first command line usage, or from `meshroot.node` and `meshroot.ele` for the second command line usage.
- Cell neighbour information is written to `neighroot.neigh` for the first command line usage, or to `meshroot.neigh` for the second command line usage. For the second command line usage, any path information at the start of the `meshroot` parameter is snipped off before writing the output `neighroot.neigh` file, so the output file will be located in the directory where this program is run from.

### 3.1.5 mesh2poly

Converts an unstructured mesh defined by `.node` and `.ele` files to a `.poly` file for use with the TetGen and Triangle meshing programs.

Command line usages:

```
mesh2poly noderoot eleroot regionsroot polyroot [bm]
mesh2poly meshroot [bm]
```

Command line parameters:

- Mesh information is read from `noderoot.node` and `eleroot.ele` for the first command line usage, or from `meshroot.node` and `meshroot.ele` for the second command line usage.
- Regions information is read from `regionsroot.node` for the first command line usage, or from `meshroot_regions.node` for the second command line usage. If `regionsroot` is null or file `meshroot_regions.node` does not exist then regions are not read from file and the output `.poly` file will specify no regions.
- Mesh information is written to `polyroot.poly` for the first command line usage, or to `meshroot.poly` for the second command line usage. For the second command line usage, any path information at the start of the `meshroot` parameter is snipped off before writing the output `meshroot.poly` file, so the output file will be located in the directory where this program is run from.
- Supply the optional `bm` parameter to specify the integer boundary marker value to use for the facets. If the `bm` parameter is absent then the boundary markers are not altered.

### 3.1.6 mesh2tikz

Converts `.node` and `.edge` files to a `.ltx` file containing plotting information for typesetting with L<sup>A</sup>T<sub>E</sub>X using the TikZ package.

Command line usages:

```
mesh2tikz inroot bm
mesh2tikz inroot bm outroot
```

Command line parameters:

- Node and edge element information is read from files `inroot.node` and `inroot.edge` respectively.
- If `outroot` is present then file `outroot.ltx` is written, containing the TikZ plotting commands.
- If `outroot` is absent then file `inroot.ltx` is written, containing the TikZ plotting commands.
- If `bm` is `f` (false) then any boundary edge elements are omitted from the output file. Otherwise, all edges are included.

### 3.1.7 mesh2vtu

Converts an unstructured mesh defined by a pair of `.node` and `.ele` files to a `.vtu` file for viewing in ParaView. Additional node and cell attributes are added to the `.vtu` file to indicate node and cell indices, node boundary markers (if present in the `.node` file), and cell centroid coordinates. If the files specify a surface-based model (as opposed to mesh-based) then additional cell attributes are added to the `.vtu` file to indicate the facet normal vectors.

Command line usages:

```
mesh2vtu meshroot
mesh2vtu noderoot eleroot
mesh2vtu noderoot eleroot vturoot
mesh2vtu noderoot eleroot vturoot zrev
mesh2vtu noderoot eleroot vturoot ifld1 ifld2
mesh2vtu noderoot eleroot vturoot ifld1 ifld2 ifld3
```

Command line parameters:

- Mesh information is read from files `meshroot.node` and `meshroot.ele` (first usage above) or from `noderoot.node` and `eleroot.ele` (last three usages above).
- Mesh information is written to file `vturoot.vtu`.
- If the `vturoot` parameter is absent then `vturoot` is set to `eleroot` if present and to `meshroot` otherwise.
- Set the optional `zrev` parameter to `t` (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.
- For 3D problems, if `ifld1`, `ifld2` and `ifld3` are provided then they specify cell attribute indices representing a vector field in a +z up system: `ifld1` is the component towards east, `ifld2` is the component towards north, and `ifld3` is the component vertically upwards.
- For 2D problems, if `ifld1` and `ifld2` are provided then they specify cell attribute indices representing a vector field in a +z up system: `ifld1` is the component right (along profile), `ifld2` is the component vertically upwards.

Outputs:

- If the files specify a mesh-based model and the `ifld1/2/3` parameters are provided then output file `vturoot.vtu` contains a vector cell attribute named “CellVector” specifying that cell vector field.
- If the files specify a surface-based model then output file `vturoot.vtu` contains a vector cell attribute named “CellVector” that indicates the facet normal vectors.
- To view the vectors in ParaView:
  1. load the `.vtu` file into Paraview



2. create a Glyph: one of the buttons in the same basic toolbar as the clip, slice and threshold tools; it looks like a globe with dots on its surface
3. In the glyph properties tab:
  - (a) under “Glyph Source” set the “Glyph Type” to “Arrow” (which may be the default)
  - (b) under “Active Attributes” set “Vectors” to “CellVectors”
  - (c) under “Masking” set “Glyph Mode” to “All Points”

You should then see arrows indicating the cell normals, defined by a right-hand-rule moving along/around the facet nodes: fingers curving around the facet following the node ordering and the thumb then indicating the vector direction.

### 3.1.8 node2d

Changes the header (first line) in a 3D `.node` file to make it specify a 2D object. The number of attributes is increased by one, essentially changing the third coordinate column (elevations) into the first attribute column.

Command line usage:

```
node2d fileroot outroot
```

Command line parameters:

- Writes a copy of file `fileroot.node` named `outroot.node` with the first line changed.

### 3.1.9 node3d

Changes the header (first line) in a 2D `.node` file to make it specify a 3D object. The number of attributes is decreased by one, essentially changing the first attribute column into the third coordinate column (elevation). If there are no attribute columns then the boundary markers become the elevation column. If there are neither attributes nor boundary markers then an error message is thrown.

Command line usage:

```
node3d fileroot outroot
```

Command line parameters:

- Writes a copy of file `fileroot.node` named `outroot.node` with the first line changed.

### 3.1.10 node2node

Reads a `.node` file and re-writes it. Use this program when you want to have the same precision in files generated from a meshing program and from other PODIUM programs.

Command line usages:

```
node2node inroot
```

```
node2node inroot outroot
```

```
node2node inroot outroot d
```

Command line parameters:

- Reads node information from `inroot.node`.
- Writes node information to `outroot.node`. If the optional `outroot` parameter is absent then file `inroot.node` is overwritten.
- If the optional `d` parameter is provided then each node is randomly moved (uniform random distribution) in Cartesian space, at most a distance `d` in each direction.

### 3.1.11 node2vtu

Converts a `.node` file to a `.vtu` file for viewing in ParaView. Attributes are added to the `.vtu` file to indicate node indices, elevation and line number (if present in the `.node` file).

Command line usages:

```

node2vtu noderoot
node2vtu noderoot zrev
node2vtu noderoot zrev ix [iy] iz
node2vtu noderoot zrev ix [iy] iz zrev2
node2vtu noderoot vturoot
node2vtu noderoot vturoot zrev
node2vtu noderoot vturoot zrev ix [iy] iz
node2vtu noderoot vturoot zrev ix [iy] iz zrev2

```

Command line parameters:

- Reads node information from a file named **noderoot.node**.
- Writes node information to a file named **vturoot.vtu**. If **vturoot** is absent from the command line usage then the node information is written to a file named **noderoot.vtu**.
- Set the optional **zrev** parameter to **t** (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.
- **ix**, **[iy]** and **iz** can be used to indicate node attribute indices for a vector field. Only a single vector field can currently be written to the **.vtu** file. Omit the **iy** value for a 2D file.
- **zrev2** applies only to the vector field components. The default behaviour, with **zrev2** set to **"t"** (true), is to switch the coordinate system of the field components. Hence, by default, they field components are assumed to be provided in a +z-down coordinate system. Set **zrev2** to **"f"** (false) if your scenario is otherwise.

### 3.1.12 poly2mesh

Converts a **.poly** file to a pair of **.node** and **.ele** files. This program currently only supports one polygon per facet and does not support holes.

Command line usages:

```

poly2mesh polyroot
poly2mesh polyroot meshroot

```

Command line parameters:

- PSLG or PLC information is read from file **polyroot.poly**.
- PSLG or PLC information is written to files **meshroot.node** and **meshroot.ele**.
- If the optional **meshroot** parameter is absent then **meshroot** is set to **polyroot**.

### 3.1.13 poly2vtu

Converts a **.poly** file to a **.vtu** file for viewing in ParaView. This program currently only supports one polygon per facet and does not support holes. Additional node and cell attributes are added to the **.vtu** file to indicate node and cell indices.

Command line usages:

```

poly2vtu polyroot
poly2vtu polyroot vturoot
poly2vtu polyroot vturoot zrev

```

Command line parameters:

- PSLG or PLC information is read from file **polyroot.poly**.
- PSLG or PLC information is written to file **vturoot.vtu**.
- If the optional **vturoot** parameter is absent then **vturoot** is set to **polyroot**.

- Set the optional **zrev** parameter to **t** (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.

## 3.2 Peter's file formats

These programs work with some of the file formats above and others that I have devised.

### 3.2.1 blocks2poly

Writes a **.poly** file containing a number of different rectangular or prismatic blocks. Reads a blocks specification file, converts it to an unstructured mesh object and writes the facet information to a **.poly** file.

Command line usage:

```
blocks2poly blocksfile [outroot]
```

Command line parameters:

- Block information is read from **blocksfile**. The block file format is explained in [Section 2.8](#).
- If **outroot** is provided then the PSLG or PLC is written to **outroot.poly**.
- If **outroot** is not provided then the PSLG or PCL is written to **blocksroot.poly** where the **blocksfile** parameter is assumed to be of the form **blocksroot.ext** (the output file is named similarly to the input blocks specification file but with the **.poly** extension).

### 3.2.2 blocks2vtu

Writes a **.vtu** file containing a number of different rectangular or prismatic blocks. Reads a blocks specification file, converts it to an unstructured mesh object and writes the facet information to a **.vtu** file.

Command line usage:

```
blocks2vtu blocksfile [outroot]
```

Command line parameters:

- Block information is read from **blocksfile**. The block file format is explained in [Section 2.8](#).
- If **outroot** is provided then the PSLG or PLC is written to **outroot.vtu**.
- If **outroot** is not provided then the PSLG or PCL is written to **blocksroot.vtu** where the **blocksfile** parameter is assumed to be of the form **blocksroot.ext** (the output file is named similarly to the input blocks specification file but with the **.vtu** extension).

### 3.2.3 rockunits2ele

Reads rock unit information from a file and adds that rock unit information to the attributes columns in a **.ele** file or UBC-GIF format model file.

Command line usages:

```
rockunits2ele modelfile unitfile
rockunits2ele modelfile unitfile outfile
rockunits2ele modelfile unitfile outfile iat
```

Command line parameters:

- Reads cell information from **modelfile**. If the model is on an unstructured mesh then **modelfile** should be a **.ele** file. If **modelfile** does not have the **.ele** extension then a UBC-GIF format model file (rectilinear mesh) is assumed.
- Reads rock unit information from **unitfile**. The rock unit file format is explained in [Section 2.7](#).
- Parameter **iat** specifies the index of the cell attribute column in the **modelfile** to use for the unit ID values. If parameter **iat** is absent then unit ID values are assumed to be in the first cell attribute column.

- Writes new cell information (ID values and the corresponding rock unit attributes) to **outfile**. If the optional **outfile** parameter is absent then file **modelfile** is overwritten including any existing cell attributes.

Notes:

- The ID values read from the specified attribute column in the **modelfile** should be integers. Otherwise, they are rounded to the closest integer.
- For each cell, the ID value read from the **modelfile** is matched to an ID in the **unitfile**. The attributes in the **unitfile** for that ID are transferred to the cell.
- An error occurs if a cell ID value from the **modelfile** is not found in the rock unit information in the **unitfile**.
- A warning is provided if not all ID values defined in **unitfile** are used.

### 3.2.4 rockunits2node

Reads rock unit information from a file and adds that rock unit information to the attributes columns in a **.node** file.

Command line usage:

```
rockunits2node noderoot unitfile [outroot]
```

Command line parameters:

- Reads node information from **noderoot.node**. Unit ID values are assumed to be in the first attribute column.
- Reads rock unit information from **unitfile**. The rock unit file format is explained in [Section 2.7](#).
- Writes new node information (ID values and rock unit attributes) to **outroot.node**. If the optional **outroot** parameter is absent then file **noderoot.node** is overwritten.

An error is thrown if a node ID value is not found in the rock unit information. A warning is thrown if not all ID values defined in **unitfile** are used.

## 3.3 UBC-GIF file formats

See [Section 2.2](#) for more information on UBC-GIF mesh and model file formats.

### 3.3.1 node2xyz

Converts a **.node** file to a simple **.xyz** format file. The **.xyz** file is identical to the **.node** file except the index values in the first column of the **.node** file are not written to the **.xyz** file. You can use this program to convert a **.node** file containing gravity or magnetic measurements to a format that the UBC-GIF programs can work with (you must manually fix the header information after conversion).

Command line usages:

```
node2xyz root
node2xyz root noheader
```

Command line parameters:

- Reads node information from **root.node**.
- Writes node information to **root.xyz**.
- If **noheader** is "t" (true) then no header line is written, if "f" (false, the default) then the header line is written.

### 3.3.2 reorder\_model

Reorders a model on a rectilinear mesh from one ordering system to another.

Command line usage:

```
reorder_model meshfile modelfile outfile ord1 ord2
```

Command line parameters:

- Reads rectilinear mesh and model information from files `meshfile` and `modelfile` respectively. The format is as for UBC-GIF format files but the ordering system for the `modelfile` may be different.
- Parameter `ord1` specifies the ordering system for the input model read from file `modelfile`.
- Parameter `ord2` specifies the ordering system for the output model written to file `outfile`.
- The standard ordering for values in a UBC-GIF format model is to start at the top-most (highest elevation) cell in the southwestern corner of the mesh, move downwards through a vertical column of cells in the mesh, then raster across in the easting direction, then northing. To specify this standard UBC-GIF ordering set `ord1` or `ord2` to "DEN". In a Cartesian coordinate system with  $+z$  down, this would be first along the z-direction, then y-direction, then x-direction.
- The currently supported order specifications for parameters `ord1` and `ord2` are:
  - "DEN" = starting from the top southwest corner, first moving Down, then East, then North.
  - "ESD" = starting from the top northwest corner, first moving East, then South, then Down.
  - "WND" = starting from the top southeast corner, first moving West, then North, then Down.
  - "SED" = starting from the top northwest corner, first moving South, then East, then Down.
  - "NWD" = starting from the top southeast corner, first moving North, then West, then Down.

### 3.3.3 ubcgif2mesh

Converts a UBC-GIF model on a rectilinear mesh to unstructured mesh format.

Command line usages:

```
ubcgif2mesh mesh model outroot
ubcgif2mesh mesh model outroot split
```

Command line parameters:

- Reads rectilinear mesh and model information from UBC-GIF format files `mesh` and `model` respectively. Enter `null` for the `model` parameter if you just want to work with the mesh.
- Writes unstructured mesh and model information to `outroot.node`, `outroot.ele` and `outroot.neigh`.
- Supply the optional `split` parameter if you want to split the rectilinear mesh cells into triangles (2D) or tetrahedra (3D) in an unstructured mesh:
  - a value of 0 splits rectangles symmetrically into four triangles (for 2D only)
  - a value of  $\pm 1$  splits rectangles into two triangles and prisms into five tetrahedra with alternating splitting in neighbours
  - a value of  $\pm 2$  splits rectangles into two triangles with the same splitting everywhere.
 For the latter two options, the sign of `split` determines the geometry of the asymmetrical splitting.

### 3.3.4 ubcgif2vtr

Converts UBC-GIF format mesh and model files to a `.vtr` file for viewing in ParaView.

Command line usages:

```
ubcgif2vtr mesh
ubcgif2vtr mesh model
ubcgif2vtr mesh model outroot
ubcgif2vtr mesh model outroot name
ubcgif2vtr mesh model outroot name logflag
ubcgif2vtr mesh model outroot name i1 i2 i3
```

Command line parameters:

- Reads rectilinear mesh and optional model information from UBC-GIF format files `mesh` and `model` respectively.
- Enter `null` for the `model` parameter if you just want to work with the mesh, or use the first command line usage above.
- Writes the mesh and model information to `outroot.vtr`.

- For the first command line usage above, the root name of `mesh` (with path and extension removed) is used as the `outroot`.
- For the second command line usage above, the root name of `model` is used as the `outroot`.
- If there is only a single model attribute then it will take the name `name` in the output file `outroot.vtr`. If parameter `name` is absent then a default attribute name is used. If there are multiple model attributes then they are given default names and parameter `name` is ignored.
- If `logflag` is `t` (true) then the output holds  $\log_{10}$  model values. The default is `f` (false).
- If the extension of file `model` is `.fld` then a field file is assumed with three columns indicating the vector components in the **Easting, Northing and Downward** directions, in that order (which is the standard UBC-GIF ordering for `.fld` files).
- If the extension of file `model` is something other than `.fld` then a standard model file is assumed with one or more columns indicating the different model attributes. However, if you want to specify that three of those model attributes are vector field components then use the command line parameters `i1`, `i2` and `i3` to specify the vector components in the **Northing, Easting and Downward** directions respectively (which is an “x-y-z” order in my +z down coordinate system).
- Note the discrepancy in the vector component order for the two bullet points above. Hey, it’s the UBC-GIF format that is strange (it’s not even right-handed) so don’t blame me!

### 3.3.5 xyz2csv

Reads a file and replaces sequences of spaces with commas.

Command line usage:

```
xyz2csv inputfile outputfile
```

Command line parameters:

- File `inputfile` is read and the same text is written to file `outputfile` but with any sequences of spaces replaced with commas.

### 3.3.6 xyz2node

Converts a column-based data file (e.g. `.xyz`) to a `.node` format file. This program has many options to enable the use of many possible input file formats.

Command line usages:

```
xyz2node xyzfile nhead hassize ndim hasid csv
xyz2node xyzfile nhead hassize ndim hasid csv [outroot]
```

Command line parameters:

- Reads column-based data from `xyzfile`. Any comment lines starting with the `#` character at the top of the file are skipped.
- An additional optional uncommented header, consisting of `nhead` lines, is skipped and ignored (`nhead` is an integer value).
- If `hassize` is `t` (true) then the number of data and the number of columns are read from the line directly after the header. The number of columns is not required on that line: if absent, `xyz2node` determines the number of columns automatically based on the number of values on the first data line.
- `ndim` is the number of dimensions (2 or 3).
- If `hasid` is `t` (true) then the first column is assumed to hold ID values and is ignored.
- Set `csv` to `t` (true) if the `xyzfile` is in comma-separated-variable format.

Outputs:

- Writes column-based data to `outroot.node`.
- If `outroot` is absent then the file root of `xyzfile` is used (extension stripped off).

Assumptions:

- If `hasid` is `f` then the coordinates are assumed to be in the first (left-most) 2 or 3 columns (2 or 3 depending on the number of dimensions). If `hasid` is `t` then the coordinates are assumed to be in the 2 or 3 columns to the right of the column of ID values.

Notes:

- Reading a large file without the size information, i.e. with `hassize` set to `f` (false), requires that `xyz2node` reads the file twice: once to determine the number of data lines and the number of data columns, and again to read the information into storage. Therefore, I encourage users to insert the size information into their large data files before running `xyz2node`.

Examples:

- To convert a UBC-GIF format gravity data file use  
`xyz2node xyzfile 0 t 3 f f`
- To convert a UBC-GIF format magnetics data file use  
`xyz2node xyzfile 2 t 3 f f`  
(the `nhead` parameter is set to 2 to skip the two lines that specify the geomagnetic field and measurement directions, so that information will not be present in the `.node` file).

### 3.3.7 xyz2xyz

Extracts a single column out of a multiple-column data file and writes it to a single-column data file. This program can be used to extract a single model from my multi-model extension of the UBC-GIF model file format.

Command line usage:

```
xyz2xyz inputfile outputfile i
```

Command line parameters:

- File `inputfile` is read and the  $i^{th}$  data column written to file `outputfile`.

Assumptions:

- `inputfile` has no uncommented header lines
- `inputfile` has the same number of values on every line (after any commented header lines)

## 3.4 Industry file formats

### 3.4.1 convert\_format

Converts from various industry file formats to unstructured mesh file formats, or the reverse.

Command line usages:

```
convert_format inputfile1 outputfile [...]    convert_format inputfile1 inputfile2 outputfile [...]
```

Command line parameters:

- Converts file `inputfile1` (and `inputfile2` if present) to file `outputfile`.
- The program may prompt the user for various information if not present in the optional parameters [...]. Those optional parameters change depending on the conversion being performed. This functionality is only provided for users who wish to perform some batch processing and avoid the user prompts.

Notes:

- The formats of the input and output files are determined automatically from their extensions if possible. The table completely below these bulleted notes indicates the output formats when no extension is supplied. If formats can not be determined from extensions, the program attempts to read minimal information from the files to assess the file type:
  - If the extension is `.txt`:
    - \* If the first line of the file contains the text “CX1”, “IP-DCR”, “DCIP”, “DC/IP” or “DC-IP” then the file is treated as a column-based DC/IP file.
    - \* Failing that, if there is a line that specifies the “type” keyword followed by the equals symbol “=” then the file is treated as a Geomatic file.
    - \* Failing both of those, the file is probably a GEOMAR AUV MAG Survey file but the user is still prompted for further information.

If the file type still cannot be identified after that, the user is prompted for further information.

