Tutorials and documentation for various programs written at Mount Allison University and Memorial University

Peter G. Lelièvre

April 11, 2025

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

Contents

Part I

Preliminaries

Chapter 1

Policies

The following general polices pertain to use of the PODIUM and MAGNUM program suites, the FacetModeller utility and any other software described in this documentation, from herein referred to as the "software". These policies should not discourage use of the software but, rather, are meant to clarify the conditions of use and to protect the developers' significant investment in the software code.

In the following, "I" refers to the primary developer, Peter Lelièvre, and "the developers" refers to those owners currently administrating the distribution and licensing of the software (currently Peter Lelièvre and Colin Farquharson).

The disclaimers and conditions below are revocable by the developers at any time and may be replaced by a formal license agreement.

1.1 (Dis)claimer

While the software described in this documentation can not be guaranteed in any way, I provide the following assurances:

- I will keep this document up-to-date with the current version of the software.
- Unless otherwise stated in this document, these programs have been thoroughly tested and see frequent use. I am happy to run further tests with any specified parameters as requested.
- I value direct and constructive feedback: please contact me at plelievre@mta.ca for any questions, suggestions or requests with respect the software.
- I will deal with any reported bugs or issues in a timely manner, as my affairs allow.
- I will provide basic user support for the software, provided users have first reviewed this documentation thoroughly.

1.2 Conditions of use

The software is freely available for academic or other non-commercial use. Other uses require expressed agreement with the developers. Use of the software requires that the developers will not be held responsible for any damages, monetary or otherwise, to the user caused directly or in part through the use of the software.

User responsibilities:

1. properly cite usage of the software in any publications resulting from its use (suggested citations are supplied in the individual chapters of this document)
2. inform the developers of any such publications
3. inform the developers of any bugs or issues encountered while using the software
4. inform the developers if the user changes institutional affiliation or email address
5. do not make available nor distribute all or part of the software, by any means, to anyone
6. do not redistribute username or password details for the repository that houses the software
7. users understand that the username or password details for the repository may change periodically for security reasons
8. completely and immediately delete all components of the software if requested by the developers
9. update the software frequently (as instructed in this documentation)
10. users agree that their name and institution or company position may appear on the developers' academic websites in a list of users of the software.

Chapter 2

Obtaining the software

The software and documentation are upgraded frequently without announcement. Therefore, it is unfeasible for us to email the software directly to users. Instead, the software and documentation is housed on a repository that is made accessible to users.

2.1 MUN users

For those in my research group, I'm in the process of creating a dedicated second document with further instructions. Until then, please check our Wiki for more instructions for your particular platform (the Wiki is destined for deletion soon):

https://www.esd.mun.ca/geophysicswiki/index.php/Main_Page

https://www.esd.mun.ca/geophysicswiki/index.php/SVN_Subversion

2.2 Other users

> The instructions below are for using SVN from the command line. If needed, further instructions for creating sparse checkouts using SVN are here: http://svnbook.red-bean.com/en/1.7/svn.advanced.sparsedirs.html. Instructions for TortoiseSVN that might help do something similar are here: https://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-dug-checkout.html. If you have trouble with SVN, you can simply download the required files through a web browser. However, **I strongly encourage you to use SVN** so that you can easily and frequently update your software to the current version.

PLEASE READ THESE INSTRUCTIONS IN FULL BEFORE ATTEMPTING THEM.

For public users, I may have emailed you the required files directly, in which case you can ignore instructions relating to the public code repository. Otherwise, you'll need to connect to the public repository by taking the following steps:

1. Install an SVN Subversion client on your computer. SVN (http://subversion.apache.org) often comes pre-installed on unix-like operating systems (e.g. Mac, Linux). Options for Windows users include the command-line-based Cygwin (https://www.cygwin.com) and the graphical-interface-based TortoiseSVN (http://tortoisesvn.net). The instructions below are for use of unix-like operating systems, or Cygwin on Windows. Users of TortoiseSVN should refer to their online support page (http://tortoisesvn.net/support.html.)
2. Open a terminal/command window and move to the directory on your computer where you want to place the software.
3. Checkout the public repository. If you are using a unix-like operating system, or Cygwin, use the following command:

   ```
   svn co --depth empty https://www.esd.mun.ca/svn/repos2/
   ```

4. Enter the username and password that I should have already sent you via email. If you already have SVN on your computer then you may be asked for a password for your existing username: if so, just enter any random password, which will fail, and you will then be asked to enter a new username and password, at which point you can enter those I sent you via email. We will be changing the password every year or so, just to keep track of who has access, so if at any point your password doesn't work, please contact me for the latest password: plelievre@mta.ca.
5. The checkout above must be performed once and only once. You will be left with an empty directory named `repos2`. You now need to move into that directory (`cd repos2`) and then download whichever additional subdirectories that you need, depending on which code and which operating systems you want to use. The command to obtain any of the subdirectories is:

```
svn update subdir
```

where `subdir` is the name of the subdirectory you wish to obtain. The repository contains the following subdirectories:

- `dll` (contains dll files for use with the Windows executables if not using Cygwin)
- `doc` (contains documentation and example files, including this documentation file, doc.pdf)
- `facetmodeller` (distribution, documentation and tutorial files for the JAVA application `FacetModeller`)
- `lin` (executables for Linux for the `PODIUM` program suite)
- `mac` (executables for Mac OS X for the `PODIUM` program suite)
- `win` (executables for Windows for the `PODIUM` program suite)

The `svn update subdir` command may take some time to run, depending on your connection spead. In fact, the process may time-out and you would then need to perform the following commands to restart:

```
svn cleanup
```

```
svn update
```

You may have to do that multiple times, depending on your connection speed.

We suggest that users update the software frequently. To update to the latest version of the software on a unix-like operating system, or Cygwin, use the following command from the `repos2` directory:

```
svn update
```

2.3 Best practices

These are actively developed research codes that change frequently, and are changed by many developers. Therefore, all users should do the following things before starting work with the programs on any given day.

- Check repository for changes (`svn update`).
- Read log messages for information about changes (`svn log`). These messages may suggest that input files have changed format, or that default values have changed, or possibly other changes that might affect the reproducibility of your working examples.
- For those working with source code, recompile if you see something important that changed.

2.4 Supported operating systems

Current compilation platforms:

- The Mac executables are compiled on macOS 10.13.2 (High Sierra) using GNU Fortran 6.3.0.
- The Linux executables are compiled on Ubuntu ??? using ???.
- The Windows executables are compiled on Windows 7 using Cygwin ??? and GNU Fortran ???.

In addition to the platforms above, I can confirm that the executables also run on the following platforms:

- macOS 10.15.4 (Catalina) with GNU Fortran 8.2.0
- macOS 10.14 (Mojave) with GNU Fortran 7.1.0
- Windows 8.1 with Cygwin 3.1.5 and GNU Fortran 9.3.0.
- Windows 10 ??? with Cygwin ??? and GNU Fortran ???.

There should be no need to install Cygwin on Windows, or GNU Fortran (gfortran) on Mac or Linux, but without them you may be missing some .dll or .dylib files. If such files are missing, error messages will be provided when running the programs that will clearly indicate the missing files, and I can then supply you with anything missing.

FacetModeller should run on any operating system that has the most recent version of Java installed.

Chapter 3

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.

3.1 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.