- If an extension combination is not currently supported then the program provides an error message stating that is the case. Please contact the developers if a desired extension combination is not currently supported.
- Some format conversions assume particular file format conventions so may not work for all files. Please contact the developers if you have problems.
- For ESRI grids, the origin coordinates specified in the file by parameters *xllcorner* and *yllcorner* are assumed to be the lower left corner of the lower left cell. If that is a problem for you, you can use program [transform\\_coordinates](#) to shift the output *.node* file by the required amount. The program also accepts parameters *xllcenter* and *yllcenter* for ESRI grids.
- For conversion of *.node* and *.ele* files to R3t *mesh3d.dat* file or R2 *mesh.dat* file (Andrew Binley's programs), if attribute columns named "param" and "zone" are specified then those are used to specify the *param\_elem(i)* and *zone\_elem(i)* information referred to in the R3t and R2 manuals. You don't need to include both of those columns: if "param" is missing then it defaults to values equal to the cell indices; if "zone" is missing it defaults to values of 1 (one). If "param" is present and equal to -1 for the  $i^{th}$  cell then the *param\_elem(i)* used in the output is equal to the cell index (*i*); hence, if you want some output *param\_elem(i)* values equal to the cell indices but others equal to zero then use values of -1 and 0 respectively in the input *.ele* file.
- The output files are in the directory where this program is run from unless a path is specified in the *outputfile* parameter.
- For SCINTREX IPR-12 *.dmp* file conversion:
  - The *.txt* input file (second argument) contains electrode coordinate information with a single header line and each subsequent line providing the following information:  
 Index X Y Z LineID StationID  
 for example:  
 1 350711.00 5509493.00 0.00 1000E 20000W
  - In that electrode coordinates *.txt* file, only numerical digits and Cardinal directions N/S/E/W are allowed in the line and station IDs. Also, all stations for a single line should all appear in order.
  - The conversion is for any pole/dipole arrangements.
  - The outputs are four *.node* files and a *\*\_config.txt* file. Files *\*\_np.node*, *\*\_mx.node* and *\*\_rho.node* contain the data measurements: normalized potential (V/A), chargeability (V/V) and apparent resistivity respectively. The other *\*\_coords.node* file contains the information from the input electrode coordinate *.txt* file but with the LineID and StationID columns removed.
  - In the three output *.node* files containing the data measurements, the coordinates are set to points located between electrodes C2 and P1 at a depth equal to the distance between C2 and P1.
  - The remote current electrode is listed as the second in a pair in the corresponding lines in the output *\*\_config.txt* file.
- For column-based DC/IP *.txt* file conversion:
  - The first line of the *.txt* input file should contain two integer values specifying the *tp* and *ar* parameters (see more below) and should contain one of the keywords indicated above (see first item in the bullet list of Notes above).
  - The second and subsequent lines in the *.txt* input file should each contain the following information:  
 CX1, CY1, CZ1, CX2, CY2, CZ2, PX1, PY1, PZ1, PX2, PY2, PZ2, I, D, [rho]  
 where (CX1,CY1,CZ1) are the coordinates for the C1 electrode (first current electrode) and similarly for C2, P1 and P2 (second current electrode and the two potential electrodes); I is injected current (mA) and D is measured data (see more below); rho is optional and only used for a specific type of data (see more below).
  - The *tp* parameter specifies the type of data:
    - \* 0 for apparent resistivity (in Ohm-m). The data are not converted and are written to the output file as-is.
    - \* 1 for normalized potentials (in V/I). The data are not converted and are written to the output file as-is.
    - \* 2 for measured voltage (in mV) which are converted to normalized potential (V/I) by dividing it by current (I).
    - \* 3 for chargeability Vp/Vs (unitless) which are converted to an absolute value.
    - \* 4 for chargeability (in mV/V) which are converted to Vp/Vs by dividing its absolute value by 1000.
    - \* 5 for chargeability apparent (in ms) which are converted to Vp/Vs by dividing its absolute value by 700.
    - \* 6 for chargeability phase (in mrad) which are converted to Vp/Vs by dividing its absolute value by 700.



- \* 7 for chargeability PFE (in %) which are converted to Vp/Vs by dividing its absolute value by 100.
- \* 8 for chargeability FE which are converted to an absolute value.
- \* 9 for chargeability (time-domain) Metal Factor (in conductivity) which are converted to Vp/Vs by multiplying its absolute value by 0.35-times- $\rho$  where  $\rho$  is the apparent resistivity (in Ohm-m).

For all cases of chargeability data, any final value of Vp/Vs larger than 0.9 is changed to 0.9.

- The **ar** parameter specifies the type of survey array. This only affects the coordinates in the output **\*\_data.node** file (for visualization, discussed further below). These coordinates have no effect on the inversion.
  - \* 1 for pole/dipole arrays: the coordinates are located between electrodes C1 and P1 at a depth equal to the distance between C1 and P1.
  - \* 2 for Wenner arrays: the coordinates are located between electrodes P1 and P2 at a depth equal to the one third of the distance between C1 and C2.
  - \* For any other values (e.g. Schlumberger, Gradient): the coordinates are located between electrodes P1 and P2 at a depth equal to the half of the distance between C1 and C2.
- In the **.txt** input file, for any of pole arrays, the user should specify remote electrodes as the second listed electrodes (C2 or P2). If the coordinates of C2 or P2 aren't provided, a value of -999999 should be assigned to them by the user.
- The outputs are two **.node** files and a **\*\_config.txt** file. File **\*\_data.node** contains the data measurements (see discussion above on **tp** parameter for units and discussion above on **ar** parameter for coordinates assigned to each data measurement). File **\*\_coords.node** contains the information on the input electrode coordinates.

Format	inputfile1	inputfile2	outputfile
Datamine	.pt .ssv	.tr .ssv	Files <b>outputfile.node/.ele/.vtu</b> are written if <b>outputfile</b> has no extension.
Geomatic	.dat .txt etc.		(As above)
Geomview	.off		(As above)
Medit	.mesh		(As above)
Gocad solids	.ts .so .mx		(As above)
Wavefront obj	.obj		(As above)
AutoCAD dxf	.dxf		(As above)
GSS xyz mag	.xyz etc.		File <b>outputfile.node</b> is written if <b>outputfile</b> has no extension.
Geonics EM31 (Geosoft xyz)	.xyz		File <b>outputfile.node</b> is written if <b>outputfile</b> has no extension.
ESRI grid	.grd .asc		(As above)
Geosoft GXF grid	.gxf		(As above)
WSINV3DMT model	.model .initmod	[.sitenames]	Files <b>outputfile.mesh/.con/.vtr</b> are written if <b>outputfile</b> has no extension.
WSINV3DMT data	.data .responses	[.sitenames]	Files <b>outputfile.txt/.vtu</b> are written if <b>outputfile</b> has no extension.
Triangle/TetGen mesh	.node	.ele	File <b>outputfile.ts</b> or <b>outputfile.so</b> is written (for 3D or 2D respectively) if <b>outputfile</b> has no extension. If <b>outputfile</b> is named <b>mesh3d.dat</b> then it is written in that R3t file format.
CSV file, e.g. Oasis Montaj, QGIS	.csv		File <b>outputfile.node</b> is written if <b>outputfile</b> has no extension.
GEOMAR AUV MAG Survey	.txt		(As above)
SCINTREX IPR-12 .dmp file	.dmp	.txt	The <b>outputfile</b> should have no extension.
Column-based DC/IP .txt file	.txt		The <b>outputfile</b> should have no extension.

## 3.5 Combining multiple files of the same format

### 3.5.1 combine\_files

Combines the information in several sets of files. The supported file types are:

- `.node`
- `.ele`
- `.poly`
- unstructured meshes defined by pairs of `.node` and `.ele` files, and optionally additional `.neigh` files

Command line usages:

```
combine_files dir n ext fileroot1 fileroot2 ... filerootn outroot [dups]
```

```
combine_files dir n ext inputfile outroot [dups]
```

Command line parameters:

- `dir` specifies whether to combine the files “vertically” (`dir = V`) or “horizontally” (`dir = H`). For `.node` files, vertical combination assumes that you have several files specifying the same attributes at different coordinate locations: for example, two `.node` files holding vertical gravity data collected at different locations, one an airborne survey and one ground-based. For `.ele` files, vertical combination assumes that you have several files specifying the same attributes for different cells (different sets of node indices). Horizontal combination assumes that you have several files specifying different attributes at the same coordinate locations or node indices: for example, six `.node` files all specifying identical locations and each holding a different gravity gradiometry component.
- The program combines  $n \leq 16$  files (or sets of mesh files) named `fileroot1.ext`, `fileroot2.ext`, ... , `filerootn.ext` and writes the information to a file (or set of mesh files) named `outroot.exe`.
- `ext` specifies the type of files to combine. Possible options are `node`, `ele`, `poly` or `mesh`. The `mesh` option specifies `.node/.ele` pairs, or `.node/.ele/.neigh` triplets.
- If `ext` is `mesh` then files named `fileroot1.neigh` etc. are read, if they exist, and file `outroot.neigh` is written in addition to `outroot.node` and `outroot.ele`.
- If the second command line usage is used, the `inputfile` should list the `fileroot1` through `filerootn` parameters, one per line.
- If `dir = V` and `dups` is “t” (true, the default) then duplicate nodes or cells are removed. For nodes to be considered duplicates, all coordinates must be equal. For cells to be considered duplicates, all node indices must be equal (but can be in any order). The added “file.index” attributes are set to 0 for nodes or cells that were duplicates.

Notes:

- When `dir = V`, node and/or cell attribute columns are added named “file.index”. The values are on  $[1,n]$ , the value indicating which file the nodes and/or cells came from. However, if `ext` is `poly` then cell attribute information is not written to the output file `outroot.poly` because that file format only allows node attributes.
- When `dir = H`, there is no checking that the coordinate locations or node indices are identical, only that the number of nodes and/or cells is identical. The coordinates from the first file are used.
- The limitation of  $n \leq 16$  can be increased by the developers if you need.

# Chapter 4

## Data processing

### 4.1 Observation creation and data manipulation

#### 4.1.1 add\_noise

Adds noise or assigns uncertainty standard deviations to data in a `.node` or `.ele` file.

Command line usages:

```
add_noise datafile outroot mode col1 col2
add_noise datafile outroot mode col1 col2 seed
add_noise datafile outroot mode col1 col2 tol
add_noise datafile outroot mode col1 col2 flo perflo per move
add_noise datafile outroot mode col1 col2 flo perflo per move zrev
add_noise datafile outroot mode col1 col2 flo perflo per move zrev sym
add_noise datafile outroot mode col1 col2 flo perflo per move zrev sym i1 i2
add_noise datafile outroot mode col1 col2 flo perflo per seed
add_noise datafile outroot mode col1 col2 flo perflo per tol
add_noise datafile outroot mode inputfile
add_noise datafile outroot mode inputfile seed
add_noise datafile outroot mode inputfile tol
add_noise datafile outroot sigroot mode inputfile
add_noise datafile outroot sigroot mode inputfile seed
add_noise datafile outroot sigroot mode inputfile tol
```

For the last six command line usages above, the `inputfile` should be in this format:

```
nlines [zrev]
col1 col2 flo perflo per
col1 col2 flo perflo per
...
```

The `nlines` on the first line indicates the number of lines that follow. The `col1 col2` information on each line will normally specify different columns than on the other lines.

Command line or input file parameters:

- `datafile` specifies the data file to use. It can be a `.node` file or `.ele` file.
- If `sigroot` is absent in the command line usage then there is a single output file, with noise added or uncertainties assigned, named `outroot.ext` where the extension `exe` is taken from the `datafile`.
- If `sigroot` is present in the command line usage then there are two output files: `outroot.ext` contains data with noise added (only written if `mode` is "add"), and `sigroot.ext` contains the noise or uncertainties assigned.
- `sigroot` may not be set to "add" or "sig" because the usage is then unclear (see parameter `mode`).

- `col1` and `col2` are integers specifying attribute columns in the input or output files:
  - `col1` specifies the data column to use; this column must exist in `datafile` already
  - `col2` specifies the column to use for the uncertainties in `datafile` or `sigroot.ext`.
- If `mode` is set to "add" then noise is added to the data in column `col1` in `datafile`. If `mode` is set to "sig" then uncertainties are specified and placed in column `col2` (the data in `col1` are not altered). Any other specification of `mode` will cause an error.
- For the first three command line usages above (where `flo`, `perflo` and `per` are absent), the uncertainties are taken directly from attribute column `col2` (which must exist).
- For the other command line usages above, parameters `flo`, `perflo` and `per` specify how to calculate the uncertainties. Let array `d` hold the data values  $d_i$ . The standard deviation of the noise is then calculated as

$$\sigma_i = \text{flo} + (\text{perflo} \times \text{range}(\mathbf{d})/100) + (\text{per} \times |d_i|/100) \quad (4.1)$$

The first term is an absolute floor. The second term is a floor based on some percentage of the data range. The third term is relative to the data values. The noise  $\epsilon_i$  is created by multiplying those  $\sigma_i$  values by values taken from a random normal distribution with zero mean and unit standard deviation,  $N(0, 1)$ :

$$\epsilon_i = \sigma_i N(0, 1)_i = N(0, \sigma_i^2)_i \quad (4.2)$$

If `sigroot` is absent then the calculated uncertainties are placed in column `col2` of the output file. However, you can chose not to insert that column of uncertainties if you specify a non-positive value of `col2`.

- If `move` is set to `x`, `y` or `z`, and `datafile` is a `.node` file, then an additional `.vtu` file is written. The `x`-, `y`- or `z`-coordinates in that `.vtu` file are replaced with the data values. This is only for visualization purposes and does not apply to the `.node` file. For example, if you have data along a 2D horizontal profile then you may want to set `move` to `z` for visualization purposes: the data will then be plotted at heights corresponding to the data values instead of their measurement elevations. Similarly, if you have data down a vertical borehole then you may want to set `move` to `x` or `y` for visualization purposes. If `datafile` is a `.ele` file then `move` is ignored. Setting `move` to `y` is only valid for 3D data sets.
- Set `zrev` to "t" (true) if you want to switch coordinate systems such that the direction of `z` is reversed in the `.vtu` output file. If `datafile` is a `.ele` file then `zrev` is ignored.
- Set `sym` to "t" (true) if you want the noise added to be symmetric (this only works for a 2D data profile).
- If the integer parameters `i1` and `i2` are supplied then the program only operates on data indices (file rows) `i1` through `i2`.
- If an integer `seed` is supplied then it is used to seed the random number generator. Otherwise a clock-dependent seed is used, which will be different every time you run the program. The seed used is printed to the screen so you can use the same seed later if you want to reproduce the results.
- If a real-value tolerance `tol` is supplied then the program pulls new seeds for the random number generator until one is found such that

$$\frac{1}{n} \sum_{i=1}^n \left( \frac{\epsilon_i}{\sigma_i} \right)^2 = \frac{1}{n} \sum_{i=1}^n N(0, 1)_i^2 \leq \text{tol} \quad (4.3)$$

where  $n$  is the number of data. This is helpful when running synthetic inversions where you want the theoretical expected value for the  $\chi^2$  misfit (i.e. the target misfit equal to the number of data) to be consistent with the actual noise added.

- If specifying `tol`, you **must** include a decimal place to distinguish it from an integer `seed`.

Notes:

- For data where there is only a single component, such as gravity, magnetics and traveltimes, set `col1` to 1 and `col2` to 2.
- For data where there are several components, such as gravity tensor data, use the second command line usage above and specify `col1` and `col2` as required in the `inputfile` to specify different noise parameters for the different components.
- Space-filling zeros are added into the output file if required, e.g. if `col2` is larger than the number of existing data columns, or if `i1` and `i2` are used to specify a subset of data to operate on.

#### 4.1.2 break\_lines

Breaks the data observation locations in a `.node` file into lines based on data separation distances and line deviation angles.

Command line usages:

```
break_lines fileroot outroot atol dtol
```

```
break_lines fileroot outroot atol xtol ytol
```

Command line parameters:

- Reads data from file `fileroot.node`, adds line break information in a new attribute named “lineIndices”, and writes the new information to file `outroot.node`.
- Spatial vectors are calculated between each pair of observations and deviation angles are calculated between each pair of vectors. `atol` specifies a threshold on the deviation angles.
- `dtol` specifies a threshold on the distance between observations.
- `xtol` and `ytol` specify a threshold on the x- and y-distances between observations.
- For the first command line usage above, a line break is made whenever a lateral distance is larger than `dtol` OR a deviation is larger than `atol` (in degrees).
- For the second command line usage above, a line break is made whenever an x-distance is larger than `xtol` OR a y-distance is larger than `ytol` OR a deviation is larger than `atol` (in degrees).

Outputs:

- File `outroot.node` contains the input nodes plus new line information.
- File `outroot.ele` holds linear elements between nodes in the same lines. A single cell attribute in that file named “lineIndices” specifies the line numbers. This file is helpful for visualization purposes.
- Files `outroot_ends.node` and `outroot_ends.ele` contain similar information to the other output files but only the nodes at the start and ends of the lines are included. These files are helpful for visualization purposes when the other two are very large.

### 4.1.3 decimate

Decimates (down-samples) data rows from a `.node` or `.ele` file. This is strictly decimation based on data ordering or line numbering. If you want to decimate data based on distance of those data points from each other then use program [remove.duplicates](#). If you want to decimate data based on distance away from another set of points then use program [decimate.by.nodes](#). If you want to decimate data based on a mesh then use program [decimate.by.mesh](#). If you want to decimate entire lines of data, e.g. flight-lines for an airborne survey, then you’ll have to use program [break.lines](#) first.

Command line usage:

```
decimate filename outroot lineflag from1 to1 by1 [from2 to2 by2 n2 tfile]
```

Command line parameters:

- Reads data from `filename`. This will usually be a `.node` file but can be a `.ele` file for traveltime data.
- Writes the decimated data to `outroot.ext` where the extension `ext` is taken from the `filename` parameter.
- If `lineflag` is `t` (true) then entire lines of data are decimated, otherwise decimation works along the data ordering. This is only possible for `.node` files with line ID’s stored in the boundary marker column. All data in a single line must be grouped together.
- If only the `from1`, `to1` and `by1` parameters are provided then there is a single decimation loop like this:

```
DO i1=from1,to1,by1
  ! Add i1-th data observation OR line of data to the output:
  ...
END DO
```

- If `lineflag` is `f` (false) and the optional parameters `from2`, `to2`, `by2` and `n2` are provided then there is a nested decimation loop like this:

```
i = 0
DO i1=from1,to1,by1
  DO i2=from2,to2,by2
    i = i + 1
    ! Add i-th data observation to the output:
    ...
  END DO
END DO
```

**n2** should specify the number of secondary items per primary item. For example, if decimating a file of sources and receivers with receivers changing fastest (secondary item) along the rows then you would set **n2** to the number of receivers. For that scenario, set **tfile** to **t** (true) to renumber the source/receiver index values stored in the two coordinate columns.

- If **lineflag** is **t** (true) and the **from2**, **to2**, **by2** and **n2** parameters are provided then:
  - the **from1**, **to1** and **by1** parameters specify the decimation for entire lines
  - the **from2** and **by2** parameters specify the decimation along each line
  - the **to2**, **n2** and **tfile** parameters are ignored but dummy values are required in the command line call.

#### 4.1.4 **decimate\_by\_mesh**

Decimates data based on a UBC-GIF mesh file such that there is only one observation point above each vertical column of cells. This program currently only supports 3D problems.

Command line usage:

```
decimate_by_mesh noderoot outroot meshfile [icol]
```

Command line parameters:

- Reads data from **noderoot.node**.
- Writes decimated data to **outroot.node**.
- File **meshfile** is the UBC-GIF mesh file. See [Section 2.2.1](#) for a description of the required file format.
- If **icol** is non-zero then its absolute value specifies an attribute index in file **noderoot.node**.
- For a particular vertical column of mesh cells, all data above that column are gathered. Then, if **icol** equals zero or is absent, the datum closest to the lateral centre of the vertical column of cells is kept. If **icol** is positive then the datum with the highest value for that attribute will be kept. If **icol** is negative then the datum with the lowest value for that attribute will be kept.

#### 4.1.5 **decimate\_by\_nodes**

Decimates data so there are no observation points within some distance from a second set of points. This program currently only supports 3D problems.

Command line usage:

```
decimate_by_nodes noderoot1 noderoot2 outroot distance
```

Command line parameters:

- Reads data from file **noderoot1.node**.
- The second set of points is read from file **noderoot2.node**
- Only the data points that lie within the specified **distance** from the second set of points are kept.
- Writes decimated data to **outroot.node**.

#### 4.1.6 **drill\_spline**

Fits a Catmull-Rom spline through down-hole observation locations and writes interpolated locations to a new file.

Command line usage:

```
drill_spline inroot ds outroot
```

Command line parameters:

- The input file **inroot.node** should contain observation locations along the drill-hole.
- The output file **outroot.node** contains interpolated locations along the drill-hole.
- The interpolated locations are spaced at a distance of **ds** apart.

Assumptions:

- The first point in the input file is assumed to be the surface collar location and the last point the deepest extent of the drill-hole.

### 4.1.7 ll2utm

Converts the coordinates from a 3D `.node` file from longitude/latitude to UTM (WGS84, units in meters). Elevation is unaffected. The program was written following Karney's 2011 method: <https://arxiv.org/pdf/1002.1417.pdf>.

Command line usage:

```
ll2utm nodesroot outroot
```

Command line parameters:

- Reads information from file `nodesroot.node`. The coordinate columns in file `nodesroot.node` should be ordered as follows:  
Longitude    Latitude    Elevation.
- The output file is named `outroot.node`. The coordinate columns in file `outroot.node` are ordered as follows:  
Easting    Northing    Elevation.

### 4.1.8 make\_obs

Creates gridded observation locations and writes them to a `.node` or `.ele` file.

Command line usages:

```
make_obs ndim x1 x2 dx [y1 y2 dy] z1 z2 dz outroot [eleflag]
make_obs inputfile [eleflag]
```

If `inputfile` is provided then it should be of the following format:

```
n ndim
x1_1 x2_1 dx_1 [y1_1 y2_1 dy_1] z1_1 z2_1 dz_1
x1_2 x2_2 dx_2 [y1_2 y2_2 dy_2] z1_2 z2_2 dz_2
...
x1_n x2_n dx_n [y1_n y2_n dy_n] z1_n z2_n dz_n
```

where `n` is the number of observation specification lines in the file.

Command line or input file parameters:

- `ndim` is the number of dimensions.
- Observations run in the x-direction from `x1` to `x2` (but not beyond `x2`), spaced `dx` apart.
- Observations run in the y-direction from `y1` to `y2` (but not beyond `y2`), spaced `dy` apart.
- Observations run in the z-direction from `z1` to `z2` (but not beyond `z2`), spaced `dz` apart.
- Do not include `y1`, `y2` and `dy` for 2D problems.
- If `dx` is set to zero then all observations will be located at `x1` (similarly for `dy` and `dz`).
- If `eleflag` is "f" (false, the default) then a `.node` file is written. Otherwise a `.ele` file is written.
- If the second command line usage is used and `eleflag` is "f" (false, the default), line numbers are added to the output `.node` file (in the boundary marker column). These count up from 1 for each line specified in the `inputfile`.
- If the second command line usage is used, any duplicates are removed. Observation locations are considered duplicates if all coordinate values are identical. Two points with identical *x* and *y* locations but different *z* locations are not considered duplicates.
- Writes file `outroot.*` (first command line option) or `inputroot.*` (second command line option) where the `inputfile` parameter is assumed to be of the form `inputroot.*`. The extension used for the output file is either `.node` or `.ele` (depending on the value of the `eleflag` parameter).

Examples:

- To generate observation locations for a 2D profile from  $x = -10\text{m}$  to  $x = 10\text{m}$  every 2m, at a height of 1m, use  
`make_obs 2 -10 10 2 1 1 0 outroot`

- To generate a 3D grid of observation locations from  $x = -10\text{m}$  to  $x = 10\text{m}$  every 2m, and from  $y = -20\text{m}$  to  $x = 20\text{m}$  every 4m, at a height of 1m, use  
`make_obs 3 -10 10 2 -20 20 4 1 1 0 outroot`
- To generate several sets of observations along vertical boreholes use an input file with **n** equal to the number of boreholes and lines of the form  
`x1_i x2_i 0 y1_i y2_i 0 z1_i z2_i dz_i`  
 where  $(x1\_i, y1\_i)$  defines the lateral location of the  $i^{th}$  borehole and the z-direction parameters ( $z1\_i$ ,  $z2\_i$  and  $dz\_i$ ) specify the depth extents and data spacing along the  $i^{th}$  borehole.

#### 4.1.9 mag\_dipole

Generates the TMI response of a magnetic dipole.

Command line usages:

```
mag_dipole obsroot outroot c xq yq zq kx ky kz lx ly lz
mag_dipole obsroot outroot s xq yq zq ki kd ks li ld ls
```

Command line parameters:

- Observation locations are taken from `.node` format file `obsroot.node`.
- TMI measurements are written to `.node` format file `outroot.node`.
- Literal characters 'c' and 's' indicate Cartesian or spherical inputs respectively.
- The dipole is located at Cartesian location  $(xq, yq, zq)$ .
- The dipole moment can be specified in a Cartesian system as the vector

`kx, ky, kz`

or in a spherical system with **ki** the inclination angle (positive down from vertical), **kd** the declination angle (positive east of north) and **ks** the moment magnitude.

- The measurement direction for the observed TMI data can be specified in a Cartesian system as the vector are calculated in measurement direction

`lx, ly, lz`

or in a spherical system with **li**, **ld** and **ls** the inclination angle, declination angle and moment magnitude respectively.

- All input coordinates are assumed to be in a  $+x$  east,  $+y$  north,  $+z$  up system.

#### 4.1.10 remove\_duplicates

Removes duplicate or closely spaced observation points from a `.node` file. If no nodes need to be removed then nothing happens (no output file is written).

Command line usages:

```
remove_duplicates noderoot eleroot outroot
remove_duplicates noderoot eleroot outroot distance
remove_duplicates noderoot eleroot outroot distance keep
remove_duplicates noderoot eleroot outroot distance keep usez
remove_duplicates noderoot eleroot outroot distance keep usez icol
```

Command line parameters:

- Reads data from `noderoot.node`.
- Reads optional cell definitions from `eleroot.ele`. Enter `null` for the `eleroot` parameter if you just want to work with nodes.
- Writes data with duplicates (or closely spaced observation points) removed to `outroot.node`. If there are no duplicates then no output file is written.
- If `eleroot` is not `null` and the cells are altered by removing nodes then altered cell definitions are written to `outroot.ele`.



- If a positive **distance** is provided then it specifies a maximum node separation distance. For any two nodes separated by less than that distance, the second of those nodes is removed. There is a nested loop in the code that makes this a much slower algorithm than removing duplicates based on equal (x,y,z) coordinate values.
- Set **keep** to "t" (true) to keep the duplicates and remove everything else; the default is "f" (false).
- Set **usez** to "f" (false) to ignore elevations; the default is "t" (true).
- If **icol** is non-zero then it specifies an attribute index in file **noderoot.node**. Duplicates with a higher value for that attribute will be preferred. If **icol** is negative then the absolute value **|icol|** defines the attribute index and duplicates with lower value for that attribute will be preferred.

#### 4.1.11 remove\_nodes

Removes any nodes in a nodes file, or in an unstructured mesh, with a specified elevation or attribute value. See also program [remove\\_cells](#).

Command line usages:

```
remove_nodes noderoot eleroot outroot ai v1
remove_nodes noderoot eleroot outroot ai v1 v2
remove_nodes noderoot eleroot outroot ai v1 v2 keep
```

Command line parameters:

- File **noderoot.node** contains the nodes to work with. If **eleroot** is specified as something other than "null" then the program assumes that files **noderoot.node** and **eleroot.ele** define a model on an unstructured mesh; the cells in **eleroot.ele** may need to be changed once nodes are removed.
- If **ai** is zero then the elevations are used. If **ai** is a positive integer then it specifies a node attribute column to use.
- If **v2** is absent, any nodes with elevation/attribute equal to value **v1** are marked.
- If **v2** is present, any nodes with elevation/attribute on the range **[v1,v2]** (equal to or between those values) are marked.
- If you set **v2** equal to **v1** then the program will provide the same results as if **v2** were absent.
- If **keep** is "f" (false, the default) then those marked nodes are removed, otherwise they are kept (everything else is removed).
- The remaining nodes, and cells if available, are written to files named **outroot.node/.ele**. If the nodes or cells do not change then the respective file is not written.

#### 4.1.12 remove\_range

Removes nodes from a particular range in a **.node** file or masks those nodes by a polygon.

Command line usages:

```
remove_range datafile outroot keep x1 x2 [y1 y2] z1 z2
remove_range noderoot eleroot outroot keep x1 x2 [y1 y2] z1 z2
remove_range noderoot eleroot outroot keep xc [yc] r
remove_range noderoot eleroot outroot keep coordsfile [iat]
remove_range noderoot eleroot outroot keep maskfile [iat]
```

Command line parameters:

- For the first command line usage above, the **datafile** should be a column-based data file with columns x-y-z. Any additional columns will not be printed to the output file, only the coordinate information.
- For the other command line usages above, node and cell information is read from files **noderoot.node** and **eleroot.ele** respectively. If file **eleroot.neigh** exists then it is also read. Set **eleroot** to **null** to process a **.node** file only.
- For the first command line usage above, the output file is named **outroot.node**.
- For the other command line usages above, the output files are named **outroot.node/.ele/.neigh** (both the node and cell definitions will change). The output **.ele** file is only written if **eleroot** is not set to **null** and the cell definitions change. The output **.neigh** is only written if file **eleroot.neigh** exists and the cell definitions change.

- For the first two command line usages above, the **x1 x2 [y1 y2] z1 z2** parameters specify the x-, y- and z-direction limits for the range to use. Do not supply **y1** and **y2** for 2D problems. A coordinate  $(x, y, z)$  is defined as being inside the specified range if  $x1 \leq x \leq x2$  and  $y1 \leq y \leq y2$  and  $z1 \leq z \leq z2$
- For the third command line usage above, the **xc [yc] r** parameters specify the centre and radius of a circle for the range to use. Do not supply **yc** for 2D problems. A coordinate  $(x, y, z)$  is defined as being inside the specified circle if it is a distance of **r** or less from the circle centre (inside the circle or on the perimeter).
- The fourth command line usage above reads the **x1 x2 [y1 y2] z1 z2** parameters from the first line of a file named **coordsfile**. An extension must be present in the name of the **coordsfile** and it must not be **.node**.
- For the fifth command line usage above, file **maskfile** should be at **.node** file defining the 2D polygon, e.g. easting and northing coordinates of the vertices, one pair of coordinates on each line. The polygon does not need to be closed explicitly in this file. The extension of this file must be **.node**.
- If **keep** is **"t"** (true) then nodes/data inside of the specified range, circle or mask are kept (nodes outside are removed).
- If **keep** is **"f"** (false) then nodes/data inside of the specified range, circle or mask are removed (nodes outside are kept).
- If there aren't any nodes/data inside or outside of the specified range, circle or mask (depending on how **keep** is set) then nothing happens.
- If **iat** is present it should be a positive integer value that specifies a node attribute column. In this case, the nodes are marked instead of removed. Any nodes that would remain (were they actually removed) are marked with a value of 1 and all other nodes are marked with 0. In this case, no **.ele/.neigh** files are written because that information does not change.
- If **iat** is greater than the number of existing attribute columns in the node file then the required attribute column is created and its values set to appropriate defaults (1 or 0 depending on the value of **keep**).

Notes:

- You have to run this program multiple times if you want to decimate a set of nodes based on multiple ranges. If attempting to mark nodes (instead of remove) then you will want to make sure that on the first run, **iat** is greater than the number of existing attribute columns in the node file.

### 4.1.13 remove\_trend

Removes a polynomial trend from  $(x, y, d)$  data in a **.node** file.

Command line usages:

```
remove_trend inroot outroot iat n
remove_trend inroot outroot iat n cval
remove_trend inroot outroot iat n xmin ymin a0 a01 ...
```

Command line parameters:

- File **inroot.node** holds the  $(x, y, d)$  data to fit a linear trend to:  $x$  and  $y$  are taken from the first two node coordinates and  $d$  from attribute number **iat**.
- A trend of the following form is fit to the data:

$$\begin{aligned}
 t &= a_0 + a_{01}\tilde{x} + a_{10}\tilde{y} + a_{11}\tilde{x}\tilde{y} + a_{20}\tilde{x}^2 + \dots + a_{ij}\tilde{x}^i\tilde{y}^j + \dots + a_{nn}\tilde{x}^n\tilde{y}^n \\
 \tilde{x} &= x - x_{min} \\
 \tilde{y} &= y - y_{min}
 \end{aligned}
 \tag{4.4}$$

where **n** specifies the highest order terms (the degree of the polynomial trend), and  $x_{min}$  and  $y_{min}$  are the smallest  $x$  and  $y$  values respectively.

- If the trend parameters **a0** etc. are supplied (the third command line usage above) then they are used. Otherwise, they are calculated.
- The calculated trend is removed from  $d$  and the result written to file **outroot.node**.
- If **n** is 0 and **cval** is absent (the first command line usage above) then the average data value is removed.

- If `n` is 0 and `cval` is present (the second command line usage above) then rather than calculating and removing a trend, the program adds (**adds!!!**, not subtracts) value `cval` to the data.
- If `n` is 1 or greater then:
  - `cval` should be absent.
  - The trend removed is added to file `outroot.node` as a new attribute.

Notes:

- Any  $z$  elevations are ignored completely.
- The conjugate gradient (CG) algorithm is used to calculate the matrix inverse.
- The maximum iterations for the CG algorithm are set to `n` and tolerance to zero.
- See also programs [lin\\_reg](#) and [linear\\_regression](#).

#### 4.1.14 `reorder_attributes`

Reorders the attributes in a `.node` or `.ele` file.

Command line usage:

```
reorder_attributes inputfile outputroot n i1 i2 ... in
```

Command line parameters:

- File `inputfile` is a `.node` or `.ele` file containing attribute columns. Any other extension is treated as a simple column-based data file with the same number of values on every line.
- The output file will be named with root `outputroot` with the extension taken from the `inputfile`. If no extension is found in `inputfile` then `.node` is assumed.
- The output file will have `n` attributes.
- The attribute columns are reordered as indicated by the indices `i1 i2 ... in`. For example, if one row of attributes in the input file contains values ordered

1.2 3.4 5.6 7.8 9.0

and the indices specified are

3 1 4 4 2 5

then the resulting row of attributes in the output file contains values ordered

5.6 1.2 7.8 7.8 3.4 9.0

Notes:

- You can specify as few or as many indices as you like, although currently `n` is limited to at most 32 (this can be changed if desired - please inform the developers).
- If any indices are below 1 or above the number of attributes in the input file then an error is thrown and the program will not execute.
- You do not have to move all existing attributes from the input file to the output file.
- If any indices are duplicated, the corresponding attribute columns will be duplicated in the output file, as in the example above.

#### 4.1.15 `smooth_node`

Smooths the coordinates or attributes in a `.node` file using a simple neighbourhood-averaging method.

Command line usages:

```
smooth_node inroot outroot iat d
```

```
smooth_node inroot outroot iat d mapflag
```

```
smooth_node inroot outroot iat d mapflag iat2 flag2
```

Command line parameters:

- Node information is read from file `inroot.node`.
- Smoothed node information is written to file `outroot.node`.
- If `iat` is a positive integer then it specifies the index of the node attribute to smooth. If non-positive (negative or zero) then coordinates are smoothed.
- `d` is the neighbourhood distance.
- If `mapflag` is "t" (true, the default) then the neighbourhood distance is a map distance, i.e. elevations are ignored when calculating distances.
- If `iat` is non-positive and `mapflag` is "t" (true, the default) then only elevations are smoothed (e.g. appropriate for smoothing a topography surface).
- If `iat` is non-positive and `mapflag` is "f" (false) then the node coordinates are smoothed across all the dimensions (e.g. appropriate for smoothing a surface model of an isolated shape).
- If `iat2` is provided and non-positive then it specifies an attribute column that masks the smoothing. Any nodes with attribute `iat2` equal to zero are not smoothed.
- If `flag2` is provided and "t" (true, the default) then the positions of nodes masked for smoothing (by parameter `iat2`) are affected by the positions of those not smoothed. If "f" (false) then the positions of the smoothed nodes are only affected by those masked for smoothing.

#### 4.1.16 transform\_coordinates

Coordinate transformation of data or a model.

Command line usages:

```
transform_coordinates inputfile transformfile outputroot
```

Command line parameters:

- Reads data or model information from file `inputfile`, which can be a `.node` or `.poly` file. If the extension is missing from `inputfile` then a `.node` file is assumed.
- Reads coordinate transformation instructions from `transformfile`. The file format is described below.
- Writes the transformed data or model to `outputroot.node` or `outputroot.poly` depending on the `inputfile` format.

Transform file format:

- The first line specifies the number of coordinate transforms and the number of dimensions.
- The lines that follow specify the type and parameters for each transform, which are performed in the order found in the transform file.
- The possible transform types are "translate", "scale", "rotate", "zreverse", "cart2sphr" and "sphr2cart".
- A translation needs parameters `dx` `[dy]` `dz` to specify the additive translation along each Cartesian axis (`dy` is not used in 2D).
- A scaling needs parameters `dx` `[dy]` `dz` to specify the multiplicative scaling along each Cartesian axis (`dy` is not used in 2D).
- A rotation needs parameters `[strike]` `dip` `[tilt]` `[rev]` to specify the axis rotation (`strike` and `tilt` are not used in 2D and `rev` is optional). Refer to [Li & Oldenburg \(2000a\)](#) and [Lelièvre & Oldenburg \(2009\)](#) for the conventions used. A null rotation has `strike`, `dip` and `tilt` set to 0, 90 and 0 respectively. The `rev` parameter reverses the order of the strike-dip-tilt operations (body versus axis rotation).
- The "zreverse" transform moves from a z-down to z-up, or z-up to z-down, coordinate system:
  - in 2D and 3D, `z` is multiplied by -1
  - in 3D, the `x` and `y` values are swapped.
- The "cart2sphr" and "sphr2cart" transforms move from Cartesian space to spherical/polar space or the reverse. The order of Cartesian coordinates is `x-y-z` with `+x` North, `+y` East, and `+z` down. The order of spherical coordinates is `r-p-t` with `r` the radius (distance from origin), `p` the declination positive from North towards East (`+x` towards `+y`), and `t` the inclination positive from horizontal downwards (towards `+z`). For 2D, `y` and `p` are not used. All angles are assumed to be in radians.

Notes:

- Because of the assumptions made when using the “cart2sphr” and “sphr2cart” transforms, you may also have to make use of the “zreverse” transform to convert between different Cartesian coordinate systems, the “scale” transform to convert from degrees to radians or the reverse, and the “translate” transform to add or subtract origin coordinates.

An example 3D transform file is as follows:

```
4 3
zreverse
translate -40.0 -5.0 0.0
scale 1.0 1.0 -1.0
rotate 15.0 60.0 0.0
```

An example 2D transform file is as follows:

```
4 2
zreverse
translate -40.0 0.0
scale 1.0 -1.0
rotate 60.0
```

## 4.2 Data visualization

See programs [node2vtu](#) and [ele2vtu](#).

### 4.2.1 drillcore2node

Converts drillcore information from several `.csv` files to a `.node` file.

Command line usages:

```
drillcore2node collarcsvfile surveycsvfile propcsvfile
drillcore2node collarcsvfile surveycsvfile propcsvfile noderoot
```

Command line parameters:

- `collarcsvfile` is a `.csv` file storing the drillcore collar information. It should contain columns HOLEID,DEPTH,X,Y,Z where X,Y,Z define the Easting, Northing and Elevation of the drillhole collars. The HOLEID values should increase.
- `surveycsvfile` is a `.csv` file storing the drillcore survey information. It should contain columns HOLEID,DEPTH,AZIMUTH,DIP where DEPTH is a distance along the hole (not a true depth). The rows should be sorted into groups by HOLEID. The HOLEID values should increase and the DEPTH values should increase for each hole.
- `propcsvfile` is a `.csv` file storing the drillcore physical property information. It should contain columns HOLEID,FROM,TO,PRO where FROM and TO are distances along the hole. The rows should be sorted into groups by HOLEID. The HOLEID values should increase and the FROM and TO values should increase for each hole.
- All `.csv` files should contain a single header line.
- If parameter `noderoot` is present in the command line then the output file is named `noderoot.node`. Otherwise is is named `propcsvroot.node` where `propcsvroot` is the root of the `propcsvfile` (file name with `.csv` file extension removed).
- The output `.node` file contains nodes along each drillhole, placed at the centres of the FROM-TO intervals.

### 4.2.2 drillcores2mesh

Converts drillcore information (at the moment only one attribute) from a `.csv` formatted drillcore intervals file to a `.node` and `.ele` file pair. This program assumes that each interval is oriented completely verically (no azymuth, deg 90 dip), and only accounts for the recovered drillcore segments, not the collar locations.

Command line usage:

```
drillcores2mesh inputfile outputfile
```

Command line parameters:

- **inputfile** is the root of the **.csv** file storing the drillcore data, and **outputfile** will be the root of the outputted **.node** and **.ele** files.

The drillcore **.csv** file is set up as (with headers):

Easting,	Northing,	Elevation [m],	From [m],	To [m],	Attribute
399230.7,	399230.7,	-1544.9,	1.28,	1.77,	1
399230.7,	399230.7,	-1550,	5.2,	60.5,	1
399230.7,	399230.7,	-1555.3,	61,	150.32,	2
399235,	399231,	-1530.2,	0,	22.6,	6
...					

## 4.3 Data query and calculations

### 4.3.1 calc\_cor

Calculates the correlation between two sets of numbers, e.g. data or model values. This can be used to calculate the correlation between a terrain-corrected gravity response and the topography elevations.

Command line usages:

```
calc_cor datfile
calc_cor datfile i1 i2
```

Command line parameters:

- The **datfile** should be a column-based ASCII data file from which the values to use for the correlation calculation will be taken.
- Indices **i1** and **i2** specify the columns in the **datfile** to use in the correlation calculation. If **i1** and **i2** are missing from the command line usage then the first and second columns are used.

Outputs:

- The calculated correlation value is printed to the screen. This value is on  $[-1,1]$  and represents the degree of linear relationship between the two sets of numbers. A value of 1 indicates an exact linear relationship with a positive slope. A value of -1 indicates an exact linear relationship with a negative slope.

Notes:

- Program **calc\_cor** is a version of [calculate\\_correlation](#) that has minimal dependence on other modules.
- There are hardwired values used in this program for the maximum number of rows (1,000,000) and columns (100) allowed in the input data file. Increasing those hardwires is simple; please inform the developers, or you can do it yourself if you have the code.

### 4.3.2 calculate\_correlation

Calculates the correlation between two sets of numbers, e.g. data or model values. This can be used to calculate the correlation between a terrain-corrected gravity response and the topography elevations.

Command line usages:

```
calculate_correlation datroot i1 i2
calculate_correlation datroot i1 i2 r outroot
```

Command line parameters:

- The data values are taken from **.node** format file **datroot.node**.
- Indices **i1** and **i2** specify the coordinate or attribute columns in file **datroot.node** to use in the correlation calculation. If these indices are positive then they specify attribute columns. If they are negative then their absolute values specify coordinate columns.