3.2 Rectilinear meshes

3.2.1 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
E0 N0 V0
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 (`E0`,`N0`,`V0`) 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` repititions 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
```

3.2.2 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.

3.3 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
`meshtype`	`"unstructured"`	the type of mesh (the other option is `"rectilinear"`)
`meshfile`	`""`	file containing mesh information
`modelfile`	`""`	file containing model information
`neighfile`	`""`	another file containing mesh information (unstructured meshes only)
`split`	`0`	how to convert from rectilinear to unstructured mesh
`zdir`	`1`	specifies the coordinate system
`regionsfile`	`""`	a list of regions that can be used with a surface model
`nodeunitsfile`	`""`	a list of node-related rockunits that can be used with a surface model
`facetunitsfile`	`""`	a list of facet-related rockunits that can be used with a surface model
`regionunitsfile`	`""`	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`.

3.4 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
`proptype`	`""`	e.g. set to `"den"`, `"sus"` etc. for density, mag. susceptibility, etc.
		SCALING PARAMETERS
`mmul`	`1.0`	multiplicative scalar to convert model values to physical property values
`madd`	`0.0`	additiative scalar to convert model values to physical property values
`mtrend`	`0.0`	background physical property depth trend
`uselog`	`t`	if true then log(conductivity) is used; only applies to conductivity models
`inputscaled`	`t`	specify if input model quantities are scaled (true) or actual (false) physical properties
		INVERSION WEIGHTING PARAMETERS
`wmode`	`"none"`	defines what type of weighting is used in an inversion
`wbeta`	`1.0`	distance/sensitivity weighting strength
`wnorm`	`2.0`	distance/sensitivity weighting norm
`wpower`	`0.0`	depth/distance weighting power
`wzero`	`0.0`	depth/distance weighting z0/r0
`obsfile`	`""`	file containing observation locations for use with distance weighting
`zdir`	`""`	defines coordinate system for `obsfile` and `wzero` (if used)
`datatype`	`""`	specifies the data type (e.g. `"gz"`) from which information is taken from for sensitivity weighting
		INVERSION REGULARIZATION OPTIONS
`initfile`	`"null"`	file containing an initial model
`initindex`	`0`	attribute index for the initial model file
`initvalue`	`0.0`	initial model value
`reffile`	`"null"`	file containing a reference model
`refindex`	`0`	attribute index for the reference model file
`refvalue`	`0.0`	reference model value
`wsfile`	`"null"`	file containing smallness weights
`wsindex`	`0`	attribute index for the wsfile smallness weights file
`alphas`	`0.0`	multiplier on the smallness regularization
`alphab`	`1.0`	multiplier on both the smallness and smoothness regularization terms
		INVERSION CONSTRAINT OPTIONS

Name	Default	Brief description
`boundsfile`	`"null"`	file containing model bounds
`lowerindex`	`0`	attribute index to use for the lower bound in a bounds file
`upperindex`	`0`	attribute index to use for the upper bound in a bounds file
`lowervalue`	`-1E6`	lower bound value for the entire mesh
`uppervalue`	`1E6`	upper bound value for the entire mesh

3.4.1 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 (see Section 15.3.2); 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

$$p = \texttt{mmul} * m + \texttt{madd} + \texttt{mtrend} * (z - z0)$$

  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:

$$m = \big(p - \texttt{madd} - \texttt{mtrend} * (z - z0)\big)/\texttt{mmul}$$

  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.

3.4.2 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

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

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

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

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

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

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

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

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:

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

`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.

3.4.3 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.

`reffile, refindex, refvalue`

- The `refindex` 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.

`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.

3.4.4 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).

3.5 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
`coupling`	`"null"`	specifies the type of coupling
`proptype1`	`"null"`	specifies a property type for the first model in the coupling
`proptype2`	`"null"`	specifies a property type for the second model in the coupling
`proptype3`	`"null"`	specifies a property type for the third model (only used for FCM clustering)
`rho`	`0.0`	final multiplier value for the joint measure
`nsteps`	`1`	number of lambda steps over which to heat the rho value
`wjvalue`	`1.0`	constant coupling weight value (applied across entire mesh)
`wjfile`	`"null"`	optional file containing cell-centred coupling weights
`wjindex`	`1`	attribute index for the `wjfile` for the coupling weights
OPTIONS FOR LINEAR COUPLING		

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

3.5.1 Heating the coupling (parameters rho and nsteps)

A joint inversion proceeds in several stages, heating the coupling multiplier values in the objective function (see ρ_k in Section 16.5.3) from 0.0 at stage 0 up to the final value rho at stage nsteps.

3.5.2 Coupling options (parameter coupling)

eq, equal

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

$$\mathbf{m}_1 = \mathbf{m}_2 \tag{3.1}$$

eqgrad, equalgrad

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

$$\mathbf{G}\mathbf{m}_1 = \mathbf{G}\mathbf{m}_2 \tag{3.2}$$

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:

$$a_1\mathbf{m}_1 + b_1 = a_2\mathbf{m}_2 + b_2 \tag{3.3}$$

lingrad, lineargrad

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

$$a_1\mathbf{G}\mathbf{m}_1 + b_1 = a_2\mathbf{G}\mathbf{m}_2 + b_2 \tag{3.4}$$

`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 3.5.4 below)
- `"e"` for elliptical clusters (Mahalanobis distance); cluster centres, spreads and rotations must be specified, or centres, variances and covariances (see Section 3.5.4 below)
- `"l"` for a linear regression relationship; the linear regression parameters must be specified (see Section 3.5.4 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:

$$\frac{\mathbf{Gm}_1}{|\mathbf{Gm}_1|^2 + \texttt{epsilon}^2} = \frac{\mathbf{Gm}_2}{|\mathbf{Gm}_2|^2 + \texttt{epsilon}^2} \tag{3.5}$$

`jtv, tvar, totvar`

The joint total variation minimizes the following constraint vector:

$$\left((\mathbf{Gm}_1)^2 + (\mathbf{Gm}_2)^2 + \texttt{epsilon}^2\right)^{\texttt{power}/2} \tag{3.6}$$

3.5.3 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 (inside the ψ_k function in Section 16.5.3).

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

$$\Psi_{equal} = \sum_i w_i(m_{2,i} - m_{1,i}) \tag{3.7}$$

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,

$$\Psi_{gauss} = \sum_i \sum_j w_{ij} N(m_{1,i}, m_{2,i}, ...|\mu_j, \sigma_j), \tag{3.8}$$

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

3.5.4 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 3.4.1.

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:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \tag{3.9a}$$

$$a_{11} = \frac{\cos^2\theta}{\sigma_1^2} + \frac{\sin^2\theta}{\sigma_2^2} \tag{3.9b}$$

$$a_{22} = \frac{\sin^2\theta}{\sigma_1^2} + \frac{\cos^2\theta}{\sigma_2^2} \tag{3.9c}$$

$$a_{12} = \frac{\sin 2\theta}{2\sigma_1^2} - \frac{\sin 2\theta}{2\sigma_2^2} \tag{3.9d}$$

where σ_1, σ_2 and θ are values pulled from the `spreads1`, `spreads2` and `rotations` lists respectively. That matrix is used to calculate the Mahalanobis distance:

$$\begin{aligned} &(\mathbf{p}-\boldsymbol{\mu})^{\mathrm{T}}\,\mathbf{A}\,(\mathbf{p}-\boldsymbol{\mu}) \\ &= \begin{bmatrix} p_1-\mu_1 & p_2-\mu_2 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} p_1-\mu_1 \\ p_2-\mu_2 \end{bmatrix} \\ &= a_{11}\,(p_1-\mu_1)^2 + 2a_{12}\,(p_1-\mu_1)\,(p_2-\mu_2) + a_{22}\,(p_2-\mu_2)^2 \end{aligned} \tag{3.10}$$

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:

$$f(\mathbf{p}) = \exp\left(-\frac{1}{2}\,(\mathbf{p}-\boldsymbol{\mu})^{\mathrm{T}}\,\mathbf{C}^{-1}\,(\mathbf{p}-\boldsymbol{\mu})\right). \tag{3.11}$$

See 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:

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} \\ c_{12} & c_{22} \end{bmatrix} \tag{3.12}$$

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:

$$(\mathbf{p}-\boldsymbol{\mu})^{\mathrm{T}}\,\mathbf{C}^{-1}\,(\mathbf{p}-\boldsymbol{\mu}). \tag{3.13}$$

See https://en.wikipedia.org/wiki/Mahalanobis_distance and https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Density_function for more information on the mathematics.

3.5.5 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).

3.6 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`.

3.7 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. For example, Section 15.3.5 lists possible names for physical properties needed when forward modelling; Sections 17.4, 17.4.4 and 17.4.9 list possible names for physical properties and bounds when inverting for a surface-based model.

3.8 Optimization engine specification files

These files tell the inversion programs `VIDI` and `DYNO` how to solve the problems (what optimization options to use).

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
`engine`	`"local"`	set to `"local"`, `"pso"`, `"ga"` or `"mcmc"` (the latter three are still under development)
		LOCAL OPTIMIZATION OPTIONS
`gaussnewton`	`"f"`	if `"t"` (true) then the approximate Hessian is used
`usebounds`	`"f"`	set to `"t"` (true) to perform a bound-constrained inversion
`doquad`	`"f"`	set to `"t"` (true) to perform a quadratic line search as well as bisection
`maxstepsi`	`-1`	max. # of model perturbations in first minimization (burn-in), set ≤ 0 to use `maxsteps0`
`maxsteps0`	`2`	max. # of model perturbations when far from misfit target
`maxsteps1`	`-1`	as for `maxsteps0` but used when close to misfit target, set ≤ 0 to use `maxsteps0`
`maxstepso`	`-1`	max. # of model perturbations in last minimization (burn-out), set < 0 to use `maxsteps1`
`maxstepsj`	`0`	added to maxstepsi/0/1/o when in the joint inversion stages
`omega1`	`2.0`	target-normalized misfit that determines when `maxsteps1` takes effect
`cgmaxit`	`2000`	maximum iterations for the CG algorithm when solving for the search direction
`cgtol`	`1.0E-3`	tolerance for the CG algorithm when solving for the search direction
`lstol`	`1.0`	tolerance on relative objective function value for the model perturbation line search
		GLOBAL OPTIMIZATION OPTIONS
`multiobj`	`"t"`	single- or multi-objective function inversion
`discrete`	`"f"`	continuous or discrete physical properties?
`usex0`	`"t"`	if `"t"` (true) then the initial model is used to initialize the global solver
`maxit`	`0`	maximum iterations of global optimization engine; set ≤ 0 for no maximum
`nind`	`0`	number of individual solutions, e.g. number of particles for PSO or population size for GA
`find`	`2.0`	size factor: if `nind`=0 then `nind` is set to $round$(`find` $\times$ `npar`)
`npar`	`0`	number of parameters (length of each solution vector)
`nobj`	`0`	number of objective/goal functions
`ncon`	`0`	number of constraints
`eps`	`1.0E-3`	convergence tolerance
`targ`	`0.0`	function value target; stop when objective function falls below this
`timelimit`	`0`	run time limit in minutes; set ≤ 0 for no time limit
`iverb`	`0`	verbose printing to screen every `iverb` iterations; set ≤ 0 for no printing
`iwrite`	`0`	if > 0 then the output files are written every `iwrite` iterations
`fiditer`	`0`	file used for writing the objective function and solution every `iverb` iterations
`fidbest`	`0`	file used for writing the best objective function and related solution so far
`writebest`	`f`	if `"t"` (true) then the output files are written whenever an improved solution is found
		GENETIC ALGORITHM OPTIONS
`separate`	`"t"`	how to perform the crossover and mutation
`fmutat`	`0.2`	fraction of children created by mutation v.s. crossover; only used if `separate` is `"t"` (true)
`fox`	`" "`	a "fox" output file from a previous optimization used to initialize the current optimization
		GA SELECTION OPTIONS
`selection`	`"tour"`	the type of selection used: `"tour"` for tournament, `"roul"` for roulette wheel
`ntourn`	`2`	tournament size; only used if `selection` is `"tour"`
`tourndup`	`"f"`	set to `"t"` (true) to allow duplicates in the tournament population
`coupling`	`"all"`	rules for tournament coupling: `"all"`, `"self"` or `"couples"`
		GA CROSSOVER OPTIONS
`crossint`	`"double"`	the type of crossover for integer problems: `"single"`, `"double"` or `"uniform"`
`crossreal`	`"sbx"`	the type of crossover for real problems: `"blx"` or `"sbx"`
`alpha`	`0.5`	BLX-alpha value for real crossover; see Deb et al. (2002)
`pcross`	`0.9`	SBX real crossover probability on [0.0,1.0]; see Deb et al. (2002)
`ncross`	`20.0`	n value for SBX real crossover; see Deb et al. (2002)

Name	Default	Brief description
		GA MUTATION OPTIONS
mutation	"poly"	the type of mutation used: "uniform", "normal", "poly", "probe" or "litho"
pmutat	0.1	mutation probability on [0.0,1.0]
sigma	0.5	standard deviation for normal (Gaussian) mutation relative to bounds
shrink	0.5	shrinking factor for normal (Gaussian) mutation
nmutat	20.0	n value for polynomial real mutation; see Deb et al. (2002)
		GA ELITISM OPTIONS
elitism	"all"	the type of elitism used: "none", "some" or "all"
nelite	1	number of elite individuals to maintain between generations; single obj. problem only
fpareto	0.35	maximum fraction of population pulled from the first non-dominated front each generation
chicrowd	-1.0	if >0 then crowding distance is set HUGE for solution with misfit closest to chicrowd
cutcrowd	-1.0	if >0 then crowding distances are decreased for solutions with misfit > cutcrowd
		GA CONVERGENCE OPTIONS
nstallobj	50	number of generations across which to check for stall on objective space
nstallpar	50	number of generations across which to check for stall on parameter space
stalltolobj	1.0E-6	tolerance for stall check in objective space
stalltolpar	1.0E-6	tolerance for stall check in parameter space

3.8.1 General optimization options

`engine`

- Option `local` runs a Newton-style descent-based local optimization.
- Option `pso` runs a particle swarm global optimization strategy.
- Option `ga` runs a genetic algorithm global optimization strategy.

`multiobj`

- Only used with global optimization strategies, although multi-objective optimization is not yet ready for the PSO engine.
- If `"t"` (true) then the different terms in the objective function (e.g. misfit, model regularization and joint coupling) are kept separate and many solutions on the Pareto front are found.
- If `"f"` (false) then the different terms in the objective function are summed into a single objective and a single best solution is found.
- If there is only one term in the objective function then single-objective optimization is always performed.

3.8.2 Local optimization options

`gaussnewton`

- This parameter is only applicable to magnetics inversions with the `amp` data type or `aid` model type.
- Thorough testing has yet to be performed but preliminary testing suggests that the higher order terms in the Hessian calculation are important and therefore this parameter should be set to `"f"` (false).

`lstol`

- If Φ is the objective function, $\mathbf{m}$ is the current model and $\boldsymbol{\delta}\mathbf{m}$ is the current search direction (model perturbation) then the line search for the model perturbation scaling factor α stops once

$$\frac{\Phi(\mathbf{m}+\alpha\boldsymbol{\delta}\mathbf{m})}{\Phi(\mathbf{m})} < \texttt{lstol}. \tag{3.14}$$

- If `lstol` is set to one then α is accepted for any reduction in the objective function.

3.8.3 Global optimization options

`discrete`

- If `"f"` (false) then the problem parameters are treated as continuous variables (between bounds).
- If `"t"` (true) then the problem parameters are treated as integers.

`nobj`

- This should always be equal to the number of objectives for the multi-objective problem, regardless of the value of `multiobj`.
- If `multiobj` is `"f"` (false) , meaning it is a single objective problem, then the `nobj` objectives are summed into the single aggregate objective that is minimized.
- If `discrete` is `"t"` (true) then the number of objectives is equal to the number of datasets.
- If `discrete` is `"f"` (false) then the number of objectives is equal to the number of datasets plus the number of physical properties plus the number of joint coupling pairs.

`iwrite, writebest`

- `iwrite` is currently only used for MCMC, PSO and GA.
- `writebest` is currently only used for PSO and GA.

3.8.4 Genetic algorithm options

`separate, fmutat`

- If `separate` is `"t"` (true, the default) then the crossover and mutation operators occur separately: $(1.0 - \texttt{fmutat}) \times \texttt{nind}$ children are generated from crossover, and $\texttt{fmutat} \times \texttt{nind}$ different children are generated from mutation, and those children are combined into the population of children.
- If `separate` is `"f"` (false) then `nind` children are generated from crossover and then those children are mutated.

`coupling`

- This parameter is only used if `selection` is `"tour"`.
- If `coupling` is `"all"` then all couples (father-mother pairs) are allowed.
- If `coupling` is `"self"` then no self duplication (father = mother) is allowed.
- If `coupling` is `"couples"` then no duplicated couples are allowed.

`sigma, shrink`

- The standard deviation for normal (Gaussian) mutation is determined by the following equation:

$$std = \texttt{sigma} \times (1.0 - \texttt{shrink} \times (i-1)/\texttt{maxit}) * (b_u - b_l)$$

  where i is the current generation count, and b_l and b_u are the lower and upper bounds for a particular parameter respectively.

`elitism, nelite, fpareto`

- If `elitism` is `"none"` then no parents pass into the next generation.
- For single objective problems: if `elitism` is `"some"` then the best `nelite` parents pass into the next generation; if `elitism` is `"all"` then all parents and children are ranked together and the best pass into the next generation.
- For multi-objective problems, if `elitism` is `"some"` or `"all"` then `fpareto` is used. If `fpareto` = 1.0 then the new population for the next generation is filled with individuals from the first non-dominated front, then the second, third, etc., until the new population is completely filled. If `fpareto` < 1.0 then only that fraction of the individuals are taken from the first non-dominated front and the portion taken from later fronts follows a geometric reduction.

`nstallobj, nstallpar, stalltolobj, stalltolpar`

- Two of the convergence criteria are stalls in objective space and parameter space.
- These convergence tests are only performed for single objective problems.
- If the relative average change of the best single objective function value across `nstallobj` generations is below `stalltolobj` then the optimization stops.
- If the relative average change of the best solution (a 2-norm of the model parameters) across `nstallpar` generations is below `stalltolpar` then the optimization stops.
- Set `nstall`$* \geq$ `maxit` or `nstall`$* \leq 1$ or `stalltol`$* = 0.0$ to avoid these convergence tests.

3.9 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)
- `(xci,[yci],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]
```

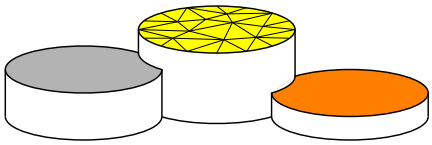
where that "A" character must exist at the start of the line.

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.

Part II

PODIUM



A suite of software utilities for

Preparation Of Data for Inversion on Unstructured Meshes

Chapter 4

PODIUM preliminaries

4.1 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.

4.2 Policies and conditions of use

Refer to Chapter 1 for general disclaimers and conditions of use.

Suggested citation:

- P. G. Lelièvre and C. G. Farquharson, 2015, PODIUM: a suite of software utilities for Preparation Of Data for Inversion on Unstructured Meshes

4.3 Obtaining PODIUM

Refer to Chapter 2 for general instructions on obtaining the software described in this documentation.

4.4 Installing PODIUM

Source code is NOT available. PODIUM is provided as a collection of compiled executables for particular operating systems. Hence, no installation is required once the programs have been obtained from the repository. We make efforts to test the compiled programs on various operating systems but without direct access to users' computing systems we can not guarantee that these codes will execute on your system. The `mac` version is currently being compiled on OS X 10.13.2 (High Sierra), the `win` version on Windows 7 and the `lin` version on Ubuntu, all using `gfortran` as the compiler.

4.5 Running the programs

The programs are run in the command line, e.g. from a Terminal window on a Mac or a Command Prompt window on a Windows machine. No graphical user interfaces have been developed for two reasons: 1) it is simple to develop batch scripts using programs developed for command line use; 2) we feel that the significant development time required for a GUI is better spent developing new functionality. The documentation below should be reviewed carefully before running the programs. Any program can be run with no input parameters (that is, simply type the name of the program in a command window and hit return/enter) to obtain a short reminder of the command line usage.

Many of these programs will run in parallel (via OpenMP directives). Remember to set the OMP_NUM_THREADS environment variable. On a Mac you use the following command in the terminal window before running this program:

```
> export OMP_NUM_THREADS=value
```

4.6 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 5.

4.7 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 LaTeX 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/latittude 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 statistcial 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 know 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 5

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`.

5.1 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

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.1.6 mesh2tikz

Converts `.node` and `.edge` files to a `.ltx` file containing plotting information for typesetting with LaTeX 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.

5.1.7 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.

5.1.8 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.

5.1.9 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.

5.1.10 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.

5.1.11 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.

5.1.12 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`.

5.1.13 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.

5.2 Peter's file formats

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

5.2.1 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 3.9.
- 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).

5.2.2 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 3.9.
- 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).

5.2.3 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 3.7.
- 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.

5.2.4 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 3.7.
- 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.

5.3 UBC-GIF file formats

See Section 3.2 for more information on UBC-GIF mesh and model file formats.