- If **r** is present then it specifies a window radius. For each data measurement, the correlation is calculated for that data measurement and any others within distance **r**. This provides a correlation value for every data measurement and those values are written as a node attribute to file **outroot.node**. If for any data measurement there are no other data measurements within distance **r**, a value of 2 is used for the correlation for that data measurement.

Outputs:

- The calculated correlation value for the entire data set is printed to the screen. This value is on [-1,1] and represents the degree of linear relationship between the two sets of numbers. A value of 1 indicates an exact linear relationship with a positive slope. A value of -1 indicates an exact linear relationship with a negative slope.
- If **r** and **outroot** are present then windowed correlation values for every data measurement are written as a node attribute to file **outroot.node**. The coordinate or attribute columns used to calculate the correlation, and the number of data in each window, are also included as attributes in that output file.

Examples:

- To calculate the correlation between a terrain-corrected gravity response, in the first attribute of a 3D **.node** file, and the topography elevations, you would use this command:  
`calculate_correlation datroot -3 1`

Notes:

- Program **calc\_cor** is a version of **calculate\_correlation** that has minimal dependence on other modules.

### 4.3.3 cluster\_analysis

Performs cluster analysis.

Command line usages:

```
cluster_analysis inputfile outputroot method normalize dovol maxj nclusters icol1 icol2
cluster_analysis inputfile outputroot method normalize dovol maxj nclusters icol1 icol2 icol3
cluster_analysis inputfile outputroot method normalize dovol maxj nclusters icol1 icol2 icol3 seed
```

Command line parameters:

- The **inputfile** can be a **.node** format file, a **.ele** format file, or a simple column-based data file.
  - If **inputfile** specifies a **.node** file then the data values are taken from the coordinate columns in that file.
  - If **inputfile** specifies a **.ele** file then the data values are taken from the attribute columns in that file.
  - If **inputfile** has any other extension then a simple column-based data file is assumed and the data values are taken from the data columns in that file.
- A file is written with root name **outputroot** and extension taken from the **inputfile**.
- **method** specifies the clustering method to use:
  - "k" for hard k-means clustering
  - "c" for fuzzy c-means clustering
  - "g" for Gustafson-Kessel clustering (which is a fuzzy approach using Mahalanobis distance instead of Euclidean distance, meaning the clusters can be rotated ellipses/ellipsoids instead of circles/spheres).
  - "l" for a linear regression relationship.
- **normalize** specifies the normalization to use:
  - 0 for no normalization
  - 1 to subtract the minimum value, then divide by the range, so that the normalized values are scaled linearly to lie on [0,1]
  - 2 to subtract the mean value, then divide by the standard deviation (this statistical "normalization" or "standardization" works well if the data are normally distributed).
- Set **dovol** to "t" (true) to solve for different cluster areas/volumes (default is "f" (false)). This is only relevant to methods "c" and "g".

- `maxj` specifies the number of randomly initialized minimizations to run. Each of the `maxj` solutions are local minimizers. The best solution found is returned. The number of minimizations defaults to 100 if a non-positive value is specified for `maxj`.
- `nclusters` specifies the number of cluster centres to fit to the data.
- `icol1`, `icol2` and `icol3` (if present and greater than zero) specify the indices of the data columns to use in the calculation (e.g. coordinates or attributes if the `inputfile` is a `.node` or `.ele` file respectively).
- If using a linear regression relationship (`method` equal to "1") then `icol1` should specify the independent variable, `icol2` should specify the dependent variable, and `icol3` should not be used (or set to a non-positive value).
- Reproducible results can be obtained by specifying the random `seed`.

#### Outputs:

- The random seed used is printed to the screen.
- The solution (the `ncluster` cluster centres and volumes if used, or the linear relationship information), the specified clustering measure, and the XBI validity measure (if relevant for the method specified) are printed to the screen.
- A file is written with root name `outputroot` and extension taken from the `inputfile`. The output file contains the following information, in columns in the following order:
  - the data used for the calculation, taken from the `inputfile`
  - for clustering methods other than `k`: the membership values for each cluster for each datum (if the output file is a `.node` or `.ele` file then these attributes are named "MembershipCluster1", "MembershipCluster2", etc.).
  - when the input file is a `.node` or `.ele` file only: indices specifying the closest cluster centre for each datum (this attribute is named "ClosestCluster" for method "k" or "BestMembership" otherwise)

#### Notes:

- The cluster analysis is automatically performed many times (see parameter `maxj`) with different random starting solutions. The best solution from all the runs is returned. **This does not guarantee that the true global minimum solution is found.** You should increase parameter `maxj`, or run the program many times, until you are confident that the solution returned is in fact the true global minimum solution. Alternatively, some sensible starting solution could be provided by the user but that would require further development of this program (contact the developers if you would like this option developed).
- See [Paasche & Eberle \(2009\)](#) and [Sun & Li \(2017\)](#) for further information.

### 4.3.4 data\_addition

Adds attributes from two datasets together.

Command line usages:

```
data_addition datafile1 datafile2 outfile
data_addition datafile1 datafile2 outfile ai1 ai2
data_addition datafile1 datafile2 outfile ai1 ai2 keepatts
```

Command line parameters:

- Reads data from `datafile1` and `datafile2` (both `.node` or `.ele` files).
- If `ai1` and `ai2` are present then the only addition calculated is that for attribute (data column) `ai1` in the first data file and attribute `ai2` in the second.
- If `ai1` and `ai2` are absent then additions are calculated for all shared attribute columns (see notes below).
- If `ai1` and `ai2` are set to zero then the program calculates the addition of the data elevations.
- If `ai1` and `ai2` are set to zero and `keepatts` is "t" (true) then the attributes from `datafile2` are copied over to the output file `outfile`. The default for `keepatts` (if absent) is "f" (false).

#### Outputs:

- A single file is written, named `outfile`.

#### Notes:



- If the two data files do not have the same number of attributes then the first  $n$  attributes are used where  $n$  is the number of attributes in the file with the least number of attributes.
- If attribute names differ in the two data files, the attribute names used in the output file are those taken from the 2nd datafile.

### 4.3.5 data\_difference

Provides statistics on the difference between two data sets. To do similarly for models, you can use this program with `.ele` format model files or use program [model\\_difference](#) with UBC-GIF format model files. Program [model\\_difference](#) can be used to calculate the difference between two column-based data files with the same number of values on every line. For magnetic vector inversion results, you may need program [mvi\\_difference](#).

Command line usages:

```
data_difference datafile1 datafile2
data_difference datafile1 datafile2 outfile
data_difference datafile1 datafile2 outfile i1
data_difference datafile1 datafile2 outfile i1 i2
data_difference datafile1 datafile2 outfile i1 i2 ai1
data_difference datafile1 datafile2 outfile i1 i2 ai1 ai2
data_difference datafile1 datafile2 outfile i1 i2 ai1 ai2 ai1sig
data_difference datafile1 datafile2 outfile i1 i2 ai1 ai2 ai1sig ai2dif
data_difference datafile1 datafile2 outfile i1 i2 ai1 ai2 ai1sig b1 b2
```

Command line parameters:

- Reads data from `datafile1` and `datafile2` (both `.node` or `.ele` files).
- If `i1` and `i2` are present and  $> 0$  then only data indices between and including `i1` and `i2` are included in the summary statistics (the information printed to screen), but anything written to file calculates differences for all data. If  $i1 \leq 0$  then `i1` defaults to 1. If  $i2 \leq 0$  or is absent then `i2` defaults to the number of data.
- If `ai1` and `ai2` are present then the only difference calculated is that between attribute (data column) `ai1` in the first data file and attribute `ai2` in the second. If `ai2` is absent then it defaults to `ai1`.
- If `ai1` and `ai2` are absent then differences are calculated for all shared attribute columns (see notes below).
- If `ai1sig` is present and  $> 0$  then the data differences are divided by the values in attribute `ai1sig` of the first file. Hence, to calculate the misfit between observed and predicted data, normalized by uncertainties, use a command like `data_difference observed predicted outfile ai1 ai2 ai1sig`
- If `ai2dif` is present and  $> 0$  then the normalized data difference “Diff” (see notes below) is inserted into attribute `ai2dif` of `datafile2`. Any existing attribute name is left unaltered. This feature is helpful if you have performed a gravity gradiometry inversion with a reduced set of components, and have then forward modelling the response of the recovered model for all components, and you then want to use `data_difference` to calculate the normalized data residuals. At present, you have to manually alter the attribute names for the normalized data residuals.
- If `ai1` and `ai2` are set  $\leq 0$  then `ai1sig` is ignored and the program calculates the difference between the data elevations. If `ai1` and `ai2` are set  $< 0$  then the elevations are not actually changed in the output files, but the calculated differences are still added to the attributes in the output files.
- If `b1` and `b2` are supplied then they multiply the values taken from the specified attributes of the input data files before the difference is calculated.

Outputs:

- Statistical information about the attribute differences is printed to the screen.
- If `outfile` is present and not equal to `null` then one or more output files are written.
- If `outfile` contains an extension or `ai1` and `ai2` are absent then only a single file is written and the attribute names are altered to have suffixes “Diff”, “AbsDiff” and “PerAbsDiff” (see notes below for definitions). When run in this mode,

if **ai1** and **ai2** are absent then there will be  $3n$  attribute columns in the output file, where  $n$  is the number of shared attribute columns (see notes below). If **ai1** and **ai2** are present then there will be 3 attribute columns in the output file.

- If **outfile** does not contain an extension and **ai1** and **ai2** are present then three files are written with attribute names unaltered and the files have suffixes as above.

Notes:

- If the two data files do not have the same number of attributes then the first  $n$  attributes are used where  $n$  is the number of attributes in the file with the least number of attributes.
- Differences are calculated between the first and second data files. If  $d_1$  is a data value in some row and attribute column from the first data file, and  $d_2$  is the corresponding value from the second data file, then data differences are calculated as follows:
  - “Diff” corresponds to  $d_2 - d_1$
  - “AbsDiff” corresponds to  $|d_2 - d_1|$
  - “PerAbsDiff” corresponds to  $|d_2 - d_1|/|d_1|$
- If **b1** and **b2** are supplied then “Diff” corresponds to  $b_2d_2 - b_1d_1$ , etc.
- If attribute names differ in the two data files, the attribute names used in the output file are those taken from the 2nd datafile.

### 4.3.6 data\_histogram

Reads data from a file and prints histogram information for a specified attribute column.

Command line usage:

```
data_histogram filename ai nbin
```

Command line parameters:

- Data is read from file **filename**, which can be a **.node** or **.ele** file. Any other extension is treated as a simple column-based data file with the same number of values on every line.
- **ai** is an index specifying the attribute column to use.
- **nbin** is the number of bins to use. The bins are spaced evenly within the data range.

### 4.3.7 lin\_reg

Performs linear regression on two sets of numbers, e.g. data or model values.

Command line usages:

```
lin_reg infile outfile
lin_reg infile outfile ix iy
lin_reg infile outfile ix iy a b
```

Command line parameters:

- The **infile** should be a column-based ASCII data file from which the values to use for the linear regression calculation will be taken.
- Indices **ix** and **iy** specify the columns in the **datfile** to use for the independent (x) and dependent (y) variables, respectively. If **ix** and **iy** are missing from the command line usage then the first and second columns are used, respectively.
- An ASCII column-based data file named **outfile** is written (see outputs below).
- If parameters **a** and **b** are provided then the linear regression is not calculated. Instead, **a** is taken as the slope of the linear relationship, **b** is taken as the y-intercept of the linear relationship, and the residual information is calculated (see outputs below).

Outputs:

- If parameters **a** and **b** are absent from the command line usage then the calculated slope and y-intercept linear parameters are printed to the screen, as is the calculated correlation value.
- The  $R^2$  value is printed to the screen (sum of squares of the residual).

- An ASCII column-based data file named **outfile** is written with four columns:
  1. the data for the independent variable (x)
  2. the data for the dependent variable (y)
  3. the residual =  $ax + b - y$
  4. the absolute value of the residual

Notes:

- Program **lin\_reg** is a version of **linear\_regression** that has minimal dependence on other modules.
- There are hardwired values used in this program for the maximum number of rows (1,000,000) and columns (100) allowed in the input data file. Increasing those hardwires is simple; please inform the developers, or you can do it yourself if you have the code.
- See also program **remove\_trend**.

#### 4.3.8 linear\_regression

Performs linear regression on two sets of numbers, e.g. data or model values.

Command line usages:

```
linear_regression inroot outroot ix iy
linear_regression inroot outroot ix iy a b
```

Command line parameters:

- The data values are taken from **.node** format file **inroot.node**.
- Indices **ix** and **iy** specify the coordinate or attribute columns in file **inroot.node** to use for the independent (x) and dependent (y) variables, respectively. If these indices are positive then they specify attribute columns. If they are negative then their absolute values specify coordinate columns.
- A **.node** format file named **outroot.node** is written (see outputs below).
- If parameters **a** and **b** are provided then the linear regression is not calculated. Instead, **a** is taken as the slope of the linear relationship, **b** is taken as the y-intercept of the linear relationship, and the residual information is calculated (see outputs below).

Outputs:

- If parameters **a** and **b** are absent from the command line usage then the calculated slope and y-intercept linear parameters are printed to the screen, as is the calculated correlation value.
- The  $R^2$  value is printed to the screen (sum of squares of the residual).
- A **.node** format file named **outroot.node** is written with four attribute columns:
  1. the data for the independent variable (x)
  2. the data for the dependent variable (y)
  3. the residual =  $ax + b - y$
  4. the absolute value of the residual

Notes:

- Program **lin\_reg** is a version of **linear\_regression** that has minimal dependence on other modules.
- See also program **remove\_trend**.

#### 4.3.9 matrix\_sums

Reads a matrix file and outputs various information about the matrix.

Command line usage:

```
matrix_sums infile
```

Command line or input file parameters:

- **infile** is the name of a file containing a full matrix in the following format:

```
nrow ncol
A(1,1)
A(2,1)
...
A(nrow,1)
A(1,2)
A(2,2)
...
A(nrow,ncol)
```

where **nrow** and **ncol** are the number of rows and columns in the matrix **A**.

Outputs:

- Currently, all this program does is print the following information to the terminal/command window:

```
1 n_1 s_1
2 n_2 s_2
...
nrow n_nrow s_nrow
```

where the **n** values are the number of non-zero elements counted in each row and the **s** values are the sums of the elements in each row.

#### 4.3.10 mean\_coordinates

Calculates the averages of the coordinate columns in two or more node files.

Command line usage:

```
mean_coordinates n noderoot1 noderoot2 ... noderootn outroot
```

Command line parameters:

- Reads **n** **.node** files named **noderoot1.node**, **noderoot2.node**, ... , **noderootn.node** and writes the averaged coordinates to file **outroot.node**.

Notes:

- All files must contain the same number of nodes.
- Any node attributes in input file **noderoot1.node** are written to the output file **outroot.node**.
- Currently, at most 8 files can be summed. This can easily be changed.

#### 4.3.11 print\_coordinates

Reads data from a file and prints coordinate and attribute range information (minimum, maximum values, etc.).

Command line usage:

```
print_coordinates file
```

Command line parameters:

- Data is read from file **file**, which can be a **.node** or **.ele** file. Any other extension is treated as a simple column-based data file with the same number of values on every line.

Outputs:

- Attribute range information is listed (**v1,v2,dv,s1,s2**) with **v1** and **v2** the minimum and maximum values, **dv** the range (maximum value minus minimum value), and **s1** and **s2** the sum and sum-of-squares respectively.

### 4.3.12 sum\_node

Sums the attribute column(s) in two or more node files.

Command line usage:

```
sum_node rms n noderoot1 noderoot2 ... noderootn outroot
```

Command line parameters:

- Reads `n` `.node` files named `noderoot1.node`, `noderoot2.node`, ... , `noderootn.node` and writes the summed attribute columns to file `outroot.node`.
- If `rms` is `t` (true) then the result is the square-root of the sum-of-squares.

Notes:

- All files must contain the same number of nodes and attributes.
- Node coordinates are not checked for consistency.
- Node coordinates in input file `noderoot1.node` are written to the output file `outroot.node`.
- Currently, at most 8 files can be summed. This can easily be changed.

### 4.3.13 terrain\_correction

Performs terrain correction for vertical component gravity data. Requires that the data response for the topography has already been computed.

Command line usages:

```
terrain_correction datafreeroot datatoporoot outroot
terrain_correction datafreeroot datatoporoot outroot dotrend
terrain_correction datafreeroot datatoporoot outroot dotrend aifree
```

Command line parameters:

- File `datafreeroot.node` should contain the free air anomaly data.
- File `datatoporoot.node` should contain the response of the topography (and sides and bottom of the modelling volume) for unit density. That response should be in the first attribute column.
- If integer parameter `aifree` is provided then the free air anomaly is taken from attribute column number `aifree` in file `datafreeroot.node`, otherwise the first attribute column is used.
- The program calculates the background density that provides the best fit to the free air data. If `dotrend` is `"t"` (true, the default) then the computation determines an additional linear trend to remove from the free air anomaly.

## 4.4 Interpolation and projection

See also program [interpolate\\_mesh](#).

### 4.4.1 interpolate\_data

Data interpolation at specified points.

Command line usages:

```
interpolate_data noderoot eleroot interproot outroot 1 ai1
interpolate_data noderoot eleroot interproot outroot nai ai1 ... ain
interpolate_data noderoot eleroot interproot outroot nai ai1 ... ain usez
interpolate_data noderoot eleroot interproot outroot nai ai1 ... ain usez drape
interpolate_data noderoot eleroot interproot outroot nai ai1 ... ain usez drape pow
interpolate_data noderoot eleroot interproot outroot nai ai1 ... ain usez drape pow nl
```

Command line parameters:

- Reads topography or geophysical data from files `noderoot.node` and `eleroot.ele`.
- Reads the points at which to interpolate from files `interproot.node`.
- If `eleroot` specifies an existing file then linear interpolation is used across the triangulated surface facets (3D) or line elements (2D) in files `noderoot.node` and `eleroot.ele`.
- If `eleroot` is `null` then inverse-distance-weighting (IDW) is used for the interpolation.
- The `nai` parameter specifies how many attribute indices `ai1 ... ain` you want to use.
- The `ai1 ... ain` parameters specify what to interpolate:
  - A value of 0 specifies that the elevations should be interpolated. Duplicate instances of 0 are ignored.
  - Positive integer values specify the indices of node attributes to interpolate. Duplicate instances of a positive integer causes duplicate interpolation (duplicate attribute columns will exist in the output `.node` file).
  - Negative values are ignored or may cause an error.
- If `usez` is `"f"` (false, the default) then the interpolation looks only at the (x,y) coordinates for 3D problems (and only the x coordinates for 2D problems). For a 3D problem, you would set `usez` to `"f"` (false) if, for example, you wanted to interpolate the elevations from some topography data at some specified (x,y) points.
- If `usez` is `"t"` (true) then the interpolation looks at the (x,y,z) coordinates for 3D problems (x and z for 2D problems). For a 3D problem, you would set `usez` to `"t"` (true) if, for example, you used program [cells2nodes](#) to generate a cloud of nodes at the cell centroids of a mesh and you then wanted to interpolate some attribute at some specified points within the mesh volume.
- If `usez` is `"t"` (true) then parameter `drape` is ignored.
- If `drape` is `"t"` (true) then a draping over/under topography is performed, which just adds the existing `z` values to the interpolated values. The default is `"f"` (false).
- The `pow` parameter specifies the interpolation power for IDW:
  - The default value is the number of dimensions.
  - `pow`  $\ll$  `ndim` (with `ndim` the number of dimensions) cause the interpolated values to be dominated by points far away.
  - `pow`  $\gg$  `ndim` causes the interpolated values to approach a piece-wise character (nearest neighbour interpolation, like a Voronoi diagram).
- `nl` is a looping parameter used when `eleroot` is not `null` (for developers: this looping parameter can be found inside subroutine `ugrid_find_cells` in module `ugrid_cls`). You may need to increase this parameter above 1 to obtain good results, but higher values can mean much longer run times. For more explanation, see the information for program [interpolate.mesh](#).

Outputs:

- Writes the new nodes (with interpolated elevations or data) to `outroot.node`. An additional output file `outroot_2d.node` contains 2D (x,y) points for use with Triangle (3D problems only). If `eleroot` is `null` or `drape` is `t` then that additional `.node` file is not written.

#### 4.4.2 project\_geomag

Projects a geomagnetic reference field vector onto a vertical cross-section.

Command line usage:

```
project_geomag x1 y1 x2 y2 inc dec str
```

Command line parameters:

- The cross-section runs from (easting,northing) point (x1,y1) to (x2,y2).
- `inc` is the inclination of the geomagnetic field in degrees (positive angles below horizontal).
- `dec` is the declination of the geomagnetic field in degrees (zero toward north, positive east of north).
- `str` is the strength of the geomagnetic field (e.g. in nT).

Outputs:

- Prints the projected inclination and strength along the cross-section direction.

## 4.5 Utilities for seismic and muon tomography data

### 4.5.1 check\_combos

Takes three files defining seismic sources, receivers and source-receiver combinations and generates a `.vtu` file showing the connections.

Command line usage:

```
check_combos sourcesroot receiversroot combosroot outroot [zrev]
```

Command line parameters:

- Sources are read from file `sourcesroot.node`.
- Receivers are read from file `receiversroot.node`.
- Source-receiver combinations are read from file `combosroot.ele`. This should be a 2D `.ele` format file with 2 nodes per cell. Each line after the header should specify one source-receiver combination. The first index column should specify the source indices and the second the receiver indices: those indices correspond to nodes in the `sourcesroot.node` and `receiversroot.node` files. An example file follows.

```
42 2 0
1 1 1
2 1 2
3 1 3
4 2 1
5 2 2
...
```

- If `combosroot` is `all` then all possible source-receiver combinations are used.
- If `combosroot` is `match` then the  $i^{th}$  source is matched with the  $i^{th}$  receiver.
- The output file is named `outroot.vtu`. Opening it in ParaView will display sources and receivers as points and connections between them as straight lines.
- Set the optional `zrev` parameter to `t` (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.

### 4.5.2 pierce\_points

Takes files defining a 3D unstructured mesh, seismic or muon tomography sources, receivers and source-receiver combinations and generates a `.node` file containing the pierce points that lie strictly on or between the source and receivers (along straight ray paths).

Command line usage:

```
pierce_points meshroot sourcesroot receiversroot combosroot outroot [zrev]
```

Command line parameters:

- The mesh is read from files `meshroot.node`, `meshroot.ele`, and `meshroot.neigh` if it exists.
- Sources are read from file `sourcesroot.node`.
- Receivers are read from file `receiversroot.node`.
- Source-receiver combinations are read from file `combosroot.ele`. This should be a 2D `.ele` format file with 2 nodes per cell. Each line after the header should specify one source-receiver combination. The first index column should specify the source indices and the second the receiver indices: those indices correspond to nodes in the `sourcesroot.node` and `receiversroot.node` files. An example file follows.

```

42 2 0
1 1 1
2 1 2
3 1 3
4 2 1
5 2 2
...

```

- If `combosroot` is `all` then all possible source-receiver combinations are used.
- If `combosroot` is `match` then the  $i^{th}$  source is matched with the  $i^{th}$  receiver.
- The output files are named `outroot.node` and `outroot.ele`.
- Set the optional `zrev` parameter to `t` (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.

#### Outputs:

- Output file `outroot.node` contains all the pierce points. It contains an attribute named “ComboIndex” that indicates the source-receiver index associated with each pierce point. Another attribute named “NumberOfPiercePoints” indicates the number of pierce points associated with each pierce point; that is, all the pierce points for a particular source-receiver combination will have the same value for attribute “NumberOfPiercePoints” equal to the number of pierce points for that combination.
- Output file `outroot.ele` contains the source-receiver combination information. It contains an attribute named “NumberOfPiercePoints” that indicates the number of pierce points for each combination. If there are no pierce points for a particular source-receiver combination then the direct path between that source and receiver does not pass through the mesh. If there is an odd number of pierce points for a particular combination then something has gone wrong; one possibility is that the path just touches the surface of the mesh but does not pass through it, in which case the odd number of pierce points may result from a machine precision problem in the code that is difficult to avoid.

### 4.5.3 split\_tobs

Splits a traveltime data file into three files containing sources, receivers and traveltime information.

Command line usages:

```

split_tobs infile ndim
split_tobs infile ndim outroot

```

Command line parameters:

- `ndim` should be 2 or 3 to specify whether it is a 2D or 3D problem.
- If `infile` is a `.node` file then it should specify 4 or 6 dimensions in the header with data rows in this format:

```

sx sz rx rz t
or

```

```

sx sy sz rx ry rz t

```

where `(sx,sy,sz)` defines a source coordinate, `(rx,ry,rz)` a receiver coordinate and `t` a traveltime. If `ndim` is 3 then the second option must be used. If `ndim` is 2 then either option can be used: if the second is used then the `sy` and `ry` coordinate values are ignored.

- If `infile` is not a `.node` file then it should be a 2D traveltime pick database file with the format below and `ndim` must be 2.

```

# Any amount of lines (but at least one) starting with any character other than <.
# The following lines specify the source locations:
< is1 xs1 zs1 ...
< is2 xs2 zs2 ... (anything else on these data lines is ignored)
< is3 xs3 zs3 ...
...
# Any amount of lines (but at least one) starting with any character other than <.

```



```

# The following lines specify the traveltime picks for the first source:
< it1 t1 xr1 zr1 ...
< it2 t2 xr2 zr2 ... (anything else on these data lines is ignored)
< it3 t3 xr3 zr3 ...
...
# Any amount of lines (but at least one) starting with any character other than <.
# The following lines specify the traveltime picks for the second source:
< it1 t1 xr1 zr1 ...
< it2 t2 xr2 zr2 ... (anything else on these data lines is ignored)
< it3 t3 xr3 zr3 ...
...
# Any amount of lines (but at least one) starting with any character other than <.
...
```

where **is1** is a source index, (**xs1,zs1**) a source location, **it1** a traveltime pick index, (**xr1,zr1**) a receiver location and **t1** a traveltime. There must be the same number of receivers connected to every source.

#### Outputs:

- The input file is split into three files:
  - 1) `outroot_sources.node`
  - 2) `outroot_receivers.node`
  - 3) `outroot_ttime_data.ele`
- If `outroot` is absent then the file root of `infile` is used (extension stripped off).

# Chapter 5

## Model processing

### 5.1 Model creation and manipulation

#### 5.1.1 add\_blocks

Adds rectangular prisms, spheres and horizontal discs inside models on rectilinear or unstructured meshes. When adding blocks (rectangular prisms) to a UBC-GIF format model, this is like you were using the UBC-GIF MeshTools utility program.

Command line usages:

```
add_blocks meshfile modelfile shapesfile
add_blocks meshfile modelfile shapesfile ai
add_blocks meshfile modelfile outfile shapesfile
add_blocks meshfile modelfile outfile shapesfile ai
```

Command line parameters:

- **meshfile** and **modelfile** are UBC-GIF format mesh and model files (for rectilinear meshes) or **.node/.ele** files (for unstructured meshes). The **modelfile** must exist for unstructured meshes. If the **modelfile** file exists then the shapes are added to the specified attribute (parameter **ai**) of the existing model, replacing any existing values in the specified model cells. If the **modelfile** file does not exist (rectilinear meshes only) then the model values are set to the specified background value (**v0**) in the **shapesfile**.
- **outfile** is the name of the output model file after adding the shape(s). If it is not present in the command line usage then it is set to **modelfile** and that file will be overwritten.
- **shapesfile** is a file of format below. The first line specifies three values: **n** is the number of shapes, **ndim** is the number of dimensions (2 or 3), and **v0** is the background attribute value. The format of each following **n** lines depends on the shape being added:
  - For blocks (rectilinear prisms) placed inside a rectilinear mesh, the line specifies start and end directional indices for the block and the attribute value for the block:  
**i\_start i\_end [j\_start j\_end] k\_start k\_end attribute\_value**
  - Alternatively, for either type of mesh, blocks can be specified with the "b" character followed by centroid location and dimensions in each Cartesian direction:  
**b x [y] z dx dy dz attribute\_value**
  - For spheres, in either type of mesh, the line has the "s" character followed by the centroid location, radius and attribute value:  
**s x [y] z radius attribute\_value**
  - For horizontal discs, in either type of mesh, the line has the "d" character followed by the centroid location, radius, vertical thickness and attribute value:  
**d x [y] z radius thickness attribute\_value**
- Those centroid coordinates should be entered in a +z-up system (+x is east, +y is north, +z is up).
- Integer parameter **ai** specifies a model attribute index to use. If **ai** is absent from the command line usage then the first attribute is used.

Example shape specification file with only rectilinear prisms:

```

n ndim v0
i11 i12 [j11 j12] k11 k12 v1
i21 i22 [j21 j22] k21 k22 v2
...
in1 in2 [jn1 jn2] kn1 kn2 vn

```

Notes:

- With the exception of blocks inside rectilinear meshes, all shapes will be pixellated representations inside the meshes.
- For rectilinear meshes, if `modelfile` is `null` or `NULL` or the specified file does not exist then `ai` is ignored. For unstructured meshes, if `modelfile` has no attributes then `ai` is ignored. In either case, the output model file will contain a single attribute and a background value of `v0` (specified in the `shapesfile`) is applied throughout the model before adding the shapes.
- If there is more than one model in the existing `model` file and `ai` is absent then only the first model attribute is written to the output file.
- If there is more than one model in the existing `model` file and `ai` is present then all the model attributes are written to the output file.
- The specified shapes are added in the order in which they appear in the `shapesfile`. Hence, later shapes may overlay earlier shapes by replacing any values in the specified model cells.
- Warning: the algorithm for adding alternatively specified shapes is not smart, it cycles over every cell, so this program may run very slowly for large meshes. However, tests on one million cells ran fast enough.

Example:

- The following file adds a single 2x3x4 block with attribute value 6.7 to a model. The top left corner of the block is one cell away, in all directions, from the top left corner of the mesh.

```

1 3 0.0
2 3 2 4 2 5 6.7

```

### 5.1.2 boundary\_outline

Determines the boundary outline of a meshed surface.

Command line usage:

```
boundary_outline meshroot inode [outroot]
```

Command line parameters:

- Files `meshroot.node/.ele/.neigh` define the meshed surface. If the `.neigh` file does not exist then cell neighbour information is calculated automatically, which may increase the run-time significantly for large meshes.
- `inode1` should be an index specifying a node on the boundary. The user must determine a correct value for this parameter.
- If `outroot` is present then the result is written to files `outroot.node/.ele/.neigh`. Otherwise, the input files `meshroot.node/.ele/.neigh` are overwritten.

Notes:

- The output files have the nodes reordered so that the boundary outline nodes are listed first.
- The number of boundary nodes is printed to the screen.
- The algorithm used will fail if:
  1. `inode1` specifies a node that is not on the boundary
  2. there are squashed cells on the boundary (all nodes in the same zero-area cell).

### 5.1.3 build\_tetmesh

Allows the user to build tetrahedral meshes following typical forward and inverse modelling requirements.

Command line usage:

```
build_tetmesh inputfile outputroot
```

Command line parameters:

- File **inputfile** defines various aspects of the meshing requirements. Run the program without any input parameters for more information on the format of that input file.
- Output files are named with the prefix **outputroot**.

### 5.1.4 clean\_surface

Performs several cleaning operations on a surface model:

- removes duplicate nodes
- removes cells with duplicated node indices
- removes duplicate cells
- removes isolated pieces of the model, with respect to the number of cells in each model piece, assuming only the largest piece should be kept
- removes cells with zero area or volume
- removes unused nodes.

Command line usage:

```
clean_surface noderoot eleroot outroot
```

Command line parameters:

- Files **noderoot.node** and **eleroot.ele** define the input surface.
- Files **outroot.node/.ele** are written with the cleaned surface.

### 5.1.5 coarsen\_model

Coarsens a mesh-based model using a thresholding, grouping, and averaging method.

Command line usage:

```
coarsen_model meshfile modelfile outfile ai v
```

Command line parameters:

- Files **meshfile** and **modelfile** define the model.
- If the mesh is unstructured then **meshfile** should be a **.node** file and **modelfile** a **.ele** file (the extensions must be included in the command line usage).
- If the mesh is rectilinear then those files should be UBC-GIF format (the extensions can be anything).
- If **modelfile** has extension **.ele** then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- Integer parameter **ai** specifies which model attribute to coarsen.
- Real value parameter **v** specifies a threshold value to use.
- The coarsened model information is written to file **outfile**. All attributes in the original mesh will exist in the output file but only the specified attribute will have been coarsened. To coarsen multiple attributes, run this program multiple times.

Algorithm:

- The program first thresholds the specified model attribute, assigning the cells an ID value of 1 anywhere the model attribute is below the threshold value **v**, and an ID value of 2 elsewhere.

- Then, groups of contiguous cells with the same ID values (1 or 2) are found. This will usually yield one large group of cells with an ID value of 1, corresponding to cells with attribute below the threshold, and several groups of cells with an ID value of 2, corresponding to cells with attribute above the threshold.
- For each group of contiguous cells with an ID value of 1, the program sets the output model to 0.0, indicating cells with attribute below the threshold.
- For each group of contiguous cells with an ID value of 2, the program sets the output model to the average attribute value for those cells.

### 5.1.6 conform\_topography

Cuts the lateral sides of a single 3D boundary block by a topography surface.

Command line usages:

```
conform_topography blocksfile toporoot outroot retri bm
conform_topography elevation toporoot outroot retri bm
```

Command line parameters:

- The first command line usage reads information about the single 3D boundary block from **blocksfile**. The block file format is explained in [Section 2.8](#). Only a single non-rotated boundary block is allowed.
- The second command line usage automatically generates a single 3D block with sides at the topography limits and bottom at the specified **elevation** (in a +z up coordinate system).
- Reads the topography surface from **toporoot.node** and **toporoot.ele**.
- The coordinate system in the block specification and topography files are assumed to both have +z up.
- Writes the new topography surface, cut by and sewn to the boundary block, to **outroot\_topo.node** and **outroot\_topo.ele**.
- Writes the new boundary facets, cut by the topography on the lateral sides, to **outroot\_boundary.node** and **outroot\_boundary.ele**.
- If **retri** is **f** (false) then the topography nodes/cells outside the boundary are removed and the remaining nodes on the outline of the resulting topography surface are moved onto the boundary. This maintains the existing topography surface everywhere except close to the boundary but may result in poor-quality triangles on the boundary. When poor-quality triangles are an issue, it is better to use Triangle to remesh the topography surface (see next item).
- If **retri** is **t** (true) then Triangle is used to mesh the topography to the boundary. Any existing surface facets may be scrapped. Therefore, I do not recommend this approach. If this approach is required, it would be better for you to run Triangle yourself and then pass the result into **conform\_topography** with **retri** set to **f**. The procedure is as follows:
  1. Generate a 2D **.poly** file containing the topography nodes and line-elements defining the boundary outline (the 3D scenario is projected into a 2D scenario along the vertical axis).
  2. Mesh that **.poly** file with Triangle, using whatever flags you wish.
  3. Interpolate elevations from the original topography surface to the new one (generated by Triangle) using program [interpolate\\_data](#).
  4. Run **conform\_topography** with **retri** set to **f** such that none of the new topography nodes are moved.
- Parameter **bm** specifies an integer boundary marker value. The boundary markers in the input **toporoot.node** file are used in the output **outroot\_topo.node** file. However, any nodes in the output **outroot\_topo.node** file that lie on the input block will have boundary markers equal to the specified **bm** value. If the input **toporoot.node** file does not have boundary markers then any nodes in the output **outroot\_topo.node** file that do not lie on the input block will have boundary markers equal to zero.

Notes:

- The topography surface should extend exactly to the sides of the boundary block (ideal) or beyond it.
- The program may fail if the topography surfaces does not cover the lateral extents of the boundary block.
- Only a single non-rotated boundary block is currently allowed. If you need a rotated boundary block (rotated with respect to the Cartesian axes) then you can make use of program [transform\\_coordinates](#) to rotate the topography information before and after running **conform\_topography**.

- The side and bottom facets of the conformed block have normal vectors pointing away from the centre of the block (outward pointing normals). I'm not sure about the top facet; please contact me and I'll figure it out, but usually the top facet (above topography) gets removed in standard processing for potential field problems.
- If you want to conform more than one surface to some boundary block/range then use a combination of programs [snap\\_surface](#), [sew\\_surfaces](#) and [combine\\_files](#).

### 5.1.7 count\_cells

Counts the cells and calculates a volume or area sum for a collection of triangular or tetrahedral cells.

Command line usages:

```
count_cells noderoot eleroot
count_cells noderoot eleroot ai v
```

Command line parameters:

- Files `meshroot.node/.ele` define a model on an unstructured mesh.
- The number of cells and summed volume (for 3D tetrahedra) or summed area (for 2D or 3D triangles) are printed to the terminal window.
- If the optional `ai` and `v` parameters are present then only cells with attribute number `ai` equal to `v` are included in the counting and volume or area summation.

### 5.1.8 mark\_model

This program has been absorbed into program [populate\\_model](#) and is now obsolete.

### 5.1.9 mesh\_core

Extracts the core (removes padding cells) from a UBC-GIF mesh.

Command line usages:

```
mesh_core meshfile1 modelfile1 meshfile2 modelfile2 x1 x2 z1 z2
mesh_core meshfile1 modelfile1 meshfile2 modelfile2 x1 x2 y1 y2 z1 z2
```

Command line parameters:

- The first command line usage above is for a 2D mesh and the second is for a 3D mesh.
- The input rectilinear mesh is read from the UBC-GIF format mesh file `meshfile1`.
- The input model is read from the UBC-GIF format model file `modelfile1`.
- The mesh and model with padding cells removed are written to UBC-GIF format files `meshfile2` and `modelfile2` respectively.
- For a 3D mesh:
  - `x1` is the number of cells to remove from the west side of the mesh
  - `x2` is the number of cells to remove from the east side of the mesh
  - `y1` is the number of cells to remove from the south side of the mesh
  - `y2` is the number of cells to remove from the north side of the mesh
  - `z1` is the number of cells to remove from the top of the mesh
  - `z2` is the number of cells to remove from the bottom of the mesh.
- For a 2D mesh:
  - `x1` is the number of cells to remove from the left side of the mesh
  - `x2` is the number of cells to remove from the right side of the mesh
  - `z1` is the number of cells to remove from the top of the mesh
  - `z2` is the number of cells to remove from the bottom of the mesh.

### 5.1.10 mesh\_faces

Converts an unstructured mesh of triangular (2D) or tetrahedral (3D) cells to one containing line element (2D) or triangular (3D) cell face information. This may be helpful for visualization of an unstructured mesh in Gocad, combined with program [convert\\_format](#); another option is to interpolate onto a fine rectilinear mesh using program [interpolate\\_mesh](#).

Command line usage:

```
mesh_faces noderoot eleroot faceroot [a]
```

Command line parameters:

- The mesh is read from files `noderoot.node` and `eleroot.ele`, and `noderoot.neigh` or `eleroot.neigh` if such file exists.
- The output mesh is written to files `faceroot.node` and `faceroot.ele`.
- Parameter `a` is an integer value that specifies how the cell attributes are interpolated onto the faces. If `a = 0` then attributes in neighbouring cells are averaged. If `a > 0` then the maximum is taken. If `a < 0` then the minimum is taken.

### 5.1.11 outlines2plc

Generates a 3D poly file containing outlines specified in 2D input files. Designed for micromagnetic work on thin sections of drillcore.

Command line usages:

```
outlines2plc meshroot outroot thickness
```

```
outlines2plc noderoot eleroot outroot thickness
```

Command line parameters:

- A 2D unstructured mesh is read from files `meshroot.node/.ele` (first command line usage), or `noderoot.node` and `eleroot.node` (second command line usage).
- Those files should specify edge element cells that indicate closed 2D polygonal outlines around each grain in a micromagnetic sample. The ordering is important. The following assumptions are made:
  - All nodes in the node file are ordered by outline.
  - Each edge should be in logical order around the grain outline such that the second node index of an edge is the first node index of the next.
  - Each outline must be explicitly closed, so the second node index of the final edge in an outline must equal the first node index of the first edge in that outline.

An example of an input `.ele` file is provided below, in which the first outline contains 4 nodes and the second has 3.

- The 2D polygonal outlines are used to build 3D representations of the grains with the specified `thickness`.
- The resulting 3D PLC is written to files `outroot.poly/.vtu`.

```
7 2 0 0
1 1 2
2 2 3
3 3 4
4 4 1
5 5 6
6 6 7
7 7 5
```

### 5.1.12 pad\_mesh

Adds padding cells to the sides and bottom of a UBC-GIF mesh using a user-defined expansion ratio. The distance to pad in each direction is determined automatically: the padding extends at least as far as the dimensions of the existing mesh.

Command line usage:

```
pad_mesh meshfile outfile fac
```

Command line parameters:

- File **meshfile** is a UBC-GIF format rectilinear mesh file.
- The padded mesh is written to **outfile**.
- A **.vtr** file is also written for viewing in ParaView: the name of that file is the same as **outfile** but with the file extension replaced.
- **fac** is the expansion ratio.

### 5.1.13 populate\_model

Populates a voxel model with values at specified points. Does essentially the opposite of program [query\\_model](#).

Command line usages:

```
populate_model noderoot1 meshfile2 modelfile2 outroot
populate_model noderoot1 meshfile2 modelfile2 outroot "defs ..."
populate_model noderoot1 meshfile2 modelfile2 outroot ai1 ai2
populate_model noderoot1 meshfile2 modelfile2 outroot ai1 ai2 add
```

Command line parameters:

- File **noderoot1.node** defines the point locations at which the model is populated.
- Files **meshfile2** and **modelfile2** define the model that gets populated.
- If the mesh is unstructured then **meshfile2** should be a **.node** file and **modelfile2** a **.ele** file.
- If the mesh is rectilinear then those files should be UBC-GIF format and the **.node** file is assumed to be in a +z up coordinate system (*x* is easting, *y* is northing, *z* is elevation).
- If **modelfile2** has extension **.ele** then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- Parameter **outroot** specifies the root name for the output model file written by the program. The same extension on file **modelfile2** is used for the output model file.
- If the "defs ..." list is provided, it should be a list of default numerical values for each attributed being populated, in the order they appear in file **noderoot1.node**. The list must be surrounded by double quotes. If this list is not provided then the program will prompt the user to enter the required values in the command line when necessary.
- Parameters **ai1** and **ai2** specify attribute columns in files **noderoot1.node** and **modelfile2** respectively (see notes below).
- If **add** is present and non-zero then this additional value is added to any populated values.

Notes:

- The program finds cells in the mesh that contain the specified node points and populates those cells with the node attributes.
- If multiple node points are found in a single cell then the attributes for those nodes are averaged.
- For the first and second command line usages above, all node attributes are populated into the model. The model values are set to defaults before populating the model.
- For the third and fourth command line usages above, attribute number **ai1** in file **noderoot1.node** is used to populate attribute number **ai2** in file **modelfile2**. The existing values of attribute number **ai2** in file **modelfile2** are overwritten for cells containing the node point locations.
- WARNING: for the third and fourth command line usages above, if attribute number **ai2** does not exist in file **modelfile2** then all cell attributes except the first are cleared; the first attribute is set to zero and then populated using the node information.
- If a node is not within the bounds of the mesh then it is ignored.