5.3.1 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.

5.3.2 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.

5.3.3 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 +/-1 splits rectangles into two triangles and prisms into five tetrahedra with alternating splitting in neighbours
  - a value of +/-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.

5.3.4 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!

5.3.5 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.

5.3.6 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).

5.3.7 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)

5.4 Industry file formats

5.4.1 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 `outputfile.node/.ele/.vtu` are written if `outputfile` 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 `outputfile.node` is written if `outputfile` has no extension.
Geonics EM31 (Geosoft xyz)	`.xyz`		File `outputfile.node` is written if `outputfile` has no extension.
ESRI grid	`.grd .asc`		(As above)
Geosoft GXF grid	`.gxf`		(As above)
WSINV3DMT model	`.model .initmod`	`[.sitenames]`	Files `outputfile.mesh/.con/.vtr` are written if `outputfile` has no extension.
WSINV3DMT data	`.data .responses`	`[.sitenames]`	Files `outputfile.txt/.vtu` are written if `outputfile` has no extension.
Triangle/TetGen mesh	`.node`	`.ele`	File `outputfile.ts` or `outputfile.so` is written (for 3D or 2D respectively) if `outputfile` has no extension. If `outputfile` is named `mesh3d.dat` then it is written in that R3t file format.
CSV file, e.g. Oasis Montaj, QGIS	`.csv`		File `outputfile.node` is written if `outputfile` has no extension.
GEOMAR AUV MAG Survey	`.txt`		(As above)
SCINTREX IPR-12 .dmp file	`.dmp`	`.txt`	The `outpuffile` should have no extension.
Column-based DC/IP .txt file	`.txt`		The `outpuffile` should have no extension.

5.5 Combining multiple files of the same format

5.5.1 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` ≤ 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 us 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` ≤ 16 can be increased by the developers if you need.

Chapter 6

Data processing

6.1 Observation creation and data manipulation

6.1.1 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 $\mathbf{d}$ hold the data values d_i. The standard deviation of the noise is then calculated as

$$\sigma_i = \texttt{flo} + (\texttt{perflo} \times range(\mathbf{d})/100) + (\texttt{per} \times |d_i|/100) \tag{6.1}$$

  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 ϵ_i is created by multiplying those σ_i values by values taken from a random normal distribution with zero mean and unit standard deviation, $N(0,1)$:

$$\epsilon_i = \sigma_i N(0,1)_i = N(0,\sigma_i^2)_i \tag{6.2}$$

  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

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\epsilon_i}{\sigma_i}\right)^2 = \frac{1}{n}\sum_{i=1}^{n} N(0,1)_i^2 \leq \texttt{tol} \tag{6.3}$$

  where n is the number of data. This is helpful when running synthetic inversions where you want the theoretical expected value for the χ^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.

6.1.2 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.

6.1.3 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.

6.1.4 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 3.2.1 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.

6.1.5 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.

6.1.6 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.

6.1.7 ll2utm

Converts the coordinates from a 3D `.node` file from longitude/latittude 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`.

6.1.8 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:

- `nidm` 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 $x = -10$m to $x = 10$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 $x = -10$m to $x = 10$m every 2m, and from $y = -20$m to $x = 20$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.

6.1.9 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

  $$\texttt{kx}, \texttt{ky}, \texttt{kz}$$

  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

  $$\texttt{lx}, \texttt{ly}, \texttt{lz}$$

  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.

6.1.10 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.

6.1.11 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.

6.1.12 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.

6.1.13 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:

$$\begin{aligned} t &= a_0 + a_{01}\tilde{x} + a_{10}\tilde{y} + a_{11}\tilde{x}\tilde{y} + a_{20}\tilde{x}^2 + ... + a_{ij}\tilde{x}^i\tilde{y}^j + ... + a_{nn}\tilde{x}^n\tilde{y}^n \\ \tilde{x} &= x - x_{min} \\ \tilde{y} &= y - y_{min} \end{aligned} \tag{6.4}$$

  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`.

6.1.14 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.

6.1.15 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.

6.1.16 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
```

6.2 Data visualization

See programs node2vtu and ele2vtu.

6.2.1 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.

6.2.2 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 inverval 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
...					

6.3 Data query and calculations

6.3.1 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.

6.3.2 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.

6.3.3 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 `"l"`) 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.

6.3.4 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.

6.3.5 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` ≤ 0 then `i1` defaults to 1. If `i2` ≤ 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 ≤ 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 $b_2 d_2 - b_1 d_1$, 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.

6.3.6 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.

6.3.7 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`.

6.3.8 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`.

6.3.9 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.

6.3.10 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.

6.3.11 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.

6.3.12 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.

6.3.13 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.

6.4 Interpolation and projection

See also program interpolate_mesh.

6.4.1 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.

6.4.2 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.

6.5 Utilities for seismic and muon tomography data

6.5.1 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.

6.5.2 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.

6.5.3 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 7

Model processing

7.1 Model creation and manipulation

7.1.1 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
```

7.1.2 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).

7.1.3 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`.

7.1.4 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.

7.1.5 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.

7.1.6 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 3.9. 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.

7.1.7 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.

7.1.8 mark_model

This program has been absorbed into program populate_model and is now obsolete.

7.1.9 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.

7.1.10 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.

7.1.11 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
```

7.1.12 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.

7.1.13 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.

7.1.14 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`¿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`.

7.1.15 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.

7.1.16 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`.

7.1.17 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.

7.1.18 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.

7.1.19 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
```

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.
- If `ai` is present then it specifies which model attribute to smooth. If absent, all attributes are smoothed.
- The smoothing occurs `nsmooth` times via a simple neighbour-averaging method.
- The smoothed information is written to file `outfile`.

7.1.20 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.

7.1.21 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 ≤ 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.

7.1.22 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` ≥ 1 then normals are propagated from the cell (surface facet) with that index and `dir` is ignored.
  - If `icell` ≤ 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` ≥ 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` ≤ 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` ≤ 0 then the vector must be specified.
  - If `igroup` ≥ 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`.

7.1.23 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`.

7.1.24 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.

7.1.25 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`.

7.2 Model visualization

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

7.2.1 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`.

7.2.2 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 `outlinefile`s 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.

7.3 Model query and calculations

7.3.1 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.

7.3.2 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` multiples 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.

7.3.3 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 3.5) 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.

7.3.4 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.

7.3.5 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)

7.3.6 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)

7.3.7 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

7.3.8 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.

7.3.9 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.

7.3.10 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.

7.3.11 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.

7.3.12 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.

7.3.13 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.

7.3.14 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).

7.3.15 pun2nun

Creates two surface meshes that represent ± 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 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):
  - `positive standard deviation =` $(2.33E-7)prop^4 - (4.9E-5)prop^3 + (3.06E-3)prop^2 + (1.1E-2)prop + 2$
  - `negative standard deviation =` $(3.07E-10)prop^6 + (5.39E-8)prop^5 + (2.06E-6)prop^4 - (2.02E-5)prop^3 + (2.32E-3)prop^2 - (2.17E-3)prop + 2$

  These functions are nomalized by the maximum of a node's position contraint.
- 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%.

7.3.16 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`.

7.3.17 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,∞] then the result lies on [0,1].

Notes:

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

7.3.18 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] m^3
Volume 2 is:  [number] m^3
...
Volume N is:  [number] m^3
```

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 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, **Part 4 -region attribute lits**). Have the attibute 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
     ```

7.3.19 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.

7.4 Creation of inversion input files

7.4.1 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.

7.4.2 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.

7.4.3 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.

7.4.4 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.

7.5 Interpolation

7.5.1 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 exisiting `.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 exisiting 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.
- `nl` 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 property. Increase `nl` above `1` if you get poor results. Increasing the value of `nl` will increase the run time so do this slowly until you obtain appropriate results. Using a huge value of `nl` 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 `nl` 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 `nl` times
    4. check if P is in or on any of the cells found above.

Notes:

- The percentage completed is printed to the screen.

7.5.2 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 8

Miscellaneous utilities

8.1 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.

8.2 log2aux

Takes numbers from a DYNO `.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.

8.3 rel2abs

Reads a `.node` file and converts relative node location bounds to absolute. This program is for use with the inversion program `DYNO`, 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.

8.4 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 ommitted in the output fie.

Part III

FacetModeller

Chapter 9

FacetModeller preliminaries

9.1 About FacetModeller

FacetModeller helps you build models comprising surfaces that represent contacts (interfaces) between rock units. The surfaces are represented by nodes (vertices) and connections between them (facets). These models are stored as planar straight line graphs (PSLGs) and piecewise linear complexes (PLCs) in 2D and 3D respectively. These are for use with meshing programs like Triangle and TetGen and are explained in more detail on these webpages:

- Triangle: 2D triangular meshing program
  http://www.cs.cmu.edu/~quake/triangle.defs.html#pslg
- TetGen: 3D tetrahedral meshing program
  http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual002.html#sec7

Before using FacetModeller you should read those pages and understand the concept of a PSLG and PLC, and what other information is required for use of Triangle and TetGen.

FacetModeller was initially created to help students manually build 3D models from stacks of vertical cross-sections. It has grown considerably since then to help others with their modelling needs. FacetModeller is not intended to be a replacement for GOCAD (or any other 3D modelling programs) and it will never do some of the fancy automatic model building tasks that GOCAD can handle: FacetModeller is intended to be a simple-to-use, manual model-building program. Given that FacetModeller has grown from the needs of a small set of researchers, it may not meet all your needs. Therefore, **we encourage users to suggest additional functionality or other general improvements that would help you with your model building needs:** please contact the primary developer at plelievre@mta.ca.

9.2 Policies and conditions of use

Refer to Section 1 for general disclaimers and conditions of use.

FacetModeller has been submitted to the journal SoftwareX for publication. Until then, the suggested citation is:

- P. G. Lelièvre and C. G. Farquharson, 2015, FacetModeller: software for manual creation, manipulation and analysis of 3D surface-based models

9.3 Obtaining, installing and running FacetModeller

9.3.1 Obtaining FacetModeller

It's on GitHub. Do a Google search and find it. I recommend installing the GitHub Desktop App and then using it to clone my facetmodeller project. Alternatively, just follow the instructions on the GitHub page for that project: there is a "dist" directory that you can grab and run the contained .jar file.

9.3.2 Installing Facetmodeller

FacetModeller is a Java application. To run the application you must make sure you have the latest version of Java installed on your machine. Because the FacetModeller application is written in Java, it should in theory run on any platform that can run Java. We make efforts to test the application on various operating systems but without direct access to users' computing systems we can not guarantee that the application will execute flawlessly on your system.

9.3.3 Running FacetModeller

To run FacetModeller you must first find the `FacetModeller.jar` file: for those in my research group, use the version here:

`[repos]/peter/NetBeansProjects/FacetModeller/dist/`

where `[repos]` is the directory on your local machine that holds my repository directory.

For public users, use the version here:

`[repos2]/facetmodeller/dist/`

where `[repos2]` is the directory on your local machine that holds the public repository directory.

To launch FacetModeller,

- Run the `FacetModeller.jar` file by double clicking it, or from a terminal window using
  `java -jar FacetModeller.jar`
  or, for those in my research group, run it from the NetBeans IDE if you went that route.

9.4 About these tutorials

The best way to learn what you can do with the FacetModeller utility is to run through some quick tutorials. The following chapters provide tutorials that introduce the basic working concepts of the utility and some essential functionality. I have tried to design the application to be as intuitive as possible so hopefully you can figure out the functionality not explained here through your own exploration.

Although I try to update this documentation frequently, my code changes more frequently, so there may be differences between what you see in these tutorials and what you see when running a program. Please let me know if there are major inconsistencies that cause confusion. Also let me know if you have any suggestions that could improve the utility of these programs.

A paper on FacetModeller has been submitted to the journal SoftwareX. Once it has been accepted, new and improved tutorials will be developed based on the examples in that paper. Please stay tuned.

Throughout these tutorials you'll see some "asides", which look like ... well, like this! These can more-or-less be safely skipped if you want a faster introduction to the programs. However, I suggest you at least read them for the information provided, even if you don't follow the additional steps.

9.5 The model building philosophy of FacetModeller

FacetModeller uses a section-based model building philosophy. A section is simply a planar slice through a 3D model. For 2D models, there is only a single section. A section may be a vertical cross section, a horizontal depth section, or any planar rectangular section of arbitrary orientation.

Each node in a PSLG or PLC is attached to a single section. In a 2D model there is only a single section and all nodes are attached to that section. In a 3D model there are multiple sections and there will be different nodes attached to different sections. Nodes usually lie exactly on a section but in some cases they may lie off of the section and are projected onto it during visualization.

Facets are collections of nodes. In 2D models, facets are line elements that connect two nodes. In 3D models, facets are triangles or other planar polygonal shapes. If a facet's nodes are all attached to the same section then that facet will belong to a single section. If a facet's nodes are attached to different sections then that facet will belong to several sections.

Each watertight part of a model is a different "region" and use of some features of Triangle and TetGen requires that you provide a spatial location inside each of those different regions. FacetModeller stores each of those region specification points as a special kind of node and defining the regions is similar to defining nodes: each region is attached to a single section.

FacetModeller distinguishes different types of nodes, facets and regions by assigning them to different "groups". Each group has user-defined plotting colours associated with it. You can use the concept of groups to help you organize the different components in your model. For regions, the groups work like unique rock unit identifier integers. Actually, they are more like unique integers for each separate watertight region in a geological model (there may be multiple watertight regions that contain the same rock unit). Therefore, each group can only have one region.

Furthermore, each node, facet and region can only be assigned to a single group. In practice, this limitation means that you often have to define many groups. For example, consider the case where your model contains two intersecting surfaces: a vertical surface which we'll call "Surface V" and a horizontal surface which we'll call "Surface H". You will want some nodes associated with a group named "Surface V" and others associated with a group named "Surface H". You may also want the nodes along the intersection to be associated with a group named "Intersection H-V". That is simple enough to achieve in FacetModeller with three groups. However, now consider the same situation but you now want to be able to differentiate between nodes on "Surface V" that lie above the intersection, and nodes on "Surface V" that lie above the intersection, and similarly for the two sets of nodes on "Surface H" that lie on either side of the intersection. In FacetModeller you now need five different groups. This sometimes makes for a less-than-optimal organization of the model components. However, there are ways to merge groups, and to change node and facet group membership, which both help the process. Future development will look at ways to ease this limitation.

Chapter 10

Tutorial - Building 2D models with FacetModeller

This 2D tutorial provides basic information about FacetModeller. You should follow through this tutorial even if you are only interested in building 3D models, because the 3D tutorials rely on lessons learned here. **You should also make sure that you have read and understood the model building philosophy of FacetModeller in Section 9.5**.

10.1 Launching FacetModeller

To launch FacetModeller,

7. Run the `FacetModeller.jar` file by double clicking it, or from a terminal window using
   `java -jar FacetModeller.jar`
   or, for those in my research group, run it from the NetBeans IDE if you went that route.
8. Choose `2D` when asked what kind of model you want to create.

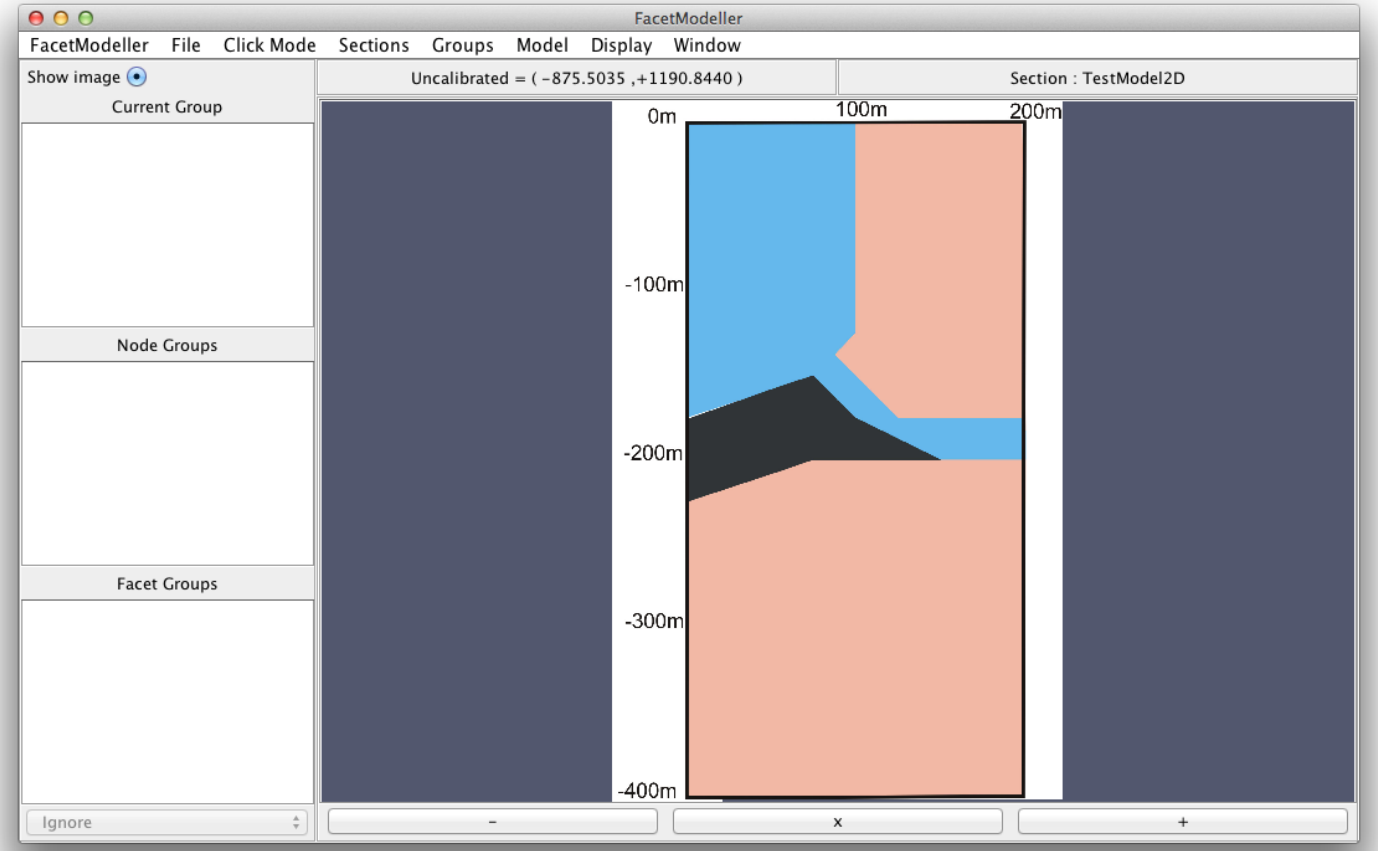
10.2 Opening your image file

The first step is to open the image file you want to discretize.

1. Select the menu item
   `File > Load cross section image file`
2. Select the file named `TestModel2D.png`. Those in my research group can find it here
   `[repos]/peter/NetBeansProjects/examples/FacetModeller2D/`
   and public users here
   `[repos2]/facetmodeller/Tutorial_2D/`
3. Close the confirmation that the file was loaded successfully by clicking `OK`.

> If the image represented a horizontal depth section, rather than a vertical cross section, then you would use the menu item
>
> `File > Load cross section image file`

You should now see something like this (being a Java application, it may look slightly different on different systems; I'm on a Mac):



There are 3 units in this model:

1. gneiss (the pink part of the image)
2. troctolite (the blue part of the image)
3. sulphides (the dark grey, almost black, part of the image)

10.3 Defining groups

You first need to define some groups so the nodes and facets that you define later can be associated with different groups. This isn't critical: you could just define some default group and use that for all nodes and facets. However, different groups of nodes and facets are coloured differently and this will help keep things clear. To define a group:

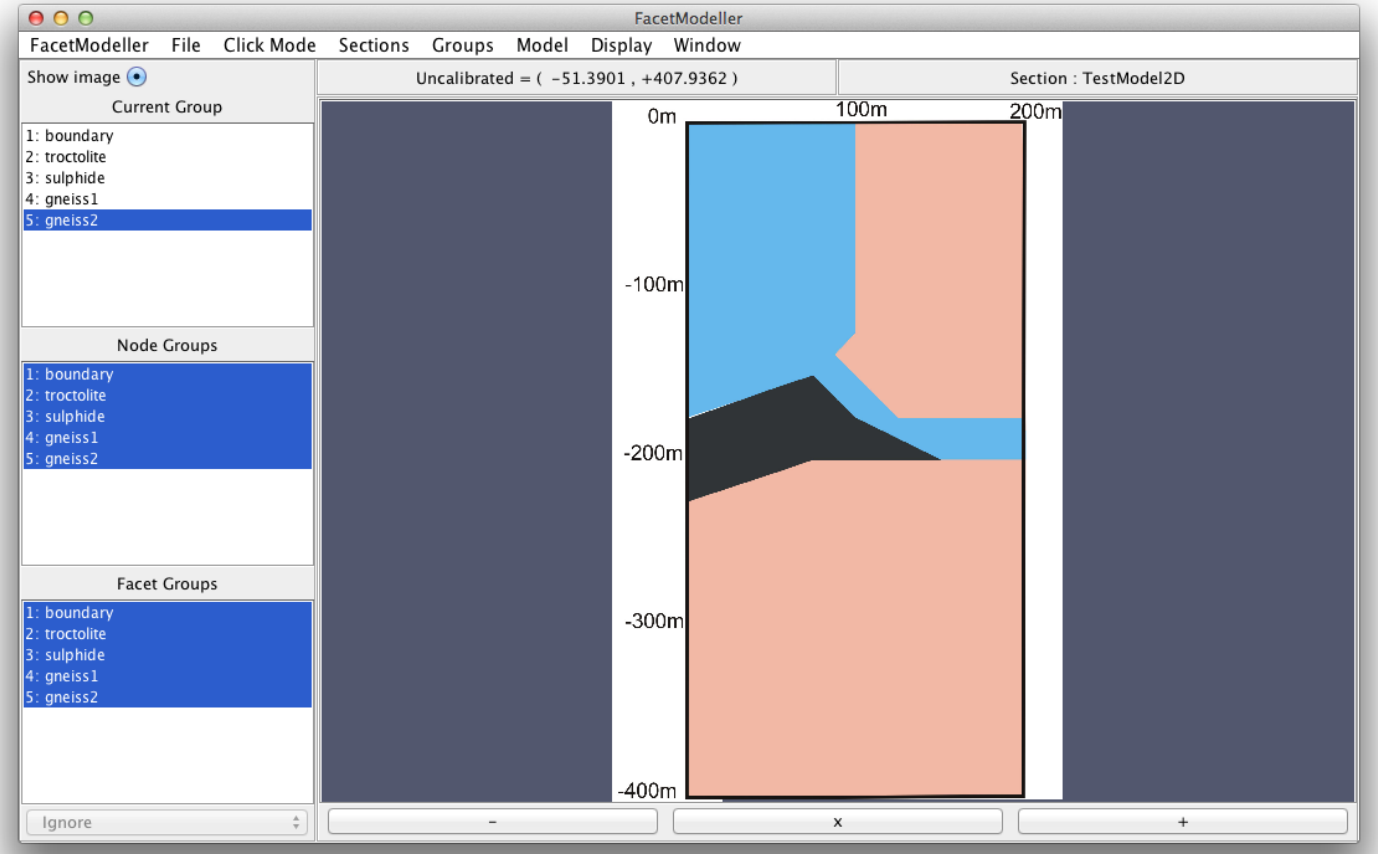
1. Select the menu item
   `Groups > New group`
2. Enter `boundary` as the name for this first group.
3. Select red as the drawing colour for the group.