### 5.1.14 remove\_cells

Removes any cells in an unstructured mesh with a specified attribute value. See also program [remove\\_nodes](#).

Command line usages:

```
remove_cells meshroot outroot ai v
remove_cells meshroot outroot ai v keep
remove_cells noderoot eleroot outroot ai v
remove_cells noderoot eleroot outroot ai v keep
```

Command line parameters:

- In the first two command line usages, files `meshroot.node/.ele` define a model on an unstructured mesh.
- In the second two command line usages, files `noderoot.node` and `eleroot.ele` define the model on the unstructured mesh.
- If `ai`  $\neq 0$  then any cells with attribute `ai` (an attribute index) equal to value `v` are marked.
- If `ai` = 0 then cell `v` (a cell index) is marked.
- If `keep` is "f" (false, the default) then those marked cells are removed, otherwise they are kept (everything else is removed).
- Any nodes that are subsequently unused, i.e. those that do not appear in the remaining cell definitions, are removed.
- The remaining nodes and cells are written to files named `outroot.node/.ele`. If the nodes and cells do not change then no output files are written.
- If file `meshroot.neigh` or `noderoot.neigh` or `eleroot.neigh` exists then it is read and the altered cell neighbour information written to a file named `outroot.neigh`.

### 5.1.15 remove\_unused\_nodes

Removes unused nodes from a `.nodes` file and adjusts the cell definitions accordingly in a companion `.ele` file.

Command line usage:

```
remove_unused_nodes noderoot eleroot outroot
```

Command line parameters:

- Any nodes in file `noderoot.node` that are not indexed in the cell definitions in `eleroot.ele` are removed.
- The remaining nodes are written to `outroot.node` and the adjusted cell definitions to `outroot.ele`.
- The indices of the original nodes removed are printed to the screen.

### 5.1.16 reorder\_regions

Reorders regions of cells in a `.ele` file.

Command line usage:

```
reorder_regions inputroot outputroot ireg
```

Command line parameters:

- File `inputroot.ele` is read.
- Parameter `ireg` should be an integer representing a region identifier value.
- Any cells with the first attribute equal to `ireg` are moved to the end of the list of cells.
- The reordered list of cells is written to file `outputroot.ele`.

### 5.1.17 scoop\_model

This program is for setting particular regions of a model to constant values. It can be used for performing the scooping part of a regional removal procedure. Program [add\\_blocks](#) may be of more help to you for performing the scooping procedure on rectilinear meshes.

Command line usages:

```
scoop_model modelfile1 j1 v1a v1b modelfile2 j2 v2
scoop_model modelfile1 j1 v1a v1b modelfile2 j2 v2 outfile
```

Command line parameters:

- **modelfile1** and **modelfile2** can be **.ele** or UBC-GIF format model files (the latter is assumed for all file extensions other than **.ele**).
- The program first finds any cells in the first model (read from **modelfile1**) with attribute index **j1** within the range [**v1a,v1b**]. The program then sets attribute index **j2** to value **v2** for those same cells in the **second model** (read from **modelfile2**).
- **outfile** specifies the name of the output file to write the altered second model. If **outfile** is absent from the command line usage then file **modelfile2** is overwritten.
- With UBC-GIF format model files, set both **j1** and **j2** to 1, unless you are using my multi-column extension to the UBC-GIF model file format.

Notes:

- An error is thrown if the two model files do not have the same number of cells.
- The two model files can be the same file.

As an example of how this program might be used, consider the scooping procedure of [Li & Oldenburg \(1998b\)](#) on unstructured meshes. The basic idea is to take a model recovered from a regional scale inversion and set part of it to zero, i.e. “scoop” out some central core volume of interest. In that scenario, you would have used TetGen to create an inversion mesh with two regions: one region for the core volume of interest, the other region for the padding cells around the core volume that are required for the regional scale inversion. That mesh **.ele** file would correspond to **modelfile1** in the command line usage above. For this example, let’s say that **modelfile1.ele** file has a single attribute with values of 1.0 or 2.0, which label the cells as belonging to either the core or padded regions respectively. You would have another **.ele** file output from the inversion, which holds some physical property distribution. That second **.ele** file would correspond to **modelfile2** in the command line usage above. For this example, let’s say that second **modelfile2.ele** file has a single attribute (with values corresponding to some physical property). The required command line usage would then be as follows:

```
scoop_model modelfile1.ele 1 1.0 1.0 modelfile2.ele 1 0.0
```

Both model files are on the same mesh so contain the same number of values. What we are requesting here is that any cells with an attribute value of 1.0 in **modelfile1.ele** are assigned a zero-valued physical property in **modelfile2.ele**. File **modelfile1.ele** is used by the program to figure out which cells are in which region, and **modelfile2.ele** is the only file changed by the program.

### 5.1.18 sew\_surfaces

Takes two surfaces and sews their lateral sides together.

Command line usage:

```
sew_surfaces surfroot1 surfroot2 outroot
```

Command line parameters:

- Files **surfroot1.node/.ele** and **surfroot2.node/.ele** specify two pairs of files that define two 3D surfaces of tessellated triangles.

Outputs:

- Files **outroot.node/.ele/.vtu** contain the triangular facets from the input surfaces and the polygonal facets that are required to sew them up on the sides.

Assumptions:

- The two surfaces do not intersect but they can have overlapping elevation ranges.
- The two surfaces are trimmed to EXACTLY the same rectangular boundary.

### 5.1.19 `smooth_model`

Smooths a mesh-based model using a simple neighbour-averaging method. To smooth a surface-based model, try program [smooth\\_node](#).

Command line usages:

```
smooth_model meshfile modelfile outfile nsmooth
smooth_model meshfile modelfile outfile nsmooth ai
smooth_model meshfile modelfile outfile nsmooth ai av
smooth_model meshfile modelfile outfile nsmooth ai av1 av2
```

Command line parameters:

- Files `meshfile` and `modelfile` define the model.
- If the mesh is unstructured then `meshfile` should be a `.node` file and `modelfile` a `.ele` file (the extensions must be included in the command line usage).
- If the mesh is rectilinear then those files should be UBC-GIF format (the extensions can be anything).
- If `modelfile` has extension `.ele` then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- The smoothed information is written to file `outfile`.
- The smoothing occurs `nsmooth` times via a simple neighbour-averaging method.
- If `ai` is present, and an integer greater than zero, then it specifies which model attribute to smooth. If absent or non-positive, all attributes are smoothed.
- If `ai` is present and  $> 0$ , and `av` is present, then cells with `ai`-th attribute equal to `av` are not smoothed, nor are those cells used in the smoothing calculation for any of their neighbours.
- If `ai` is present and  $> 0$ , and `av1` and `av2` are present, then cells with `ai`-th attribute on the range `[av1,av2]` are not smoothed, nor are those cells used in the smoothing calculation for any of their neighbours.

### 5.1.20 `snap_surface`

Snaps a 3D surface to a given rectangular lateral boundary range.

Command line usage:

```
snap_surface surfroot outroot retri bm x1 x2 y1 y2
```

Command line parameters:

- Reads the surface from files `surfroot.node/.ele`.
- Writes the new snapped surface to files `outroot.node/.ele/.vtu/.poly`.
- If `retri` is `f` (false) then the surface nodes/cells outside the boundary range are removed and the remaining nodes on the outline of the resulting surface are moved onto the boundary. This maintains the existing surface everywhere except close to the boundary but may result in poor-quality triangles on the boundary. When poor-quality triangles are an issue, it is better to use Triangle to remesh the surface (see next bullet).
- If `retri` is `t` (true) then Triangle is used to mesh the surface to the boundary. Any existing surface facets may be scrapped. Therefore, I do not recommend this approach. If this approach is required, it would be better for you to run Triangle yourself and then pass the result into `snap_surface` with `retri` set to `f`. The procedure is as follows:
  1. Generate a 2D `.poly` file containing the surface nodes and line-elements defining the boundary outline (the 3D scenario is projected into a 2D scenario along the vertical axis).
  2. Mesh that `.poly` file with Triangle, using whatever flags you wish.
  3. Interpolate elevations from the original surface to the new one (generated by Triangle) using program [interpolate\\_data](#).

4. Run `snap_surface` with `retri` set to `f` such that none of the new surface nodes are moved.
- Parameter `bm` specifies an integer boundary marker value. The boundary markers in the input `surfroot.node` file are used in the output `outroot.node` file. However, any nodes in the output `outroot.node` file that lie on the boundary rectangle will have boundary markers equal to the specified `bm` value. If the input `surfroot.node` file does not have boundary markers then any nodes in the output `outroot.node` file that do not lie on the boundary rectangle will have boundary markers equal to zero.

Notes:

- The surface should ideally extend beyond the boundary range on all sides.
- The program may fail if the surface does not cover the lateral extents of the boundary range.

### 5.1.21 subdivide

Performs a surface subdivision on a 2D or 3D wireframe surface model or a 2D triangular mesh.

Command line usages:

```
subdivide inputroot outputroot nsub smooth
subdivide inputroot outputroot nsub smooth stat
subdivide inputroot outputroot nsub smooth stat w
subdivide inputroot outputroot nsub smooth stat w unitsfile
subdivide noderoot eleroot outputroot nsub smooth
subdivide noderoot eleroot outputroot nsub smooth stat
subdivide noderoot eleroot outputroot nsub smooth stat w
subdivide noderoot eleroot outputroot nsub smooth stat w unitsfile
```

Command line parameters:

- If `inputroot` is a full file name containing the `.poly` extension then the wireframe surface model or triangular mesh is read from that poly file.
- If `inputroot` does not contain the `.poly` extension then the wireframe surface model or triangular mesh is read from files `inputroot.node/.ele`.
- If `noderoot` and `eleroot` are provided then the wireframe surface model or triangular mesh is read from files `noderoot.node` and `eleroot.ele`.
- The subdivided wireframe surface model or triangular mesh is written to file `outputroot.poly` OR files `outputroot.node/.ele`, depending on the input file(s) used.
- `nsub` specifies how many times to perform the subdivision. It should be 1 or greater.
- If `smooth` is `"t"` (true) then the subdivided surface is smoothed.
- `stat` is only used for subdivision with smoothing:
  - if `"t"` (true, the default) then a Dyn-Levin-Gregory interpolation is used and the subdivided interpolated surface will pass through the control nodes (the nodes in the input model)
  - if `"f"` (false) then a Cubic B-Spline interpolation is used and the subdivided interpolated surface does not pass through the control nodes
- If `smooth` and `stat` are both `"t"` (true) then `w` specifies the tension for the 3D D.L.G. interpolation. The default value is 1/16, which is used if `w` is absent or  $\leq 0$ .
- A node units file `unitsfile` with an attribute named `Dynamic` can be used to specify whether or not (values 1 or 0) a node can move. This information is only used if the node information is read from a `.node` file.

Notes:

- If node boundary markers are present in the `.poly` or `.node` input file then nodes on the boundary remain stationary regardless of `stat`.

- When subdividing, if a new node is placed between two original boundary nodes then that new node becomes a boundary node.
- When subdividing, if a new node is placed between two original non-dynamic nodes then that new node becomes non-dynamic.

### 5.1.22 surface\_normals

Orders the node indices in the facet definitions for a 2D polygon/polyline or 3D triangulated surface such that the normals for each facet (2D line-element or 3D triangle) follow some specified rules.

Command line usage:

```
surface_normals meshroot outroot rec dir icell
surface_normals meshroot outroot rec dir icell igroup
surface_normals meshroot outroot rec dir icell igroup vx [vy] vz
```

Command line parameters:

- Files `meshroot.node`, `meshroot.ele` and `meshroot.neigh` define the input 2D polygon/polyline or 3D triangulated surface. If the `.neigh` file does not exist then cell neighbour information is calculated automatically, which may increase the run-time significantly for large meshes.
- Set `rec` to "t" (true) or "f" (false) depending on whether you want to use a recursive method or a marching method to perform the propagation. The recursive method is memory-heavy and may give you a segmentation fault indicating you have run out of memory. In that case, use the marching method, which is memory-light but may be slower.
- If `dir`  $\neq 0$  then the surface is assumed to define a closed shape and parameters `vx`, `[vy]`, `vz` are not used.
  - If `icell`  $\geq 1$  then normals are propagated from the cell (surface facet) with that index and `dir` is ignored.
  - If `icell`  $\leq 0$  then the initial cell is chosen using an automated procedure that assumes a closed shape. In this case, set `dir`  $< 0$  for inward normals or `dir`  $> 0$  for outward normals as defined by a right-hand-rule moving along/around the facet nodes. For 3D, if the fingers curve around a triangular facet following the node ordering then the thumb indicates the vector direction. For 2D, if the fingers point along a line-element facet following the node ordering and if the fingers curve towards the  $+y$  direction then the thumb indicates the vector direction. This approach may not work for complicated bodies.
- If `dir`  $= 0$  then direction information is determined from the specified vector (`vx`, `[vy]`, `vz`). Leave the `vy` value out for 2D problems. The coordinate system for the specified vector should be consistent with that in file `meshroot.node`.
  - If `icell`  $\geq 1$  then normals are propagated from that cell. If the vector (`vx`, `[vy]`, `vz`) is supplied then the cell's normal vector is flipped, if required, before propagation such that it has a positive dot product with the specified vector. If the vector is absent from the command line call then the cell's normal is not flipped before propagation.
  - If `icell`  $\leq 0$  then a simple loop is performed over each cell in the surface (`rec` is ignored) and the cells' normals are flipped if required so they have a positive dot product with the specified vector. If `icell`  $\leq 0$  then the vector must be specified.
  - If `igroup`  $\geq 1$  then it specifies a facet group number. During propagation or looping, only cells in this group are altered. This feature is helpful when working with a multi-surface mesh because `surface_normals` will not propagate normals past edges that are connected to more than two facets, e.g. triple points.

Outputs:

- File `outroot.node` is written if `outroot` does not equal `meshroot` (not overwriting the input node file) or unused nodes were found in the input file (those nodes are removed in the output file).
- File `outroot.ele` contains the altered facet definitions. Only the ordering of the node indices for each facet is changed.
- File `outroot.neigh` contains the altered neighbour information. Only the ordering of the neighbouring indices for each facet is changed. This file is always written, regardless of whether input file `meshroot.neigh` was present.

Notes:

- Unused nodes are removed from the surface.
- To visualize the normal vectors, see program [mesh2vtu](#).

### 5.1.23 threshold\_attribute

Thresholds a tetrahedral mesh by cell attribute and extracts the bounding triangulated surface for the resulting body.

Command line usages:

```
threshold_attribute meshroot ai v1 v2 outroot
threshold_attribute meshroot ai v1 v2 outroot boundaryflag
threshold_attribute noderoot eleroot ai v1 v2 outroot
threshold_attribute noderoot eleroot ai v1 v2 outroot boundaryflag
```

Command line parameters:

- In the first two command line usages above, files `meshroot.node/.ele/.neigh` define a tetrahedral mesh. If file `meshroot.neigh` exists then it is used. Otherwise, the neighbour information is calculated by the program (a `.neigh` file is not required).
- In the latter two command line usages above, files `noderoot.node` and `eleroot.ele` define a tetrahedral mesh. If file `eleroot.neigh` exists then it is used. Otherwise, if file `noderoot.neigh` exists then it is used. Otherwise, the neighbour information is calculated by the program (a `.neigh` file is not required).
- The thresholding keeps only those cells with attribute number `ai` on `[v1,v2]`.
- The resulting volumetric body is written to files `outroot_mesh.node/.ele`.
- The bounding surface of the resulting body is written to files `outroot_surf.node/.ele/.poly`.
- Use `ai = 0` to use the entire (un-thresholded) model. The values for `v1` and `v2` are ignored in this case (use any dummy values you like).
- Use `ai < 0` to extract the topography surface. The program assumes that the boundary of the model volume is a rectangular prism oriented with the Cartesian axes. The bounding triangulated surface for the entire (un-thresholded) model is extracted and any facets with all nodes on the bottom and sides of the model volume are removed. The values for `v1` and `v2` are ignored in this case (use any dummy values you like).
- If `boundaryflag` is `t` (true) then the boundary of the modelling region is also written to `outroot.poly`. This can be helpful, for example, when you want to generate a `.poly` file for meshing with TetGen but only want to include a particular rock unit.

Notes:

- If you want mesh files (`.node/.ele`) containing the bounding triangulated surface after thresholding then you will have to run program `threshold_attribute` followed by program `poly2mesh`.
- The intention is that you will use TetGen to create a model on an unstructured mesh, then use this program to extract particular cell faces. For example, you may need to extract particular faces before running program `make_face_weights`.

### 5.1.24 threshold\_edge\_length

Thresholds a tetrahedral mesh by cell edge length and extracts the bounding triangulated surface for the resulting body.

Command line usage:

```
threshold_edge_length noderoot eleroot surfroot [d]
```

Command line parameters:

- Files `noderoot.node`, `eleroot.ele` and `eleroot.neigh` define a mesh. If the `.neigh` file does not exist then cell neighbour information is calculated automatically, which may increase the run-time significantly for large meshes.
- All cells with edge dimension(s) larger than `d` are removed (optional).
- The surface of the resulting body is then determined and those triangular cells written to files `surfroot.node/.ele`.

The intention is that this program will be part of the following procedure:

1. Use Triangle or TetGen to create a Delaunay meshing of some surface vertices as follows:

```
triangle -Den meshroot.node
```

or

```
tetgen -fn meshroot.node
```

which will generate files `meshroot.1.node/.ele/.face/.neigh`.

2. Rename/copy the `.face` file to a `.ele` file and add the missing nodes-per-cell parameter to the first line of that file. This new `.ele` file now defines a mesh of triangular facets.
3. Use program `cell_dimensions` on that new `.ele` file.
4. Use program `mesh2vtu` to create a `.vtu` file of the resulting information.
5. Load that `.vtu` into ParaView to determine an appropriate threshold value `d`.
6. Finally, run `threshold_edge_length` with the original tetrahedral mesh files and the value of `d` determine above.

### 5.1.25 toposplit

Reads files specifying a mesh (rectilinear or unstructured) and wireframe topography (unstructured), and writes a model file specifying the cells above and below the topography surface. Can also work with just a set of points and be used to indicate which are above or below a topography surface.

Command line usages:

```
toposplit noderoot eleroot meshfile modelfile outfile
toposplit noderoot eleroot meshfile modelfile outfile mode
toposplit noderoot eleroot meshfile modelfile outfile mode pow
toposplit noderoot eleroot meshfile modelfile outfile mode pow nl
```

Command line parameters:

- Topography data is read from files `noderoot.node` and `eleroot.ele`.
  - If `eleroot` specifies an existing file then linear interpolation is used across the 3D triangulated surface facets in files `noderoot.node` and `eleroot.ele`.
  - If `eleroot` is `null` then inverse-distance-weighting (IDW) is used for the interpolation.
- Files `meshfile` and `modelfile` define the model or set of points.
  - If `meshfile` has extension `.node` and `modelfile` is `"null"` then `meshfile` holds a set of points. Otherwise, a mesh is assumed (more below).
  - If `modelfile` has extension `.ele` then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
  - If the mesh is unstructured then `meshfile` should be a `.node` file and `modelfile` a `.ele` file.
  - If the mesh is rectilinear then those files should be UBC-GIF format. The `modelfile` does not need to already exist in this case and can then be specified as `"null"`.
- Parameter `mode` specifies the location in the cells to compare to the topography (ignored for a set of points):
  - use `"c"` for cell centroids
  - use `"t"` for the lateral centres of the tops of the cells (rectilinear meshes only)
  - use `"b"` for the lateral centres of the bottoms of the cells (rectilinear meshes only)
- The `pow` parameter specifies the interpolation power for IDW:
  - The default value is the number of dimensions.
  - `pow`  $\ll$  `ndim` (with `ndim` the number of dimensions) cause the interpolated values to be dominated by points far away.
  - `pow`  $\gg$  `ndim` causes the interpolated values to approach a piece-wise character (nearest neighbour interpolation, like a Voronoi diagram).
- `nl` is a looping parameter used when `eleroot` is not `null` (for developers: this looping parameter can be found inside subroutine `ugrid_find_cells` in module `ugrid_cls`). You may need to increase this parameter above 1 to obtain good results, but higher values can mean much longer run times. For more explanation, see the information for program `interpolate_mesh`.

Output:

- If a set of points is used then a `.node` format `outfile` is written that contains any existing information in `meshfile` plus a new node attribute named “toposplit” with values of 1 indicating nodes below topography and 0 above.

- If the input mesh is unstructured then a `.ele` format `outfile` is written that contains any existing information in `modelfile` plus a new cell attribute named “toposplit” with values of 1 indicating cells below topography and 0 above.
- If the input mesh is rectilinear then a UBC-GIF format `outfile` is written with values of 1 indicating cells below topography and 0 above.

Notes:

- To determine if a cell is above or below the topography surface, the cell centroid (or lateral centre of the top/bottom of the cell) is first calculated and the elevation of that cell location compared to the topography elevation at that same lateral location.
- Any cells marked with a value of 0 are at or above the topography surface (as defined by the `mode` parameter).
- Any cells marked with a value of 1 are below the topography surface.
- For a mesh, program `rockunits2ele` can be used to generate bounds using the output of program `toposplit`.
- For a set of points, program `remove_nodes` can be used to remove nodes above or below topography using the output of program `toposplit`.