> The group colour is used when drawing the nodes, facets and regions associated with the group. Initially, all three colours are set the same but they can be chosen separately using the various options in the `Groups` menu. Because you will be using this `boundary` group to define the boundary nodes, and the boundary in this image is black, red is a good choice because it will stand out against black. Of course you can use any colour you like in general.

Repeat those steps until you have defined the 5 groups with the names and colours indicated in the table below.

Name	Colour
boundary	red
troctolite	green
sulphide	white
gneiss1	black
gneiss2	black

You should now see those group names filled into the group selection boxes on the left. The group selection box at the top (labelled `Current Group`) allows you to specify which group you are currently working with, e.g. adding nodes or facets to. Only one group can be selected as the `Current Group`. The other two selection boxes (labelled `Node Groups` and `Facet Groups`) allow you to select which groups of nodes and facets are overlaid on the image. Multiple selections are allowed for the `Node Groups` and `Facet Groups` selection boxes. To select multiple items, use your shift or control/command keys while selecting the items.



10.4 Saving your progress

This is research code, still susceptible to bugs (but what isn't) so you should save early and often:

1. Select the menu item
   `File > Save session`
2. Select the location to save to and enter the name of the file to save. The file extension will be `.fms`.

You may want to save different versions of your model as you build it. To change the name of the session file that you save to:

3. Select the menu item
   `File > Save session as`
4. Select the location to save to and enter the new name of the file to save.

10.5 Calibrating the image

As you move your cursor around on the image you will see the location of the cursor indicated in a text box at the top of the FacetModeller window. Initially you will see "Uncalibrated" coordinates displayed in that text box, which are pixel coordinates for the image.

Before you can define nodes and facets, you must first calibrate the image. You will be instructed to click on two calibration points and enter their spatial coordinates in 3D.

Yes, I said 3D, and I mean 3D, despite this being a 2D model. I have done this to allow a 2D section to be registered in 3D space, in case you want to display it in a 3D viewer (such as ParaView) along with other spatially registered items. However, if you want a true 2D model, e.g. for performing 2D forward modelling, then you can simply choose to set the easting or northing values for the calibration points to equal values and FacetModeller will write a true 2D model when exporting files.

1. Select the menu item
   `Sections > Calibrate current section`
2. Close the first instruction dialog (instructing you to click on the first calibration point).
3. Click on the **top left** corner of the black boundary of the model.
4. Enter `"0 0 0"` (without the quotation marks) when asked for the easting, northing and elevation coordinates of that first calibration point.
5. Close the second instruction dialog.
6. Click on the **bottom right** corner of the black boundary of the model.
7. Enter `"200 0 -400"` as the coordinates of that second calibration point.

The image is now spatially calibrated. As you move your cursor around on the image you will see the calibrated spatial (easting, northing, elevation) location of the cursor indicated in the text box at the top of the FacetModeller window.

Here you used the axis labels present in the image to help with the calibration. Of course you won't always have that information on your image but you will at least need to know the coordinates of two landmarks in order to perform the calibration. Those landmarks must not lie at the same horizontal or vertical position!

10.6 Adding nodes

You are now ready to define the nodes in your model PSLG.

Before doing so, you may like to change some drawing defaults to make the node and facet overlays show up better:

1. Select the menu item
   `Display > Point size`
2. Enter `10`.
3. Select the menu item
   `Display > Line width`
4. Enter `2`.

You will first define some nodes around the boundary of the model:

5. Select the menu item
   `Click Mode > Add nodes`
6. Select the group named `boundary` in the `Current Group` selector box (any new nodes added will be assigned to the currently selected group).

7. Click around the boundary in the locations where you want new nodes. Make sure you add a node at the 4 corners and anywhere else on the boundary that intersects with rock unit contacts in the model. You should have a total of 9 nodes on the boundary (refer to the next screen-shot for guidance).

> As you click to add the boundary nodes, red node points will be overlaid on the image in the locations that you click. It may be a little fiddly: you have to press and release the mouse without the cursor moving (this is a safety precaution that I can remove if it annoys you).

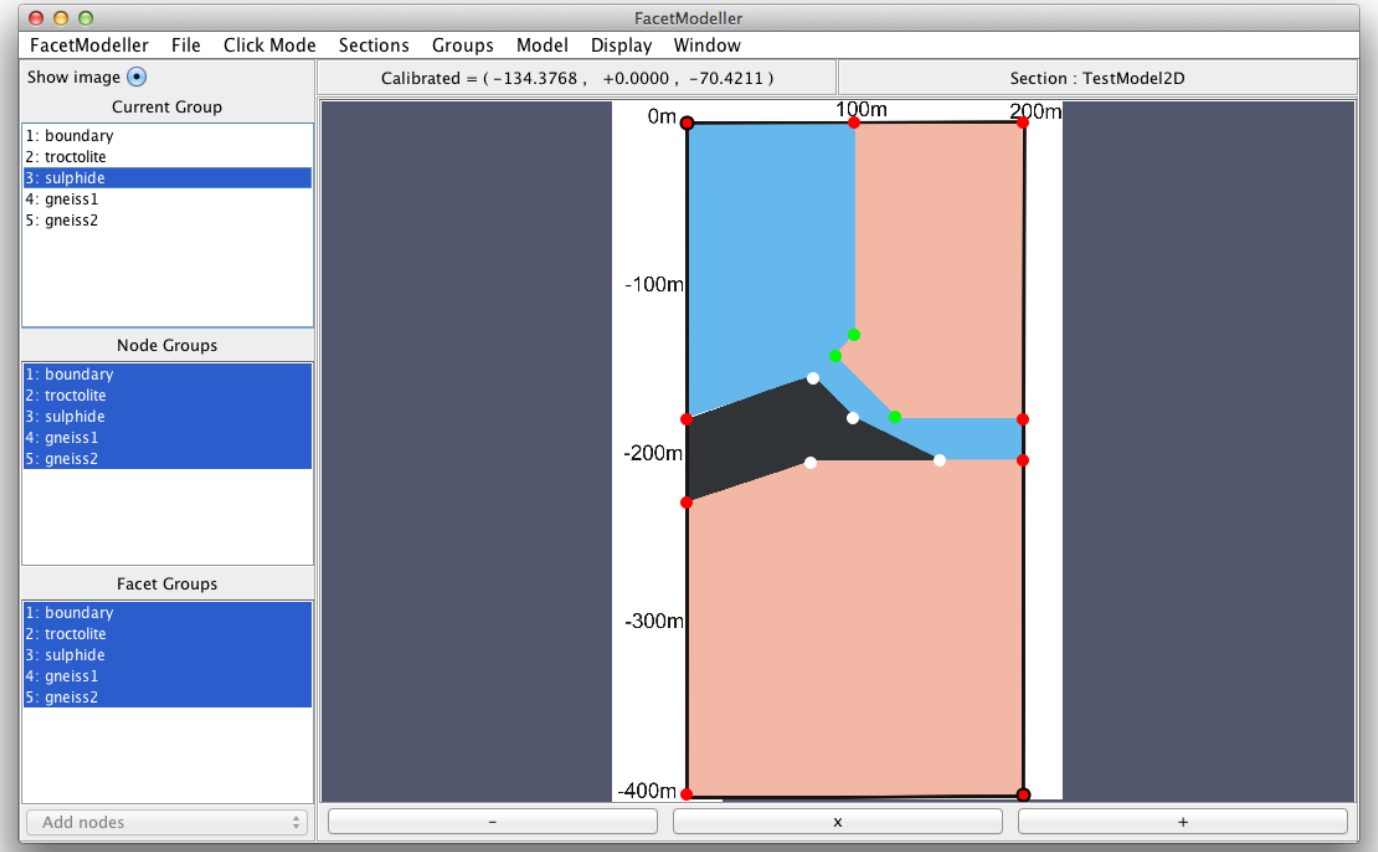
Now snap those boundary nodes to the calibration points, so that the boundary nodes will be exactly at the calibrated boundary:

8. Select the menu item
   `Model > Snap nodes to calibration points`

To finish the node definitions:

9. Select the `troctolite` group in the `Current Group` selector box.
10. Click to create the 3 nodes required to define the contact between the blue troctolite unit and the pink gneiss unit at the top right of the image (these nodes will be drawn green).
11. Select the group named `sulphide`.
12. Click to create the 4 nodes required to define the contact between the dark grey sulphide unit and the other units (these nodes will be drawn white).

Your model should now look like this:



> In the `Click Mode` menu you will see many different options that determine how to process left mouse button clicks. You will be introduced to many click modes in this tutorial. Be sure not to click in the image viewing panel by accident. You

can always select the `Ignore` click mode as a safety precaution (all mouse clicks are then ignored). The current click mode is displayed at the bottom left of the FacetModeller window.

10.7 Defining facets

You can now define facets. In 2D models, facets are line elements that connect two nodes. You will be using two different approaches to define new facets.

1. Select the menu item
   `Click Mode > Define facets node-by-node`
2. Select the group named `boundary` as the `Current Group` (as for nodes, any new facets will be added to the currently selected group).
3. Click on the first node on a facet, followed by the second. The new facet will now have been created and drawn red. Continue to define all 9 facets around the boundary of the model (refer to the next screen-shot for guidance). Remember that each facet in this 2D model contains only two nodes! Hence, for each facet you need to click on two different nodes. This means that, as you move around the boundary of the model, you may be clicking on the same node twice because it represents the second node in one facet and the first node in the next.

In this node-by-node click mode, you must have your cursor fairly close to the nodes in order to select them. When close enough, the node will be indicated by a white circle. The first node you click on will turn black, although that black circle will be hidden by the white "closest node" circle until you move your cursor away from the node. Once you click on the second node, the facet will be created and drawn.

Note the smaller black dots drawn in the centre of the facets: these are used as handles to the facets if you want to perform other processing with them, e.g. if you want to delete a facet or change the group membership of a facet.

If you make a mistake, just delete the facets you don't want (menu item `Click Mode > Delete facets`) and continue.

You will now define the facets for the contact between the blue troctolite and pink gneiss units. You will do so using a different click mode:

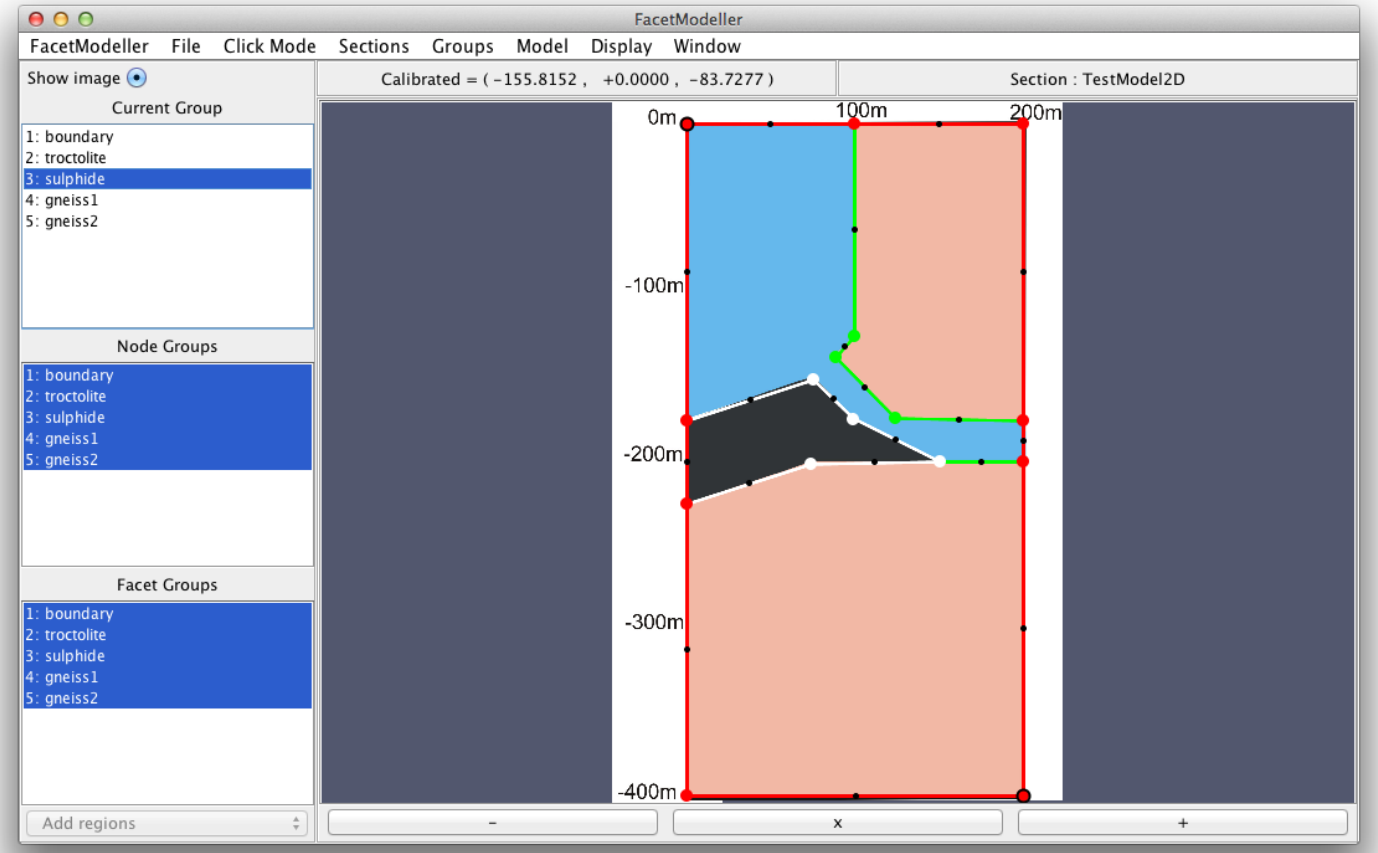
4. Select the menu item
   `Click Mode > Define linear edge facets`
   In this click mode, candidate facets are indicated as you move your cursor around on the image.
5. Select the group named `troctolite`.
6. Click when you see a facet that you wish to create. The new facet will be drawn green.
7. Continue defining the 5 facets for the troctolite-gneiss contact (there are 4 facets around the upper part of the pink gneiss unit and one facet below that connects to the dark grey sulphide unit).

As you move your cursor around in the linear edge facet click mode, candidate facets are indicated: black circles are drawn around a pair of nodes and a white dashed line is drawn between them.

The approach that the code takes to find a candidate node is a little complicated. It looks for the nodes within a particular distance from the cursor location and calculates all the possible node pairs; it then displays the facet with centroid closest to the cursor location. Hence, the candidate facet may jump around a lot as you move your cursor. The menu item `Display > Facet selection factor` allows you to change a special factor that affects the distance from the cursor location within which to search for nodes. You can increase or decrease this factor to include more or fewer nodes near the cursor location. You shouldn't decrease this factor below 1.0, and the higher it gets, the more nodes you'll catch. If the factor is set too high it may suggest more candidate facets than you would like. In fact, I have limited the factor to a maximum value of 2.0, but I can change that if needed.

Clearly, defining facets using this click mode is a little fiddly behind the scenes and it can be difficult to get exactly the facets that you want. The node-by-node approach is then available to you, but of course defining facets with that approach takes longer so I suggest use of the linear edge approach whenever possible.

To finish the model PSLG you need to define the 5 facets surrounding the dark grey sulphide body. Follow the steps above, using whatever click mode you prefer, but use the `sulphide` group. Your completed model PSLG should look like this:



To view your model without the image displayed, unselect the radio button at the top left of the window labelled `Show image`. The outline of the image is then displayed instead of the image itself. This can be helpful when you have a more complicated image that might hinder you when defining the facets in your model.

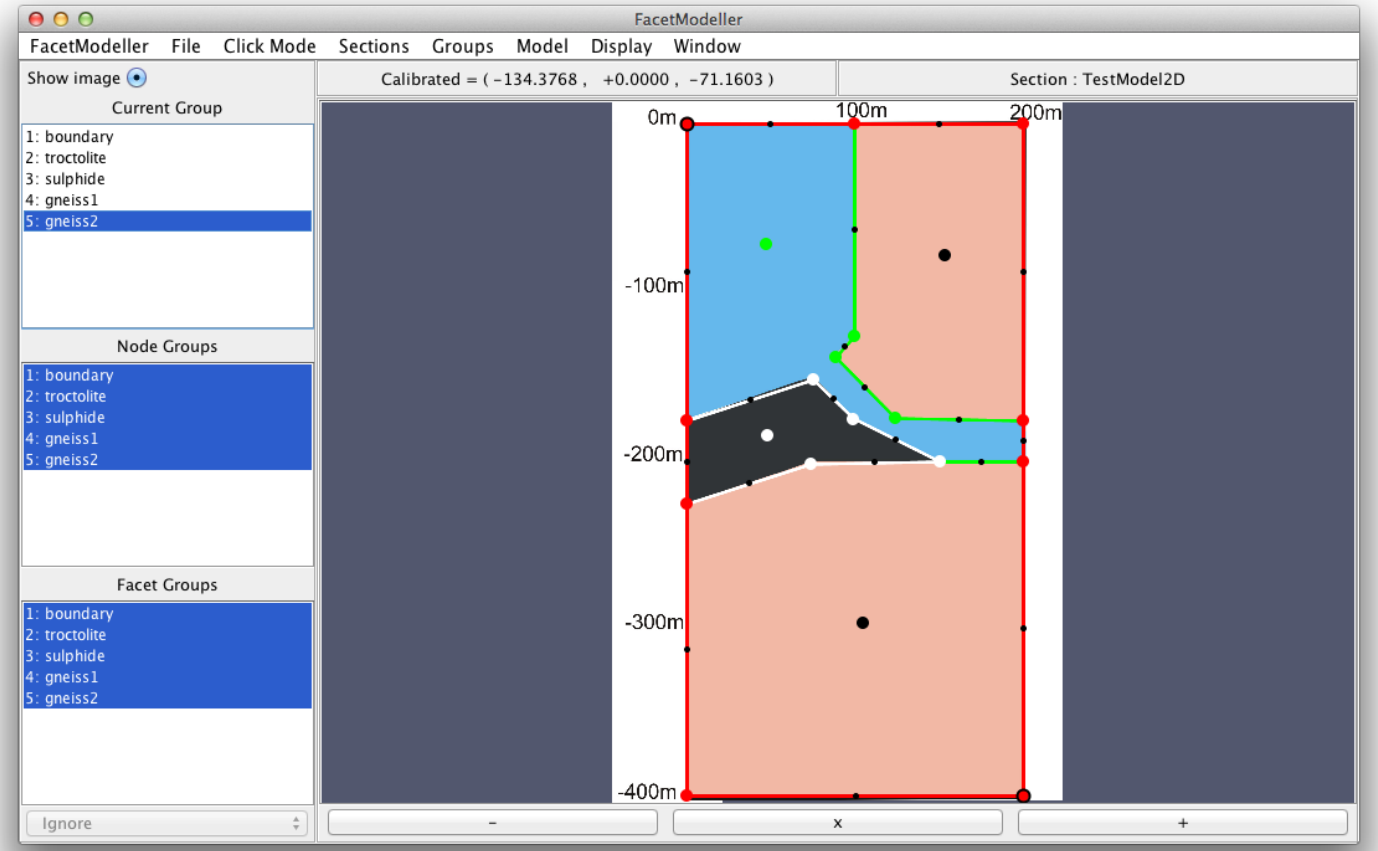
10.8 Defining regions

Now that the PSLG is defined, you need to define the different regions within it so that different attributes, e.g. physical properties, can be associated with the different parts of the model. Each watertight part of the model is a different region and you must provide a spatial location inside each of those different regions. FacetModeller stores each of those region specification points as a special kind of node and defining the regions is similar to defining nodes.

1. Select the menu item
   `Click Mode > Add regions`
2. Select the group named `troctolite` as the `Current Group`.
3. Click anywhere inside the blue troctolite unit.
4. When asked what type of point it is, choose the `Region` option. Always select the "Region" option (the "Control" option is something I need for my research).
5. Select the group named `sulphide`.
6. Click anywhere inside the dark grey sulphide unit (again, always select the "Region" option when prompted).
7. Select the group named `gneiss1`.
8. Click anywhere inside the **top** part of the pink gneiss unit.

9. Select the group named `gneiss2`.
10. Click anywhere inside the **bottom** part of the pink gneiss unit.
11. Select the menu item
    `Click Mode > Ignore`
    to tell FacetModeller to ignore any further mouse clicks (a safety precaution).

Your model should now look like this:



10.9 Exporting your model

The `File` menu provides options to export your model to various formats (`.poly`, `.node`, `.ele` and `.vtu`). Those file formats are explained on these webpages:

- Triangle: 2D triangular 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#sec69
- VTK File Formats
  http://www.vtk.org/VTK/img/file-formats.pdf

Once those files are exported, you will be able to use them with the Triangle meshing program and the ParaView visualization program.

A typical coordinate system for geological or geophysical data is

(X,Y,Z) = (easting, northing, elevation)

which is what FacetModeller uses. Others prefer to use

(X,Y,Z) = (north, east, down)

and if that is your preference then you are able to convert when exporting your model by choosing `Yes` when asked if you want to flip the z-axis: (X,Y,Z) is mapped to (X',Y',Z')=(Y,X,-Z).

10.10 Quitting FacetModeller

You can now quit your FacetModeller session by selecting the menu item

```
FacetModeller > Exit
```

or simply close the window as you normally would on whatever operating system you are using.

FacetModeller works in fundamentally different ways depending on whether you are working with a 2D or 3D model. Hence, you need to exit and relaunch the application if you want to start working with a new model of a different dimensionality.

Chapter 11

Tutorial - Building 3D models: I

A single isolated body within a boundary, digitized using cross-section images

This 3D tutorial relies on lessons learned in the 2D tutorial. You should also refer to the 2D tutorial for basic information about FacetModeller, including installation and running the program.

This first 3D tutorial does not contain any topography, nor do any of the anomalous rock units outcrop. Dealing with topography and outcropping is covered in the second and third 3D tutorials.

11.1 Running FacetModeller

Launch FacetModeller and choose `3D` when asked what kind of model you want to create. You will notice that the 3D version of FacetModeller looks a little different from the 2D version, with more panels, buttons and selection boxes (these will be explained throughout the tutorial).

11.2 Opening your image files

When building a 3D model, all the nodes in the model are associated with a particular depth- or cross-section (currently only vertical cross-sections are supported). The basic idea is to define all these sections, then add nodes to them, then create facets that connect the nodes.

The sections can be created by digitizing images or can be defined by the user without the need for an image. Here you will do the former. Select the menu item