## 5.2 Model visualization

See also programs `ele2vtu`, `mesh2vtu`, `poly2vtu` and `ubcgif2vtr`.

### 5.2.1 make\_scatter\_plot

Creates a scatter-plot (cross-plot) from two `.ele` files.

Command line usage:

```
make_scatter_plot cellsroot1 cellsroot2 [outroot]
```

Command line parameters:

- The second cell attributes in files `cellsroot1.ele` and `cellsroot2.ele` are written to the output files.
- If `outroot` is present then the output files are named `outroot_scatter_plot.vtu` and `outroot_scatter_plot.txt`.
- If `outroot` is absent then the output files are named `scatter_plot.vtu` and `scatter_plot.txt`.

Outputs:

- The `.vtu` file is for loading into ParaView and will plot the data as points.
- The `.txt` file contains two values on each line: the first column of values comes from file `cellsroot1.ele` and the second from file `cellsroot2.ele`.

### 5.2.2 outlines2poly

Generates a 2D poly file containing several outlines specified in an input file.

Command line usages:

```
outlines2poly infile
outlines2poly infile zrev
outlines2poly infile zrev outroot
```

Command line parameters:

- Information about the outlines is specified in file `infile`. The format is

```
n nout
a1 v1 outlinefile1
a2 v2 outlinefile2
...
an vn outlinefilen
```

where



- `n` is the number of outline files
- `nout` is the number of outline files that define the boundary (the rest define internal bodies)
- the `a` values are attribute values
- the `v` values are maximum volumes
- the `outlinefiles` are file names to read. Those files should contain two values on each line (`x` and `y` values).
- If `outroot` is present then the outline information is written to files `outroot.node/.ele/.poly`.
- If `outroot` is absent then `outroot` is set to the root of `infile`, assumed to be of the format `inroot.*`. Any path information in `infile` is kept, meaning that the output files will be written in the same directory as the input file.
- If `n` is greater than `nout` then some additional files are written:
  - If `nout` is greater than zero, `outroot_boundary.vtu` holds the boundary outline information.
  - Files `outroot_internal.node/.ele/.vtu` hold the outline information for internal bodies.
  - For the `.vtu` file(s), set the optional `zrev` parameter to `t` (true) if you want to switch coordinate systems such that the `x`- and `y`- directions are swapped and the `z`-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the `z` coordinates is switched.

## 5.3 Model query and calculations

### 5.3.1 bulk\_magnetization

Calculates the bulk magnetization in a model with option to mask the calculation.

Command line usages:

```
bulk_magnetization meshfile modelfile
bulk_magnetization meshfile modelfile maskfile ia a1 a2
```

Command line parameters:

- Files `meshfile` and `modelfile` define the mesh and magnetization model (`.node/.ele` files or UBC-GIF format files).
- Files `meshfile` and `maskfile` define the model to use for masking (`.node/.ele` files or UBC-GIF format files).
- If `maskfile` is present then `ia` defines the index of the attribute in that model to use for the masking. The calculation is only performed for mesh cells with that attribute between the inclusive lower and upper bounds `a1` and `a2`.

Outputs:

- Information on the bulk magnetization is printed to screen.

### 5.3.2 calculate\_crossgrad

Calculates the cross-gradient measure.

Command line usages:

```
calculate_crossgrad meshfile modelfile1 modelfile2 [ai1 ai2 [weight]] [outroot]
```

Command line parameters:

- Files `meshfile`, `modelfile1` and `modelfile2` define the mesh and two models to use (`.node/.ele` files or UBC-GIF format files).
- If using an unstructured mesh, and `meshfile` is of the form `meshroot.node`, then if file `meshroot.neigh` exists then cell neighbour information is read from that file. Failing that, cell neighbour information is calculated, which may lead to longer run-times for this program.
- Integer parameters `ai1` and `ai2` specify the indices of the attributes to be taken from each input model file for the calculation. Default values are 1.
- Real parameter `weight` multiplies the cross-gradient constraint vector. Default value is one. If this parameter is provided then `ai1` and `ai2` must also be provided.

- If parameter **outroot** is present then a new model file with root **outroot** is written, with the extension determined automatically, and with the following attributes:
  - “Model1” is the first model.
  - “Model2” is the second model.
  - “CrossGrad” is the cross-gradient measure value for each cell.
  - “Partial1” is the partial derivative of the cross-gradient measure with respect to the first model.
  - “Partial2” is the partial derivative of the cross-gradient measure with respect to the second model.
  - “CrossGradNormalized” is the cross-gradient measure value for each cell but normalized by the maximum value (this attribute is not included if the maximum value is zero).
  - “Partial1Normalized” is the partial derivative of the cross-gradient measure with respect to the first model, normalized by the maximum value of the cross-gradient measure for each cell (this attribute is not included if that maximum value is zero).
  - “Partial2Normalized” is the partial derivative of the cross-gradient measure with respect to the second model, normalized by the maximum value of the cross-gradient measure for each cell (this attribute is not included if that maximum value is zero).
  - “CrossGradWeighted” is the cross-gradient measure value for each cell but weighted by parameter **weight** (this attribute is not included if **weight** is one).

Notes:

- The cross-gradient measure is printed to screen.
- Multiple sets of gradient operators are used such that asymmetrical results are avoided.

### 5.3.3 calculate\_fcm

Calculates the fuzzy c-means (FCM) measure.

Command line usages:

```
calculate_fcm meshtype model1 model2 jointinp
calculate_fcm meshtype model1 model2 jointinp outfile
```

Command line parameters:

- **meshtype** should be **r** for rectilinear or **u** for unstructured.
- If **meshtype** is **r** then **model1** should be a UBC-GIF format model file.
- If **meshtype** is **u** then **model1** should be a **.ele** format file.
- If **meshtype** is **r** and **model2** is "null" then data values are taken from the 1st and 2nd attribute columns in file **model1**.
- If **meshtype** is **u** and **model2** is "null" then data values are taken from the 1st and 3rd attribute columns in file **model1**.
- If **model2** is not "null" then data values are taken from the 1st attribute columns in files **model1** and **model2**.
- Cluster centre locations and other information is read from file **jointinp**, which should be a joint coupling input file (see [Section 2.5](#)) specifying the **fcm** option for the **coupling** parameter.
- If **outfile** is present then a new file named **outfile** is written with the following attributes:
  - the two attributes used for the calculation, taken from the model input file(s)
  - “FCM” contains the FCM measure value for each cell
  - “BestCluster” contains cluster centre indices of highest membership (closest cluster centre)
  - “BestMembership” contains the membership value for that best cluster centre.

That information helps to connect groups of points on a physical property scatter plot to cells in the inversion mesh.

Notes:

- The FCM measure and XBI validity measure are printed to screen.

### 5.3.4 calculate\_gradients

Calculates spatial model gradients.

Command line usage:

```
calculate_gradients meshfile modelfile ai outroot
```

Command line parameters:

- Files **meshfile** and **modelfile** define the mesh and two models to use (**.node/.ele** files or UBC-GIF format files).
- If using an unstructured mesh, and **meshfile** is of the form **meshroot.node**, then if file **meshroot.neigh** exists then cell neighbour information is read from that file. Failing that, if **modelfile** is of the form **modelroot.node**, then if file **modelroot.neigh** exists then cell neighbour information is read from that file. Failing both of those options, cell neighbour information is calculated, which may lead to longer run-times for this program.
- Integer parameter **ai** specifies the attribute index to use in the **modelfile**.
- If parameter **outroot** is present then a new model file with root **outroot** is written, with the extension determined automatically, and with the following attributes:
  - **ma** is the model attribute taken from the **modelfile**
  - **gx**, **gy** and **gz** are the Cartesian components of the spatial model gradient in each mesh cell (**gy** is omitted for 2D problems).
  - **dec** and **inc** are the declination and inclination angles in degrees for the gradient vectors (**dec** is omitted for 2D problems).

Notes:

- Multiple sets of gradient operators are used such that asymmetrical results are avoided.

### 5.3.5 calculate\_tvar

Calculates the total variation (total gradient) for a model on a rectilinear or unstructured mesh.

Command line usage:

```
calculate_tvar meshfile modelfile ai outfile
```

Command line parameters:

- Files **meshfile** and **modelfile** define the model.
- If the mesh is unstructured then **meshfile** should be a **.node** file and **modelfile** a **.ele** file.
- If the mesh is rectilinear then those files should be UBC-GIF format.
- If **modelfile** has extension **.ele** then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- Parameter **ai** specifies the cell attribute (model) to use.
- The total variation is  $\sqrt{(\nabla m_1)^2 + (\nabla m_2)^2}$  where  $m_1$  and  $m_2$  are the two models. The result is normalized by the highest value and written to file **outfile** (the same file format as the input **modelfile**).

Notes:

- For a rectilinear mesh, the squared model gradients located on cell faces are interpolated to cell centres before summation.
- For an unstructured mesh, gradient operators are used that define model gradients at cell centres, so no such interpolation is required.

This program is used for a re-weighted inversion following the approach of [Lelièvre \(2009\)](#); [Lelièvre et al. \(2009\)](#) in concert with programs:

- [smooth\\_model](#) (used to smooth the total variation)
- [sum\\_models](#) (used to sum total variations from multiple models)
- [mesh2vtu](#) or [ubcgif2vtr](#) (used to create a file for viewing the total variation in ParaView)
- ParaView (used to determine an appropriate threshold on the total variation)
- [scoop\\_model](#) (used to create the cell-centred weighting file based on that threshold)

- `make_face_weights` (used to create face-centred weights based on the cell-centred weights)

### 5.3.6 cell\_dimensions

Calculates cell dimension information for an unstructured mesh.

Command line usage:

```
cell_dimensions fileroot [outroot [zrev [step]]]
```

Command line parameters:

- Files `fileroot.node/.ele/.neigh` define an unstructured mesh.
- If `outroot` is present then the output file is named `outroot.vtu`, otherwise `fileroot.vtu`.
- If `outroot` is present and different from `fileroot` then output file `outroot.ele` is also written. Hence, file `fileroot.ele` is never overwritten.
- For the `.vtu` file, set the optional `zrev` parameter to `t` (true) if you want to switch coordinate systems such that the x- and y- directions are swapped and the z-direction is reversed (from one right-handed Cartesian coordinate system to another). For a 2D scenario, the sign of the z coordinates is switched.
- If `step` is provided then some histogram information on obtuse angles is printed to the screen.

The following cell attributes are calculated and written to the output file(s):

- cell centroid x
- cell centroid y (for 3D only)
- cell centroid z
- maximum cell angle
- minimum cell angle
- cell volumes
- largest linear element (e.g. edge) length
- minimum angles encountered during gradient calculation for rotated gradient
- collinear cell centroids (values are 0 or 1)

### 5.3.7 check\_bounds

Checks if a model is on bounds (checks if the bounds are “active”) and calculates the distance of the model to the bounds in physical property space.

Command line usage:

```
check_bounds modelroot im boundsroot ib1 ib2 outroot
```

Command line parameters:

- The physical property model is read from file `modelroot.ele`.
- Parameter `im` specifies the attribute index in file `modelroot.ele` to use for the model.
- The physical property bounds are read from file `boundsroot.ele`.
- Parameters `ib1` and `ib2` specify the attribute indices in file `boundsroot.ele` to use for the lower and upper bounds respectively.
- The output file is named `outroot.ele`. It contains six attributes:
  1. the model
  2. “LowerBound” is the lower bound
  3. “UpperBound” is the upper bound
  4. “DistanceToLowerBound” is the distance between the model and the lower bound (model minus lower bound)
  5. “DistanceToUpperBound” is the distance between the model and the upper bound (upper bound minus model)

6. “ActiveBounds” holds integers that indicate whether the bounds are active for each cell in the model:
- 0 = no bounds are active
  - 1 = lower bound is active
  - 2 = upper bound is active
  - 3 = the lower and upper bounds are equal

### 5.3.8 closest\_nodes

Finds the indices of a set of nodes closest to another set of nodes.

Command line usage:

```
closest_nodes noderoot1 noderoot2 outfile
```

Command line parameters:

- Nodes are read from files **noderoot1.node** and **noderoot2.node**.
- For each node in **noderoot1.node**, the program finds the closest node in **noderoot2.node** and provides the index of that node in **noderoot2.node**.
- The output file is named **outfile**.

Outputs:

- File **outfile** has three columns. Each row corresponds to a different node in file **noderoot1.node**.
- The first column contains line numbers for the nodes in file **noderoot1.node**. These are line indices, counting from 1 up, not the original IDs in file **noderoot1.node**.
- The second column contains node indices along those lines, again counting from 1 up, and returning to 1 when a new line is encountered.
- The third column contains the indices of the closest nodes found in **noderoot2.node**.

Notes:

- If there are no line numbers indicated in file **noderoot1.node** (stored in the boundary marker column) then a default values of 1 is used.
- This program assumes that the nodes in file **noderoot1.node** are grouped by line number.

### 5.3.9 check\_coll

Checks a surface mesh for any intersections of its facets.

Command line usage:

```
check_coll meshinp [coll_method]
```

Command line parameters:

- **meshinp**
  - A model discretization specification file.
  - See the documentation on [model discretization specification files](#) for more information on the file format.
- **coll\_method**: an integer representing the algorithm wanted to be used in the intersection detection. 1 - Moller’s algorithm; 2 - Guigue and Deviller’s algorithm; 3 - Sabharwal and Leopold’s algorithm. Default: 1.

Outputs:

- The output is a display listing the the facets who are responsible for the intersection. Facets are represented by their ID numbers.

### 5.3.10 dyno\_difference

Calculates and displays statistcial information between a surface model inverted with DyNo, and its corresponding true model. Used in comparing synthetic modelling results.

Command line usage:

```
dyno_difference noderoot1 noderoot2
```

Command line parameters:

- **noderoot1** and **noderoot2** and the file roots for the node files, where either can be the inverted model or true model, the order doesn't matter.

The statistical informatin that is displayed by the program is:

- **Eucl. dif. mean**: the mean Euclidean distance between every node in the inversion model and its corresponding node in the true model.
- **Eucl. dif. standard deviation**: the standard deviation of **Eucl. dif. mean**.
- **Eucl. dif. min**: the minimum Euclidean distance between two nodes in the compared models.
- **Eucl. dif. max**: the maximum Euclidean distance between two nodes in the compared models.
- **n**: the number of nodes in each model.
- **MISFIT**: the L2 norm of the Euclidean distance between each corresponding node pairs in the models.

### 5.3.11 find\_groups

Reads unstructured mesh files and assigns new “CellGroup” attributes to each physically connected group of cells (cells that neighbour each other across a shared face).

Command line usages:

```
find_groups meshroot
find_groups meshroot outroot
find_groups noderoot eleroot outroot
```

Command line parameters:

- Reads unstructured mesh files **meshroot.node/.ele** and **meshroot.neigh** if the latter exists.
- Writes unstructured mesh files **outroot.node/.ele/.neigh/.vtu** with new cell attribute named “CellGroup” indicating each physically connected group of cells, each of which will have a different integer value for the “CellGroup” cell attribute.

### 5.3.12 find\_holes

Finds holes in a 3D surface of polygonal facets, i.e. tests for a watertight model.

Command line usages:

```
find_holes polyroot
find_holes noderoot eleroot
```

Command line parameters:

- Depending on the command line usage, a 3D surface of polygonal facets is defined by file **polyroot.poly** or files **noderoot.node** and **eleroot.ele**.

Notes:

- The approach taken is to find edge elements in the surface that belong to only a single facet. The indices of the nodes belonging to those edge elements are output to the screen.

### 5.3.13 model\_difference

Provides statistics on the difference between two UBC-GIF format models or any two column-based data files with the same number of values on every line. Program [data\\_difference](#) can be used when the models are **.ele** files. For magnetic vector inversion results, you may need program [mvi\\_difference](#).

Command line usages:

```
model_difference file1 file2
model_difference file1 file2 ai1 ai2
```

Command line parameters:

- Files `file1` and `file2` should be column-based data files with the same number of values on every line.
- If `ai1` and `ai2` are present then the only difference calculated is that between attribute (data column) `ai1` in the first data file and attribute `ai2` in the second.
- If `ai1` and `ai2` are absent then differences are calculated for all shared attribute columns (see notes below).

Outputs:

- Calculates the difference between the data values in the input files (`file2 - file1`) and prints some statistics to the screen.

Notes:

- This program will work with the UBC-GIF model format: either the standard single model format or my multiple column extension without header.
- A error is thrown if the files do not contain the same number of rows or columns.

### 5.3.14 mvi\_difference

Calculates the difference between two magnetic vector inversion (MVI) recovered models. Currently only supports results on 3D rectilinear meshes.

Command line usage:

```
mvi_difference file1 file2 outfile
```

Command line parameters:

- Files `file1` and `file2` should be the recovered model files.
- Output file `outfile` has the difference between the two models (second subtract first).

### 5.3.15 pun2nun

Creates two surface meshes that represent  $\pm 1$  standard deviation of the nodes' positions, the uncertainty resulting from an uncertainty in the model's physical properties. Used on the resulting inversion model from DyNo. Only set for the magnetic susceptibility physical property, not yet know if functional for others.

Command line usage:

```
pun2nun meshinp outward
```

Command line parameters:

- `meshinp`
  - A model discretization specification file.
  - See the documentation on [model discretization specification files](#) for more information on the file format.
- `outward`: a logical, `t` is the normals of the surface mesh are directed outwards, `f` if they are directed inwards.

Outputs:

The output is six files:

```
meshroot_plus.node
meshroot_minus.node
meshroot_plus.ele
meshroot_minus.ele
meshroot_plus.vtu
meshroot_minus.vtu
```

Notes:

- Although the `meshfile` from the `meshinputfile` is unaltered, and outputting the `_plus.ele` and `_minus.ele` files seems redundant, the program `pun2nun` is presently being used to develop an uncertainty in volumes for the sections in the surface model. This means it is being used shortly before `surf_volume`, which required Tetgen to be used, which requires the input file to be a `.poly`, which is easier to put together if the `.node` and `.ele` have the same root name.
- The function used to relate the physical property uncertainty to node position uncertainty is (with `prop` being the percent uncertainty of the physical property):

$$\begin{aligned}
 - \quad \text{positive standard deviation} &= (2.33E-7)prop^4 - (4.9E-5)prop^3 + (3.06E-3)prop^2 + (1.1E-2)prop + 2 \\
 - \quad \text{negative standard deviation} &= (3.07E-10)prop^6 + (5.39E-8)prop^5 + (2.06E-6)prop^4 - (2.02E-5)prop^3 + \\
 &\quad (2.32E-3)prop^2 - (2.17E-3)prop + 2
 \end{aligned}$$

These functions are normalized by the maximum of a node's position constraint.

- The uncertainty values for the physical property are set as the second attribute in the rock-units file, see [Rock-units files](#), as `Uncert`. Example:

```

1 2
1 0 0 0 0.1 5
MagSus
Uncert

```

For a volume with magnetic susceptibility 0.1 [SI], which has an uncertainty of 5%.

### 5.3.16 query\_model

Interpolates model values in a mesh (a voxel model) at specified points. Does essentially the opposite of program `populate_model`.

Command line usage:

```
query_model zdir1 meshfile1 modelfile1 zdir2 nodefile2 oob [outfile]
```

Command line parameters:

- Files `meshfile1` and `modelfile1` define the input voxel model.
- File `nodefile2.node` defines the point locations at which the model values are interpolated.
- If the mesh is unstructured then `meshfile1` should be a `.node` file and `modelfile1` a `.ele` file.
- If the mesh is rectilinear then those files should be UBC-GIF format.
- If `modelfile1` has extension `.ele` then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- If a node is not within the bounds of the mesh then a value of `oob` (out-of-bounds) is used for all attributes for that node.
- `zdir1` and `zdir2` specify the coordinate systems for the input information.  
For 3D, set `zdir > 0` to specify  $+z$  up,  $+x$  East,  $+y$  North;  
set `zdir < 0` to specify  $+z$  down,  $+x$  North,  $+y$  East.  
For 2D, the  $+x$  axis is always to the right and `zdir` only specifies the  $+z$  direction. The  $y$  axis is along-strike.
- If `outfile` is present then it indicates the output file name. If absent, file `nodefile2` is overwritten.

Notes:

- All cell attributes (models) in the mesh are interpolated into the node attributes and written to the `outfile`.

### 5.3.17 sum\_models

Sums two models.

Command line usage:

```
sum_models modelfile1 modelfile2 outfile [normalize]
```

Command line parameters:

- `modelfile1` and `modelfile2` can be `.ele` or UBC-GIF format model files (the latter is assumed for all file extensions other than `.ele`).
- The first cell attribute (model) in the files are summed and the result is written to file `outfile`.



- If `normalize` is present and set to `t` (true) then each model is normalized by its maximum value before summation and the result is divided by 2. Hence, if each model lies on  $[0, \infty]$  then the result lies on  $[0, 1]$ .

Notes:

- An error is thrown if the number of cells differs.

### 5.3.18 surf\_volume

Calculates the volume of each closed region of a surface mesh model.

Command line usage:

```
surf_volume meshroot
```

Command line parameters:

- `meshroot`: the root of the `.node` and `.ele` files which define the surface mesh you're working with.