```
File > Load cross section image files
```

and select the image files you want to digitize. For this example, you still start with 5 files named `x*.png`. Those in my research group can find these files here

```
[repos]/peter/NetBeansProjects/examples/FacetModeller3Da/
```

and public users here

```
[repos2]/facetmodeller/Tutorial_3D_Part1/
```

The other image files will be used later in the tutorial. Select all 5 files at once named `x*.png` and click `Open` (or whatever button your operating system gives you on the file open dialog window).

11.3 Defining the volume of interest (VOI)

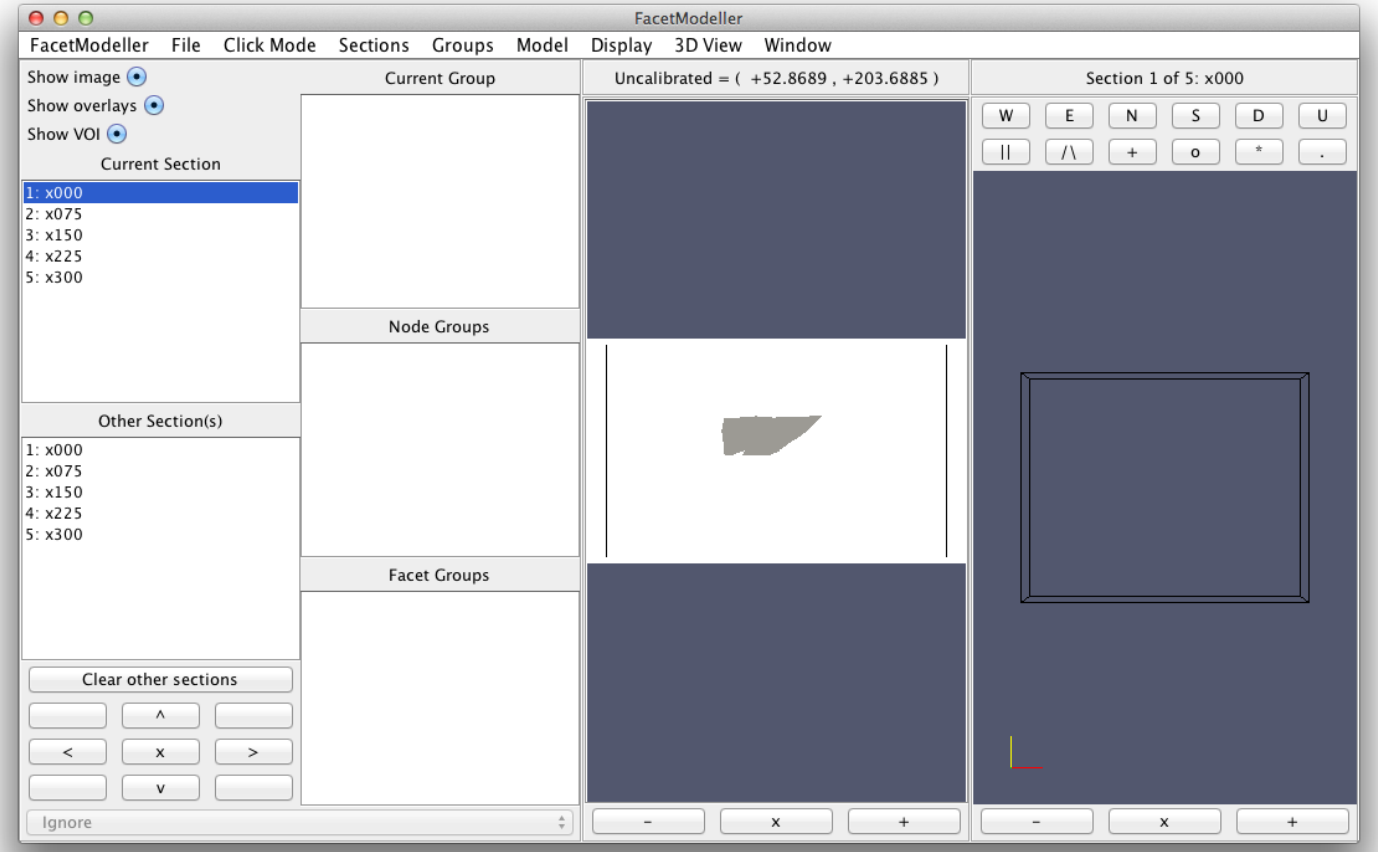
Although this step isn't required for general model building, defining the VOI can be helpful for 3D visualization, e.g. to help you make sure all of the parts of the model are inside your VOI. There are file menu items that help you make sure that all the nodes and facets you create are inside the VOI. To define the VOI, select the menu item