Outputs:

The output display for a mesh with N regions is:

```
Volume 1 is:  [number] m³
Volume 2 is:  [number] m³
...
Volume N is:  [number] m³
```

Notes:

- To run this program the input `.ele` file must have four components, i.e. represent a mesh of tetradera.
- The way I've found works best to use `surf_volume` is to:
  1. Run `mesh2poly` on your surface mesh files (`meshroot.node` and `meshroot.ele`), getting `meshroot.poly`;
  2. open `meshroot.poly`, and manually add all the closed regions of the surface mesh (see [http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual006.html#ff\\_poly](http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual006.html#ff_poly), **Part 4 -region attribute lits**). Have the attribute to each region be the same as the region number:
 

```
1 <x1> <y1> <z1> 1 1
2 <x2> <y2> <z2> 2 2
...
```
  3. Run `tetgen` with the edited `meshroot.poly` files, using the `-pAFC` flags;
  4. then with the resulting `meshroot.1.node` and `meshroot.1.ele` files, run `surf_volume` as:

```
surf_volume meshroot.1
```

### 5.3.19 tetgen\_intersections

Takes the output of TetGen with the `-d` flag and writes a `.poly` file containing intersecting facets.

Command line usage:

```
tetgen_intersections polyroot txtfile
```

Command line parameters:

- File `polyroot.poly` is passed to TetGen.
- File `txtfile` contains the output of TetGen with the `-d` flag.

The first step is to run TetGen with the `-d` flag and pipe the output to a text file, e.g.:

```
tetgen -d orig.poly > out.txt
```

The second step is to run this program with that text file, e.g.:

```
tetgen_intersections orig.poly out.txt
```

The result for that example will be a file called `out.poly` (named similarly to `txtfile` but with altered extension) that contains only the information for the intersecting facets. The output file is placed in the current directory. You can then run program `poly2vtu` and load the `.vtu` file into Paraview to help you visualize the problematic facets.

## 5.4 Creation of inversion input files

### 5.4.1 `check_dyno_bounds`

Studies a surface mesh with given node constraints from a `nodeunitsfile`, and displays the pairs of facet groups that could intersect during an inversion. This is done by checking for the intersection of each nodes' bounding boxes (a box created about each node whose dimensions are defined by that node's Cartesian constraints). Used in the writing of a `collfile`.

Command line usage:

```
check_dyno_bounds meshinp
```

Command line parameters:

- `meshinp`
  - A model discretization specification file.
  - See the documentation on [model discretization specification files](#) for more information on the file format.

Outputs:

- The output is a display listing the pairs of facet groups which could intersect.

### 5.4.2 `make_face_weights`

Sets face-based weights based on the values in an unstructured or rectilinear model.

Command line usage:

```
make_face_weights mode meshfile modelfile ai v w outroot
```

Command line parameters:

- Files `meshfile` and `modelfile` define the model.
- If `modelfile` has extension `.ele` then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- If the mesh is unstructured then `meshfile` should be a `.node` file and `modelfile` a `.ele` file.
- If the mesh is rectilinear then those files should be UBC-GIF format mesh and model files.
- The following `mode` strings are accepted:
  - `"diff"` – The program finds faces across which there are cells with different values of attribute number `ai`.
  - `"one"` – The program finds faces across which only one of the cells has attribute number `ai` equal to `v`.
  - `"two"` – The program finds faces across which both of the cells have attribute number `ai` equal to `v`.
  - `"either"` – The program finds faces across which either of the cells (or both) have attribute number `ai` equal to `v`.
- Weights are set to `w` on those faces and `1.0` on all others.
- Parameter `outroot` specifies the root name for several output files (see below).

Outputs:

- All the output files described below are written regardless of the type of mesh, rectilinear or unstructured.
- Files `outroot_faces.node/.ele` hold all the faces in the mesh and contains an attribute named "weights".
- Files `outroot_contacts.node/.ele` hold the faces in the contacts.
- File `outroot_weights.txt` holds the face-based weights, e.g. for use as an input into an inversion program such as VIDI or the UBC-GIF programs MAG3D and GRAV3D.
- The `outroot_faces.*` and `outroot_contacts.*` files are in unstructured mesh format, even if the input is a rectilinear model.

Notes:

- All modes are helpful if you have a model on an unstructured mesh, e.g. generated using TetGen, and you wish to set weights adjacent to particular rock unit contacts. Program [rockunits2ele](#) can be of help here.

### 5.4.3 make\_cell\_weights

Sets cell-based weights based on the values in an unstructured or rectilinear model.

Command line usages:

```
make_cell_weights meshfile modelfile ai w outroot
make_cell_weights meshfile modelfile ai v w outroot
```

Command line parameters:

- Files `meshfile` and `modelfile` define the model.
- If `modelfile` has extension `.ele` then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- If the mesh is unstructured then `meshfile` should be a `.node` file and `modelfile` a `.ele` file.
- If the mesh is rectilinear then those files should be UBC-GIF format mesh and model files.
- For the first command line usage (`v` not supplied), the program finds faces across which there are cells with different values of attribute number `ai`. Weights are set to `w` in those cells and 1.0 on all others.
- For the second command line usage (`v` supplied), the program finds faces across which only one of the cells has attribute number `ai` equal to `v`. Weights are set to `w` in those cells and 1.0 on all others.
- `outroot` specifies the root name for the output file (a UBC-GIF format model file or unstructured `.ele` file), which will contain a cell attribute named “weights” holding the calculated weights.

### 5.4.4 robust\_weights

Calculates the weights for the robust modelling iterative re-weighting scheme of [Günther & Rücker \(2006\)](#).

Command line usage:

```
robust_weights seninp invinp [sc]
```

Command line parameters:

- `seninp` is the input file for the `gzsen3d` program for the PREVIOUS inversion.
- `invinp` is the input file for the `gzinv3d` program for the NEXT inversion.
- `sc` (optional) can be used to rescale the weights:
  - Values of `sc` below 1.0 will lead to more sharpening of the model recovered in the next inversion.
  - Values of `sc` above 1.0 will lead to less sharpening of the model recovered in the next inversion.

Outputs:

- Writes a file containing the model smallness and smoothness weights required for the NEXT inversion. The name of that file is taken from the `invinp` file (see notes below).

Assumptions:

- Any values in the recovered model equal to or below -99 correspond to cells above the topography surface.

Notes:

- This program was designed for use with `grav3d` version 5.0, release date Sept. 2013, and may not be consistent with other versions.
- As you perform the iterative procedure, you should keep all parameters in the inversion input file constant other than the initial model file and the `wdat` model weights file.
- **The initial model file specified on the fifth line `invinp` file should equal the final recovered model from the PREVIOUS inversion.**
- The information in the sensitivity input file `seninp` should be consistent with the PREVIOUS inversion, i.e. that which generated the initial model file specified in the `invinp` file.

- The model weights file specified on the last line of the `invinp` file is the file generated by this program.
- I suggest you always set `alphas=0` and `alphax=alphay=alphaz` when performing this iterative re-weighting procedure. The program will provide a warning message if this is not the case.
- I also suggest you do not attempt this iterative re-weighting procedure with a reference model in the smoothness term of the model objective function (grav3d inversion parameter `SMOOTH_MOD_DIFF`) because the behaviour in such a situation has not been researched. The program will provide a warning message if you attempt this.
- It is possible to use a single inversion input file `invinp` throughout the iterative procedure provided you always rename the final inversion output file (`gzinv3d_###.den`, where `###` is the final iteration number) to that specified as the initial model file in `invinp`. However, keep in mind that the model weight files and recovered model files will be overwritten if you do this.

Testing history:

- This program has been tested using grav3d version 5.0 on a non-uniform mesh without topography.
- The program has yet to be tested using topography.
- The effect of the optional scaling factor `sc` has yet to be tested.

## 5.5 Interpolation

### 5.5.1 interpolate\_mesh

Interpolates values from one mesh to another. For rectilinear-to-rectilinear mesh interpolation you may want to use the more efficient program `interp_rect_mesh`, although see the assumptions for that program.

Command line usage:

```
interpolate_mesh method zdir1 meshfile1 modelfile1 zdir2 meshfile2 modelfile2 modelfileout oob [tol [nl]]
```

Command line parameters:

- There are two algorithms that differ in the way in which they search for overlapping cells (see algorithm description below). Set `method` to 1 or 2 to select from them.
  - Method 1 tends to be faster for good quality unstructured meshes (where `nl` set to 1 works fine).
  - Method 2 tends to be faster for poor quality unstructured meshes (where `nl` set to 1 works poorly).

Therefore, my suggestion is to first try setting both `method` and `nl` to 1 and assess the results to determine the best strategy.
- Files `meshfile1` and `modelfile1` define the mesh and model from which model values are interpolated. Files `meshfile2` and `modelfile2` define the mesh and model to which interpolated model values are assigned.
  - If a mesh is unstructured then the mesh file should be a `.node` file and the model file a `.ele` file. Parameter `modelfile2` must be an existing `.ele` file in this case.
  - If a mesh is rectilinear then the mesh and model files should be in UBC-GIF format. Parameter `modelfile2` does not need to be an existing file in this case and you can enter `null`.
  - If a model file has extension `.ele` then an unstructured mesh is assumed, otherwise a rectilinear mesh is assumed.
- `zdir1` and `zdir2` specify the coordinate systems for unstructured meshes. These parameters are ignored for rectilinear meshes but must be present in the command line usage.
 

For 3D, set `zdir> 0` to specify `+z` up, `+x` East, `+y` North;  
 set `zdir< 0` to specify `+z` down, `+x` North, `+y` East.

For 2D, the `+x` axis is always to the right and `zdir` only specifies the `+z` direction. The `y` axis is along-strike.
- All cell attributes (models) in the first mesh are interpolated into the second. If a cell in the second mesh is not within the bounds of the first then a value of `oob` (out-of-bounds) is used for that cell (for all attributes).
- The output model file is named `modelfileout`. Any attribute columns already existing in `modelfile2` will be copied into `modelfileout` but will be overwritten if they have the same names as any attributes being interpolated.

- `tol` is a tolerance used when checking if a point is inside an unstructured mesh cell. Set `tol` to a negative value to trigger the default, which is currently a very small non-zero value. In some cases, you may need to increase this tolerance to obtain appropriate results. If the tolerance is set too high then false point-in-cell detections may occur.
- `n1` is a fiddle factor used when searching for cells that overlap in the two meshes. Smaller values may result in poor interpolations where overlapping cells in the two meshes are not detected properly. Increase `n1` above 1 if you get poor results. Increasing the value of `n1` will increase the run time so do this slowly until you obtain appropriate results. Using a huge value of `n1` will perform the naive, exhaustive, slow double-loop search.

Algorithm:

- For each cell in the second mesh, the program:
  1. calculates the centroid
  2. finds a cell in the first mesh that contains that centroid location (which I'll refer to as  $P$  in the following discussion).
- When the first mesh is rectilinear, the latter operation is fast. When the first mesh is unstructured, the naive approach is to loop over every cell in the first mesh and determine if that cell contains  $P$ . This represents a double loop that is prohibitively expensive. Therefore, the algorithm proceeds in one of two ways:
  - For method 1:
    1. find the closest node in the first mesh to  $P$
    2. find all the cells in the first mesh that belong to that closest mesh node
    3. find all the cells in the first mesh that belong to any nodes in any of the previously-found cells, repeating at most `n1` times
    4. check if  $P$  is in or on any of the cells found above.
  - For method 2:
    1. find the cell in the first mesh with centroid closest to  $P$
    2. find all the cells in the first mesh that neighbour that closest mesh cell
    3. find all the cells in the first mesh that neighbour any of the previously-found cells, repeating at most `n1` times
    4. check if  $P$  is in or on any of the cells found above.

Notes:

- The percentage completed is printed to the screen.

### 5.5.2 `interp_rect_mesh`

Interpolates values from one rectilinear mesh to another. This program is more efficient than `interpolate_mesh` for interpolating between rectilinear meshes but pay attention to the assumptions indicated below.

This program is designed specifically to interpolate between rectilinear meshes designed for similar inverse problems. For example, one mesh may be a padded version of the other, or one may be a refinement of the other. When interpolating from a fine mesh to a coarse mesh, this program performs some volume averaging of the physical property model(s) on the mesh. In contrast, program `interpolate_mesh` performs no such averaging.

Command line usage:

```
interp_rect_mesh meshfile1 modelfile1 meshfile2 modelfile2 [oob]
```

Command line parameters:

- Files `meshfile1` and `modelfile1` define the mesh and model **from** which model values are interpolated.
- Files `meshfile2` and `modelfile2` define the mesh and model **to** which interpolated model values are assigned.
- Those mesh and model files should be UBC-GIF format.
- All cell attributes (models) in the first mesh are interpolated into the second.
- Optional parameter `oob` gives an out-of-bounds value.
  - If `oob` is absent, the mesh boundaries are assumed to be identical and an error is thrown if the interpolation can not be performed for a particular cell.

- If `oob` is present and entered as some numerical value, the mesh boundaries are not assumed to be identical, and a value of `oob` is assigned to a cell if the interpolation can not be performed for that cell.
- If `oob` is present and entered as `"F"` or `"f"` (false), the mesh boundaries are not assumed to be identical and any existing values in the model are used if the interpolation can not be performed for a particular cell. If there are no existing values in the “to” model, i.e. `modelfile2` does not exist before the program is run, then values of zero are used to populate the “to” model before interpolation.

#### Assumptions:

- This program assumes that the meshes are coarser versions of an underlying fine mesh, or that one or both are that mesh themselves. This has bearing on the way in which values are interpolated between meshes.
- The mesh with fewer cells is assumed to be the coarser of the two input meshes.
- If the interpolation is from coarse mesh to fine mesh then a simple find-enclosing-cell operation is used. The assumption is that the fine mesh cell is completely within the coarse mesh cell, possibly with their boundaries touching.
- If the interpolation is from fine mesh to coarse mesh then multiple values inside a coarse cell are volume-averaged. The assumption is that the fine mesh cells inside the coarse mesh cell are completely inside the coarse mesh cell, possibly with their boundaries touching.

# Chapter 6

## Miscellaneous utilities

### 6.1 fit\_mag\_dipole

Least-squares fitting of magnetic data by a sphere.

Command line usage:

```
fit_mag_dipole dataroot inc dec str [np]
```

Command line parameters:

- Reads magnetics data from file `dataroot.node`.
- `inc`, `dec` and `str` specify the inclination, declination and strength of the Earth's field (angles in degrees, strength in nT).
- `np` is the number of particles used in the particle swarm optimization (PSO) solver.

Outputs:

- Writes file `dataroot.best_fit_dipole.node` containing the response of the best-fit dipole.
- The dipole location, dipole moment, and data level are printed to the screen. That data level has been added to the observed data (the data in file `dataroot.node`).

Notes:

- The dipole moment direction is bounded 90 degrees away from the Earth's field orientations.
- My suggestion is to run the analysis several times to ensure repeatable results: multiple minima may occur and any one run may not yield the optimal solution.

Assumptions:

- The magnetic data is in the first attribute column in file `dataroot.node`.
- +z up coordinate system.

### 6.2 log2aux

Takes numbers from a DYN0 .log file and places them into a .aux file.

Command line usage:

```
log2aux fileroot
```

Command line parameters:

- The columns of numbers in file `fileroot.log` are placed into file `fileroot.aux`.
- If file `fileroot.aux` exists then it is overwritten.

### 6.3 rel2abs

Reads a .node file and converts relative node location bounds to absolute. This program is for use with the inversion program DYN0, which is not part of the PODIUM package.

Command line usages:

```
rel2abs 1 noderoot outroot dx dy dz
```

```
rel2abs 2 noderoot outroot centroidregions centroidunits ic
```

Command line parameters:

- Reads node information from file `noderoot.node`.
- Writes altered node information to file `outroot.node`.
- In Cartesian mode (the first command line usage above):
  - `dx`, `dy` and `dz` are the relative node location bound values.
  - Enter a dummy `dy` value for a 2D scenario.
  - The absolute bounds are calculated by adding and subtracting those relative bounds to/from the node coordinates.
- In spherical/polar mode (the second command line usage above):
  - Reads centroid information from file `centroidregions` (in regions file format).
  - Reads rock unit information from file `centroidunits` (in rockunits file format).
  - Attributes “RadiusLower” and “RadiusUpper” should exist in file `centroidunits`.
  - `ic` is the node attribute index that holds region ID values.

## 6.4 `remove_comments`

Removes any commented lines in a file.

Command line usage:

```
remove_comments cc inputfile outputfile
```

Command line parameters:

- `cc` defines the comment character, e.g. `#` or `!`.
- This program reads file `inputfile` and writes file `outputfile`. The latter will be identical to the former but any commented lines (those starting with the specified comment character) are removed. White space at the start of lines is ignored, meaning the comment character only has to be the first non-blank character on a line for it to be omitted in the output file.



# Chapter 7

## Change Log

The modifications indicated in the table below are only those that have changed the functionality of one or more programs mentioned in this document. Only major bug fixes are indicated, i.e. those that would significantly change the results of a program.

Date* yyyy/mm/dd	Revision*	Description of change
2016/01/25	2243	Added <code>usez</code> and <code>icol</code> command line parameters to program <code>remove_duplicates</code> .
"	"	Added program <code>decimate_by_mesh</code> .
"	"	Added <code>atts</code> command line parameter to program <code>interpolate.data</code> .
"	"	Added support for ESRI grid files in program <code>convert_format</code> .
"	"	Program <code>combine_node</code> now uses <code>mapflag="f"</code> (false) when removing duplicates.
"	"	Added <code>iat</code> command line parameter to program <code>remove_trend</code> .
"	"	Renamed program <code>interpolate_topography</code> to <code>interpolate.data</code> .
"	"	Added <code>iat</code> command line parameter to program <code>interpolate.data</code> .
"	"	Added <code>tol</code> command line parameter to program <code>add_noise</code> .
"	"	Absorbed programs <code>combine_mesh</code> , <code>combine_node</code> and <code>combine.poly</code> into <code>combine_files</code> .
"	"	Added support for <code>.ele</code> files in program <code>combine_files</code> .
"	"	Absorbed program <code>mesh2solid</code> into program <code>convert_format</code> .
"	"	Added new coupling weight parameters <code>wjfile</code> and <code>wjindex</code> in <code>coupling input files</code> .
"	"	Added <code>zrev2</code> command line parameter to program <code>node2vtu</code> to control field vectors.
"	"	Added support for Geosoft GXF grid files in program <code>convert_format</code> .
"	"	Extended program <code>smooth_node</code> to work with 3D surface-based models.
"	"	Added new program <code>pad_mesh</code> for padding a rectilinear mesh.
"	"	Added program <code>blocks2vtu</code> .
"	"	Added programs <code>node2d</code> and <code>node3d</code> .
"	"	Program <code>subdivide</code> now uses dynamic information from a node units file.
"	"	Added new parameter <code>ailsig</code> to program <code>data_difference</code> .
"	"	Added more outputs to program <code>data_difference</code> .
2016/11/07	2566	Added program <code>clean_surface</code> .
"	"	Extended program <code>convert_format</code> to convert <code>.node/.ele</code> to <code>.off</code> and <code>.obj</code> format.

\* The dates correspond to those when the public user repository was updated. The revision number is that for the main SVN repository accessed by users in my research group. Public users access a different repository so have different revision numbers.

# Bibliography

- A. Carter-McAuslan, et al. (2015). ‘A study of fuzzy c-means coupling for joint inversion, using seismic tomography and gravity data test scenarios’. Geophysics **80**(1):W1–W15, doi:10.1190/geo2014–0056.1.
- E. Fregoso & L. A. Gallardo (2009). ‘Cross-gradients joint 3D inversion with applications to gravity and magnetic data’. Geophysics **74**(4):L31–L42, doi:10.1190/1.3119263.
- L. A. Gallardo & M. A. Meju (2004). ‘Joint two-dimensional DC resistivity and seismic travel time inversion with cross-gradients constraints’. Journal of Geophysical Research: Space Physics **109**(B3):B03311, doi:10.1029/2003JB002716.
- T. Günther & C. Rücker (2006). ‘A new joint inversion approach applied to the combined tomography of DC resistivity and seismic refraction data’. In Proceedings of SAGEEP, Seattle, USA.
- P. G. Lelièvre (2009). Integrating geologic and geophysical data through advanced constrained inversions. Ph.D. thesis, Dep. of Earth and Ocean Sciences, The University of British Columbia, Vancouver, British Columbia, Canada.
- P. G. Lelièvre, et al. (2012). ‘Joint inversion of seismic traveltimes and gravity data on unstructured grids with application to mineral exploration’. Geophysics **77**(1):K1–K15, doi:10.1190/geo2011–0154.1.
- P. G. Lelièvre & D. W. Oldenburg (2009). ‘A comprehensive study of including structural orientation information in geophysical inversions’. Geophysical Journal International **178**(2):623–637, doi:10.1111/j.1365–246X.2009.04188.x.
- P. G. Lelièvre, et al. (2009). ‘Integrating geological and geophysical data through advanced constrained inversions’. Exploration Geophysics **40**:334–341, doi:10.1071/EG09012.
- Y. Li & D. W. Oldenburg (1998a). ‘3-D inversion of gravity data’. Geophysics **63**(1):109–119.
- Y. Li & D. W. Oldenburg (1998b). ‘Separation of regional and residual magnetic field data’. Geophysics **63**(2):431–439.
- Y. Li & D. W. Oldenburg (2000a). ‘Incorporating geological dip information into geophysical inversions’. Geophysics **65**(1):148–157, doi:10.1190/1.1444705.
- Y. Li & D. W. Oldenburg (2000b). ‘Joint inversion of surface and three-component borehole magnetic data’. Geophysics **65**(2):540–552, doi:10.1190/1.1444749.
- H. Paasche & D. Eberle (2009). ‘Automated integration of large geophysical data sets using three partitioning cluster algorithms: a comparison’. In 11th SAGA Biennial Technical Meeting and Exhibition, pp. 286–291, Swaziland. South African Geophysical Association.
- J. Sun & Y. Li (2017). ‘Joint inversion of multiple geophysical and petrophysical data using generalized fuzzy clustering algorithms’. Geophysical Journal International **208**:1201–1216.