```
Model > Define volume of interest (VOI)
```

and enter the following information in the dialogs as they pop up:

Easting limits:	`-100 400`
Northing limits:	`-200 200`
Elevation limits:	`-100 150`

You should now see the following:



Remember to save your session early and often! I won't be reminding you to do so.

Again, keep in mind that this program is under development and this is a Java program that may look different on different platforms, so what you see may be slightly different than what is shown here.

The 3D version of the FacetModeller window is split into three main panels. On the left are various buttons and selection boxes that allow you to select which objects you wish to display and work with. On the right are two viewing panels; the panel on the left is a 2D viewing panel and on the right is a 3D viewing panel. The 2D viewing panel displays the currently selected section (in the `Current Section` selection box) and any nodes or facets defined on that section or between it any of the other sections (selected in the `Other Section(s)` selection box). The 2D viewing panel is where you will define nodes and facets (as in the 2D tutorial).

Only one section can be selected as the `Current Section`. Multiple selection is allowed for the `Other Section(s)` selection box. The selected `Current Section` defines the view limits and projection used for the 2D viewing panel. The selected `Other section(s)` defines what to plot on the 2D view. Nodes, facets and regions in the selected `Other section(s)` are projected onto the `Current Section` in the 2D viewing panel.

The 3D viewing panel displays the model in either parallel or perspective (the default) projection. You should currently see the VOI displayed in the 3D viewing panel. You can move the model around in the 3D viewing panel by dragging it with your mouse, but you can not interact further with the model in the 3D viewing panel, e.g. to add nodes or define facets. I will explain various features of the FacetModeller window throughout this tutorial, but may not cover everything: hopefully anything I miss will be self explanatory, or you can just play with it to see what it does.

11.4 Defining groups

There are two units in this example:

1. host rock or "background" (white in the images)
2. anomalous body or "blob" (grey in the images)

You'll need to define new groups (menu item `Groups > New group`) with names and colours indicated in the table below (you can use whatever colours you like but I'll be referring to them in the text so I suggest you stick with these):

Name	Colour
boundary	green
blob	red

11.5 Calibrating the images

The next step is to calibrate your images. You will do this in the 2D viewing panel and the process is similar to that in the 2D tutorial. To make this a little easier, you should hide the 3D viewing panel:

1. Select the menu item
   `3D View > Show/hide 3D view`

To perform the calibration of the first section:

2. Select the `x000` section in the `Current Section` selection box.
3. Select the menu item
   `Sections > Calibrate current section`
4. Click on the **top left** of the black axes in the image.
5. Enter `"0 200 150"` when prompted for the coordinates of that first calibration point.
6. Click on the **bottom right** of the black axes in the image.
7. Enter `"0 -200 -100"` when prompted for the coordinates of that second calibration point.

This vertical section is normal to the x (easting) direction, looking east (with north to the left, south to the right).

> When calibrating the first section, you should see a black outline in the image indicating some axes. If you don't see that black outline, you may need to maximize the window or use the zoom button below the 2D viewing panel to zoom in (sorry, this black outline is very thin and may not be visible in the screen-shots shown here).

This calibration procedure would normally be required for each section. However, in this example the sections are stacked along the x (easting) direction so you can copy the calibration information from the current section to all the others below it in the list:

8. Select the menu item
   `Sections > Copy calibration`
9. Enter `75` when prompted for the step length defining the x separation between the sections.

If you now select any of the other `x*` sections in the `Current Section` selection box, you will see calibration points drawn on each image as small black unfilled circles. To check the locations of the sections, move your cursor around on the 2D image viewing panel: the coordinates at the cursor location are displayed above that panel.

If you now

10. open the 3D viewing panel and
11. select all the sections in the `Other Section(s)` selection box,

you will see the outlines of the 5 sections drawn black in the 3D view. Before continuing:

12. Make sure the 3D viewing panel is closed.
13. Click the `Clear other sections` button (near the bottom left of the FacetModeller window).

11.6 Adding nodes

You may first want to change the point size and line width display options in the `Display` menu. As for the 2D tutorial, you will define new nodes in the 2D viewing panel, and you will start by defining nodes on the boundary:

1. Select the menu item
   `Click Mode > Add nodes`
2. Select the `boundary` group in the `Current Group` selection box.
3. Select the `x000` section in the `Current Section` selection box.
4. Click on the four corners around the boundary on the image in the 2D viewing panel.
5. Repeat for every section.

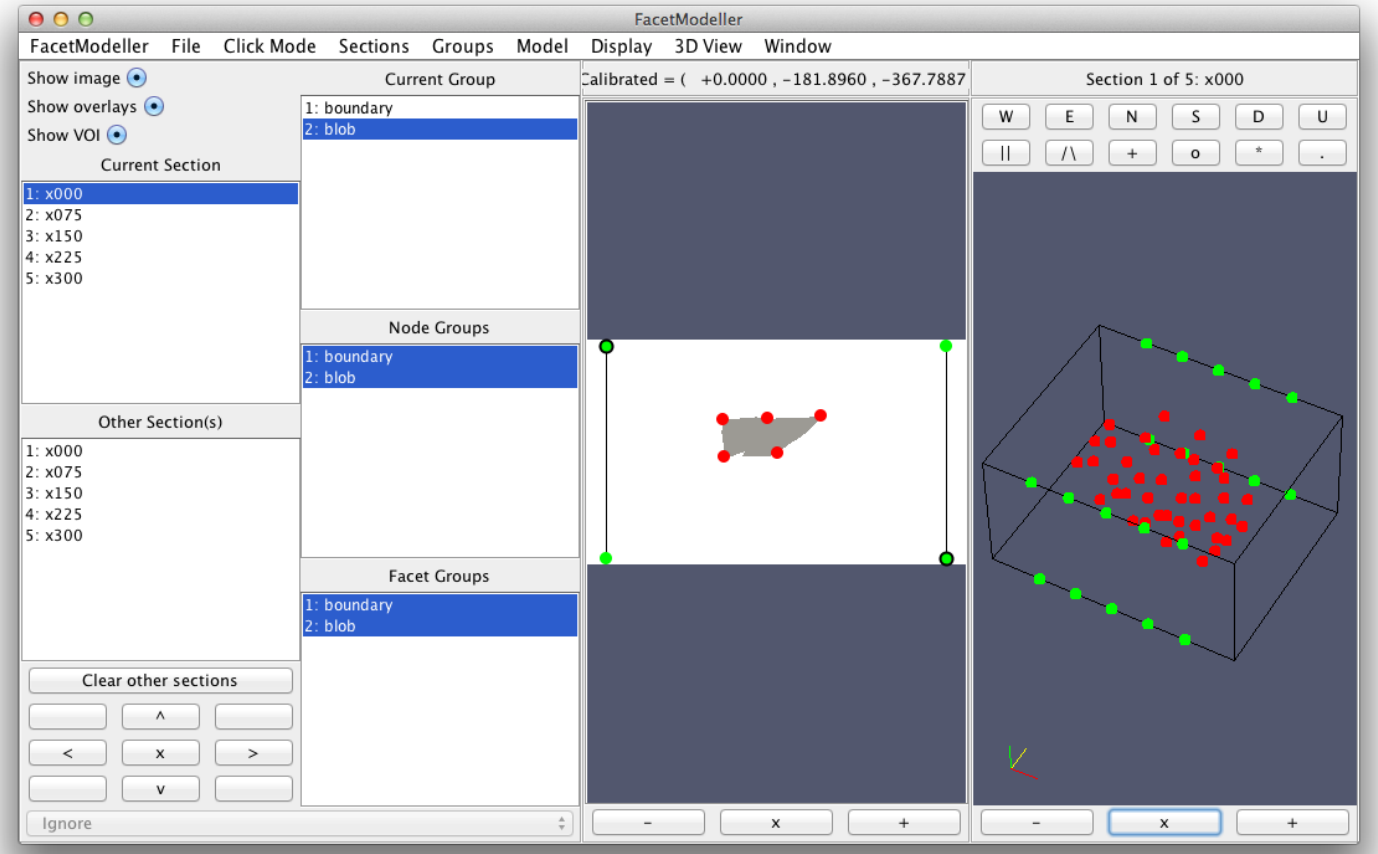
You need to snap those boundary nodes to the calibration points so that the polygonal boundary facets you create later are all planar.

6. Select the menu item
   `Model > Snap nodes to calibration points`

Now you will create nodes to define the outline of the blob:

7. Select the `blob` group as the `Current Group`
8. Select the `x000` section as the `Current Section`.
9. Click around the outline of the blob. Do this fairly coarsely (refer to the next screen-shot for guidance). You don't need to be too exact.
10. Repeat for every section.

Show the 3D view again and your model should now look like this:



If you haven't investigated the 3D viewing panel yet, now would be a good time to do so. There are zoom buttons on the bottom of the 3D viewing panel (as well as on the 2D viewing panel). The buttons on the top of the 3D viewing panel allow you to change the view direction, viewing options, etc. If you hover your mouse over those buttons, some text will pop up telling you what they do.

11.7 Defining facets between parallel sections

Remember that in 3D models, facets are triangles or other planar polygonal shapes. The first facets you will create are those on the surface of the blob:

1. Select the menu item
   `Click Mode > Define triangular facets`
2. Select the `blob` group as the `Current Group`.
3. Make sure that only the `blob` group is selected in the `Node Groups` selection box (this isn't required, but it is nice to hide any nodes that you don't need to work with).
4. Select the `x000` section as the `Current Section`.
5. Select the `x075` section in the `Other Section(s)` selection box.

In the 2D viewing panel, you should now see the nodes for the `x000` section drawn as filled circles and those for the `x075` section drawn as empty circles (nodes on the `Current Section` are always drawn filled and those on the `Other Section(s)` are always drawn empty). You will need to hide the image to help you continue:

6. Unselect the radio button `Show image` (at the top left of the FacetModeller window).
7. Move your cursor around on the 2D viewing panel to see candidate triangular facets. The candidate facets are also displayed in the 3D viewing panel so rotate and zoom the 3D view as required to help you with this task. You will also need to shift the overlaid nodes on the 2D view (see instructions below).
8. Click once you see a facet you want to create.
9. Continue clicking to define new facets between the sections.

The nodes for the current and other sections overlap along the top of the blob and it is difficult to figure out which to connect. To help with this, use the 3-by-3 grid of buttons at the bottom left of the window (under the `Clear other sections` button). These 9 buttons allow you to shift the overlaid nodes from the other section: click on any of the outer 8 buttons to move in that direction (up $\boxed{\wedge}$, down $\boxed{\vee}$, left $\boxed{<}$, right $\boxed{>}$, diagonally $\boxed{}$) and click the central $\boxed{\times}$ button to reset (no shifting).

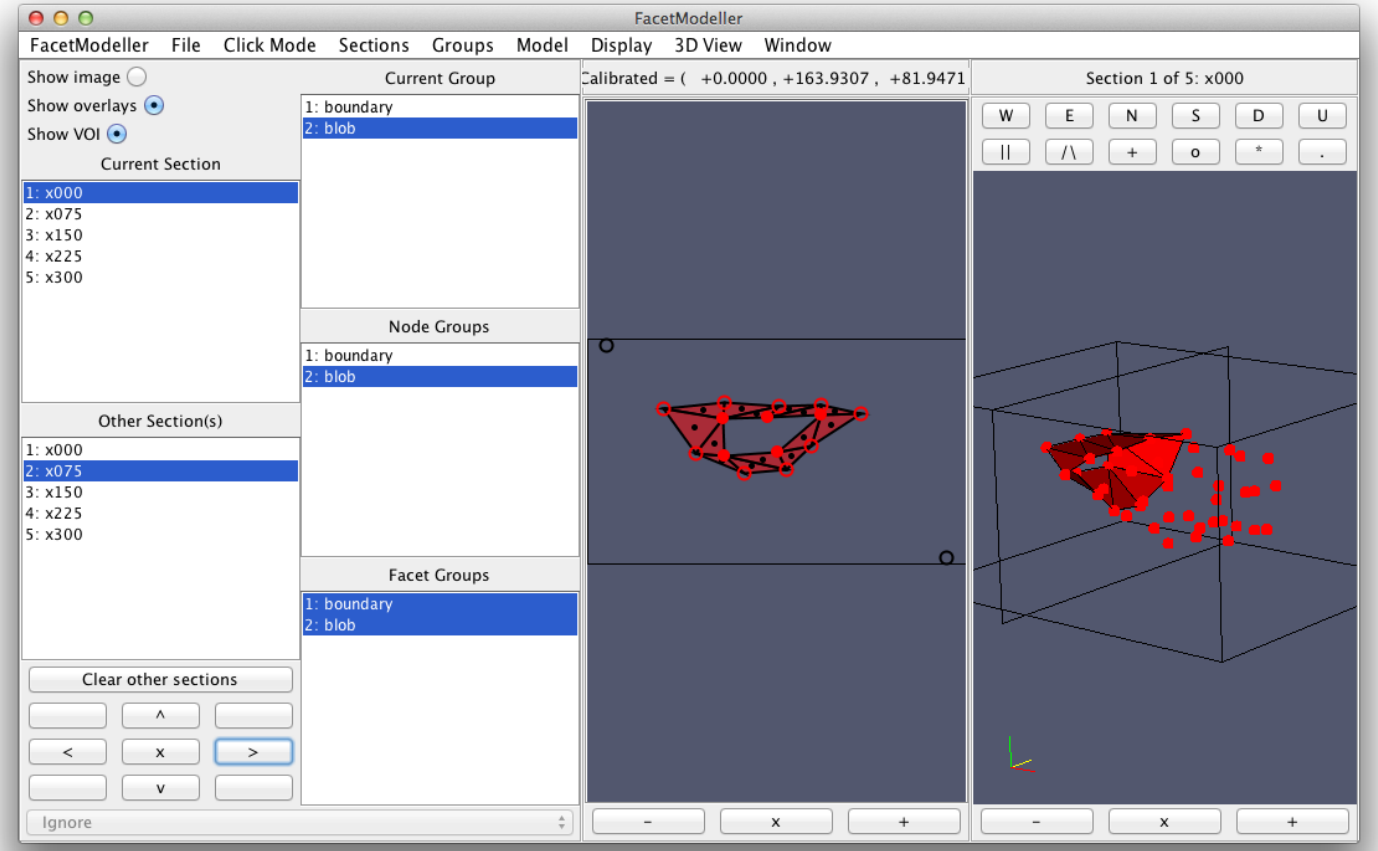
If you make a mistake, like creating overlapping facets, just delete the ones you don't want (menu item `Click Mode > Delete facets`) and continue.

Another helpful feature is the menu item

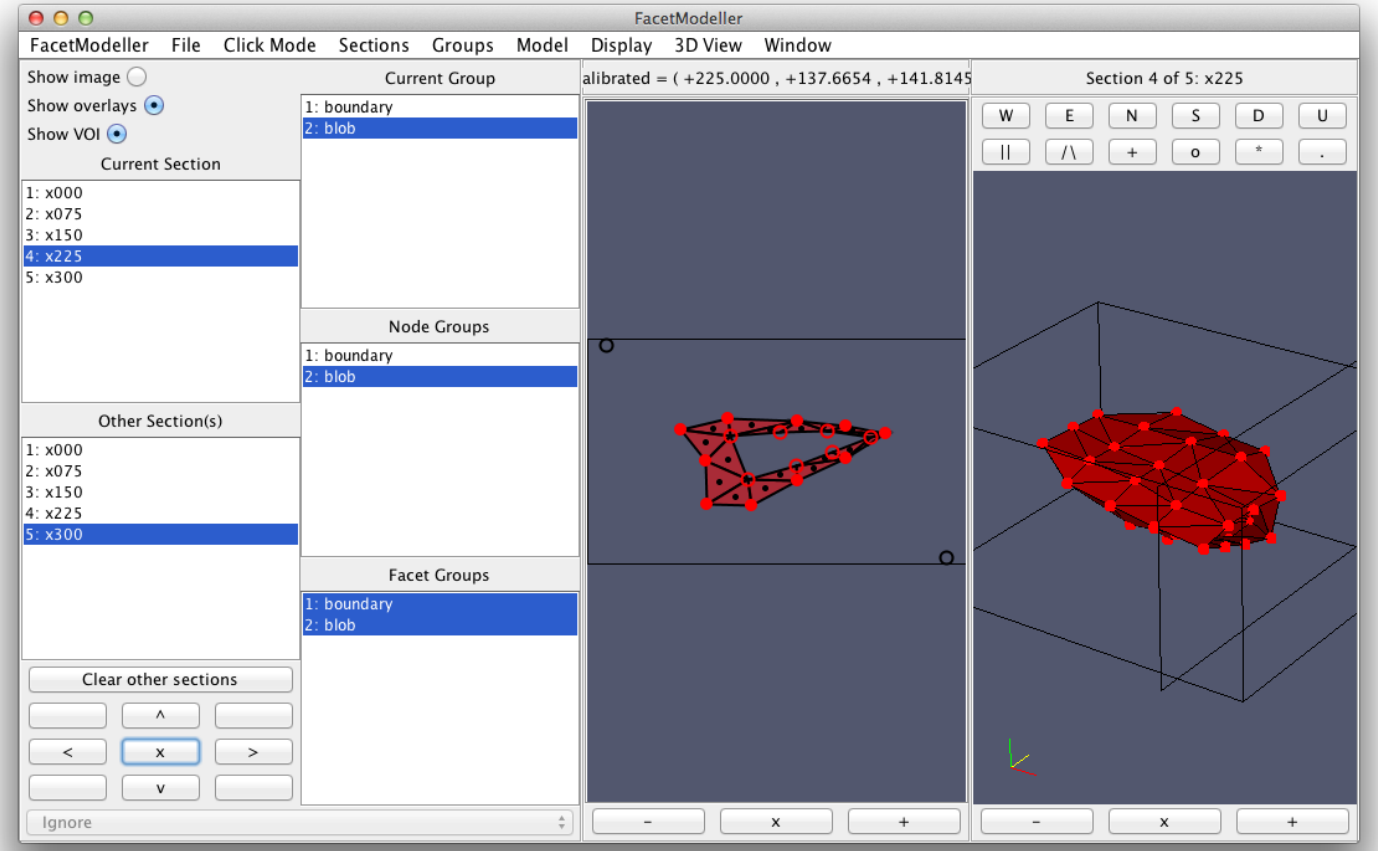
`3D View > Select 3D origin node`

which allows you to select a node to become the centre of the 3D viewing panel so that you can then zoom in on the appropriate part of the model.

Here is what you should see once you have finished this step with the first pair of sections:



You can now move on to the next pair of sections. This process takes a long time. I'd suggest you practice defining a few facets and then move on, but if you do manage to get through it, your model should now look like this:



You have yet to close the body at the far western and eastern ends. To do so:

10. Select the menu item
    `Click Mode > Define facets node-by-node`
11. Select the `x000` section as the current section.
12. Clear the overlaid other sections (the `Clear other sections` button).
13. Click to define new facets around the outline you see in the 2D view.
14. Repeat with the `x300` section to close the other end of the blob.

> In the node-by-node click mode, the facet is defined once you loop back to the same node you started with. Because all of these nodes are on the same section, the resulting facet will be planar. This is an important restriction for use in most meshing programs.
>
> Currently, FacetModeller does not check that polygonal facets with more than three nodes are planar. If you create such a facet and try running TetGen on the resulting PLC, TetGen will fail.

11.8 Defining facets between non-parallel sections

You will now load a section that is not parallel to the those already loaded:

1. Select the menu item
   `File > Load cross section image files`
2. Choose `Insert` when prompted.
3. Open the file `y00.png` (in the same location as you found the other image files).
4. Make sure the `Show image` radio button is selected.

Follow through the same calibration steps as before with this new section:

5. Make sure the `y00` section is the `Current Section`.
6. Select the menu item
   `Sections > Calibrate current section`
7. Click on the **top left** of the black axes in the image.
8. Enter `"-100 0 150"` when prompted for the coordinates of the first calibration point.
9. Click on the **bottom right** of the black axes in the image.
10. Enter `"400 0 -100"` when prompted for the coordinates of the second calibration point.

This vertical section is normal to the y (northing) direction, looking north (with west to the left, east to the right).

11. Define a new group (menu item `Groups > New group`) named `blob_y` with blue as the group colour.
12. Select the menu item
    `Click Mode > Add nodes`
13. Add nodes around this new blob outline.
14. Investigate your model in the 3D viewing panel. You should see the new blue nodes you added on the new section.

You'll now define new facets between these new blue nodes and the pre-existing red nodes. You'll focus only on those near the western extent. First you need to delete the facet you created recently on the western extent:

15. Select the `x000` section as the `Current Section`.
16. Select the menu item
    `Click Mode > Delete facets`
17. Click on the centroid handle of the displayed polygonal facet to delete it.

You now want to sew up the western extent of the blob, using the nodes on sections `x000` and `y00`.

18. Select the `x000` section as the `Current Section`.
19. Select the `y00` section as the `Other Section`.

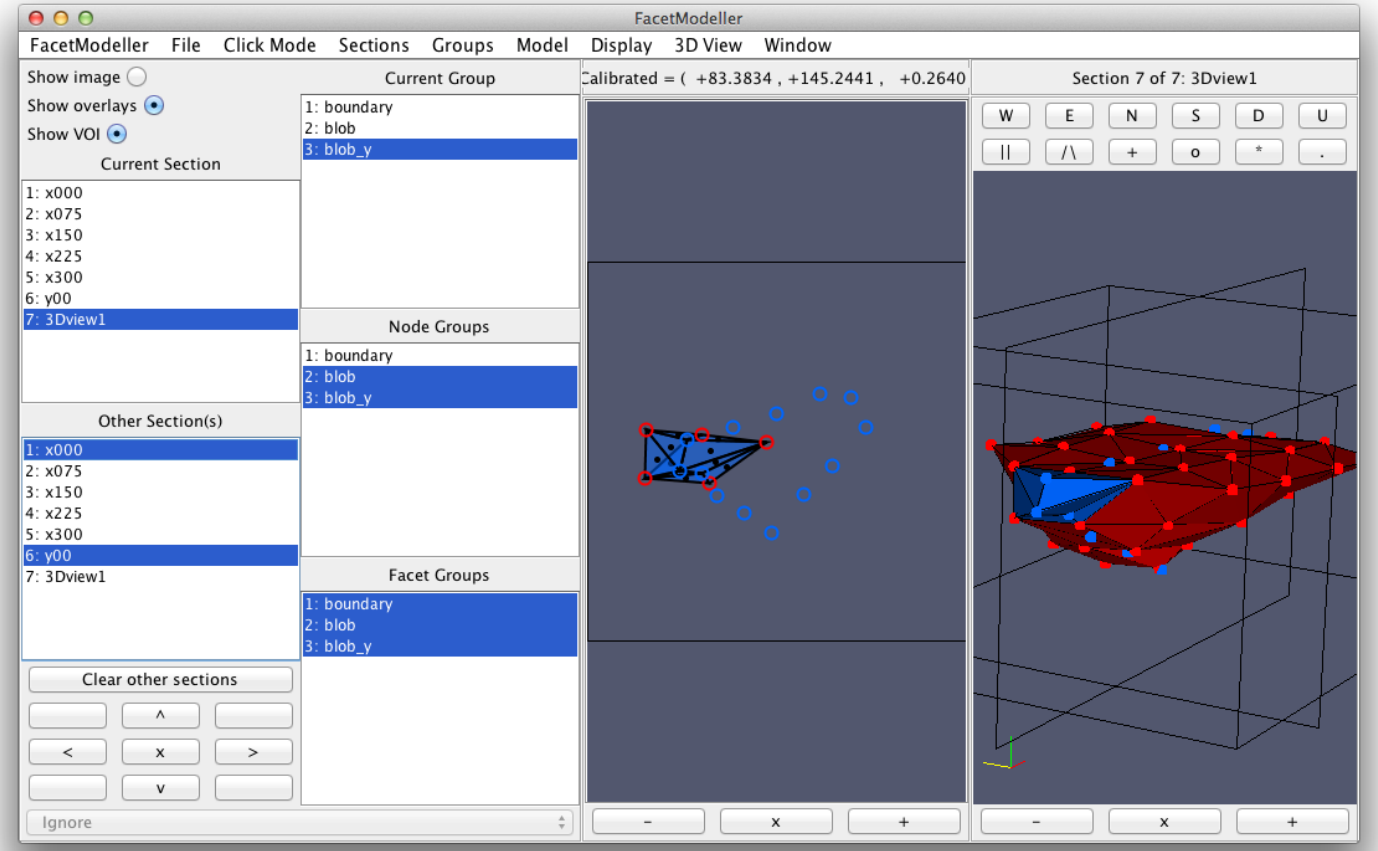
The projection in the 2D viewing panel will makes it practically impossible to define the facets required! You need another view in the 2D viewing panel. To get one:

20. Rotate the 3D viewing panel to the desired view and zoom as desired (refer to the next screen-shot for guidance).
21. Select the menu item
    `Sections > New section from current 3D view`
22. Name the section `3Dview1` when prompted.
23. Use the default white as the colour for the section when prompted.
24. Make sure this new section is selected as the `CurrentSection`.
25. Select the `x000` and `y00` sections as the `OtherSections`.
26. Make sure you have cleared any previous shifting by clicking the $\boxed{\times}$ button in the centre of the 3-by-3 grid of shift buttons at the bottom left of the window.

The 2D view should now allow you to sew up the western end of the blob with triangular facets. If that view is still not helpful, change the 3D view and use the menu item `Sections > Reset snapshot section` to reset the view for the `3Dview1` section.

27. Select the `blob_y` group as the `Current Group` (the facets will be drawn blue to help distinguish them from those already created).
28. Select the menu item
    `Click Mode > Define facets node-by-node` (you'll probably find this simpler than the other approach for defining triangular facets but you can try both if you like).
29. Click to define triangular (and therefore planar) facets.

Your model should now look like this:



11.9 Defining facets on the boundary

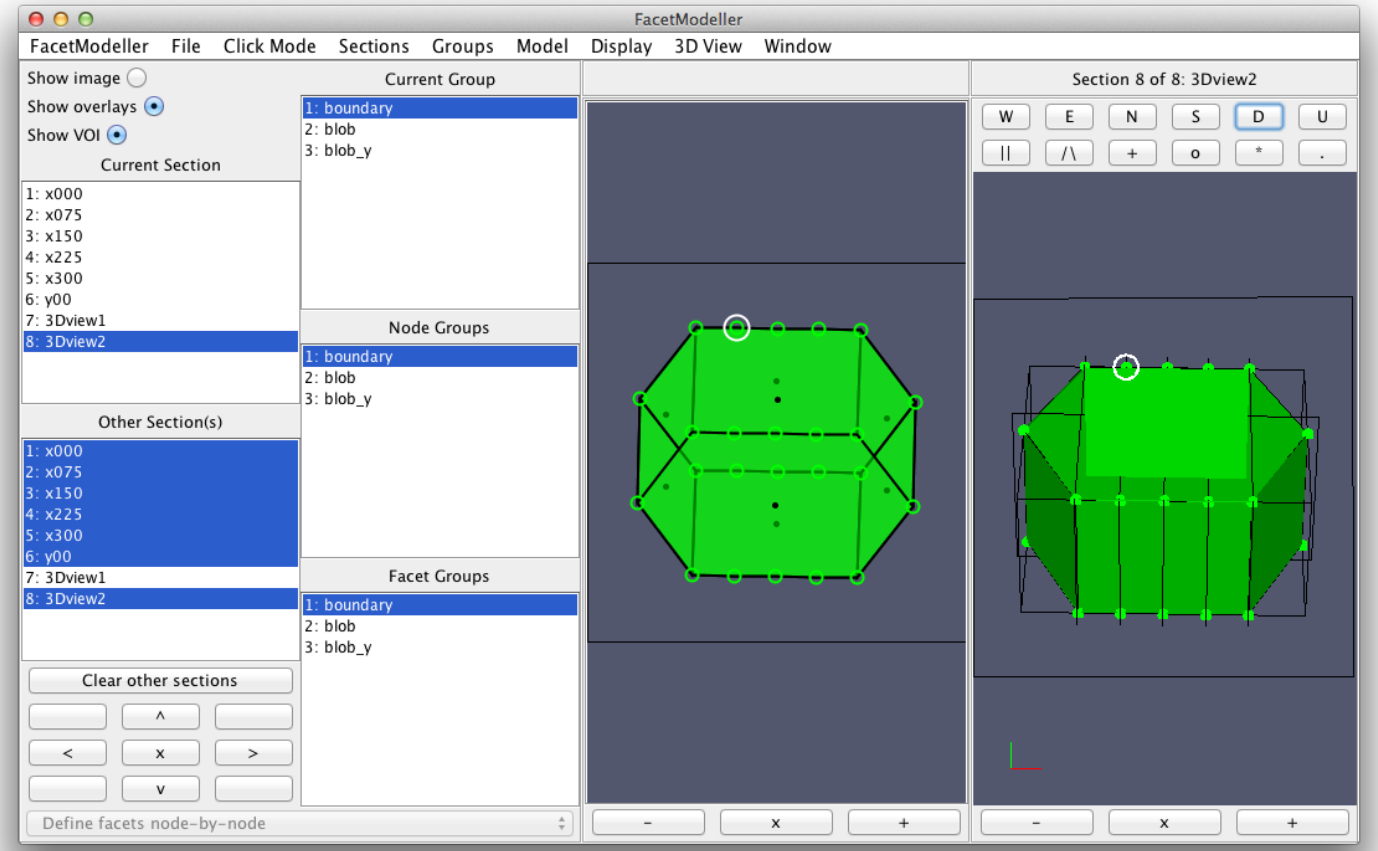
The last thing you need to do is create facets around the boundary of the model. You will need some new nodes and another view to help you:

1. Select the menu item
   `Click Mode > Add nodes`
2. Select the `boundary` group as the `Current Group`.
3. Select the new `y00` section as the `Current Section`.
4. Add four new boundary nodes at the corners of the `y00` section. You will have to click very close to the lateral extents of the 2D viewer panel.
5. Select the menu item
   `Model > Snap nodes to calibration points`
6. Select the menu item
   `Click Mode > Define facets node-by-node`
   (the node-by-node approach is the only way to define polygonal facets with more than 3 nodes).
7. Display only the nodes and facets in the `boundary` group using the `Node Groups` and `Facet Groups` selection boxes (initially there are no such facets and you are just hiding all the others).
8. Clear the zoom on the 3D view and rotate the model so that none of the boundary nodes are overlapping (refer to the next screen-shot for guidance).
9. Create a new section from the current 3D view and name it `3Dview2`.
10. Make sure that new section is selected as the `Current Section`.
11. Select all the $x*$ sections and the $y00$ section as the `Other Sections`.

12. Define eight planar polygonal facets to define a boat-like boundary surface:
    - there are rectangular facets on the southern and northern sides,
    - rectangular facets form the bow and stern of the boat-like shape, 2 at each of the western and eastern extents,
    - and hexagonal facets define the top and bottom.

    Refer to the next screen-shot for guidance.

Your model should now look like this:



Perhaps you are uncomfortable with a boundary that is not a rectangular prism? Here is how to make one. First delete all the facets on the boundary that you just created:

13. Select the `boundary` group ias the `Current Group`.
14. Select the menu item
    `Model > Delete group of facets` and poof, they are gone. Actually, you don't even need any of the existing boundary nodes ...
15. Select the menu item
    `Model > Delete group of nodes`

You now need 2 more sections to add nodes to the western and eastern extents of the VOI. You will generate 2 sections without images. They will be drawn as rectangles with some user-defined colours and they can be treated in the same way as the sections used above, e.g. they need to be spatially calibrated and you can add nodes on them. To create these new sections:

16. Select the menu item
    `Sections > New cross section without image`
17. When prompted, name the section `south` and choose the default white as the colour.
18. Enter `"-100 -200 150"` when prompted for the top left corner of the image.
19. Enter `"400 -200 -100"` when prompted for the bottom right corner of the image.

20. Repeat those steps for a new section named `north`, with calibration points `"-100 200 150"` and `"400 200 -100"`.

Now all you need to do is define 4 new `boundary` nodes on the corners of the `south` and `north` sections (8 new nodes in total), snap them to the VOI, then define the 6 `boundary` facets node-by-node.

> Alternatively, you could have used sections on the southern and northern extents of the VOI to create this rectangular prism boundary. You could also define two triangular facets on each edge of the VOI instead of rectangular facets. Since this is easily automated, I plan to provide a menu item to automate the process.

11.10 Defining regions

Once a PLC is been built, destined for input into TetGen, you may or may not want to define regions. TetGen's `-A` flag can be used to assign cell attributes to indicate the cells in different watertight volumes in the mesh generated by TetGen. If you want to specify region-based constraints on the mesh, for example maximum cell volumes, then you will want to define your own regions to which the constraint parameters can be assigned.

There are two approaches you can take to define regions. The first takes the fewest steps but may not be as effective for more complicated models where it may be difficult visualizing exactly where your region points lie in relation to the PLC surfaces.

11.10.1 Defining regions on sections

In this approach, you create new sections on which to specify the regions. The steps are as follows:

1. Create as many new sections without images as are required to slice through the different watertight volumes in the PLC.
2. Add region points on those new sections.

11.10.2 Defining regions on facets

In this approach, you create new nodes that will function as the region points. You are not using the FacetModeller "region" objects at all; instead, you are using the FacetModeller "node" objects, which are essentially the same thing anyway. Those nodes are added on the facets of the PLC and are then shifted as required to move them into the watertight volumes in the PLC. For this strategy, care must be taken when exporting the different model components. The steps are as follows:

1. Create a new node group that will hold your region coordinates. For the purposes of these instructions I'll call that new node group `region_nodes`.
2. Select that new group as the current group and choose the "Add nodes in facets" click mode.
3. Define new nodes on facets that lie on the boundary of each watertight volume in the PLC.
4. Select the menu item
   `Model > Translate nodes`
5. Translate the `region_nodes` slightly, as required to move them into the different watertight volumes in the PLC.
6. It may be impossible to move all the `region_nodes` at once such that they all enter the different watertight volumes in the PLC. In this case, you would have to define new node groups, assign the region nodes into the various new node groups, and translate each group of region nodes separately.
7. Export the group(s) of region nodes to a `.node` file.
8. At this point, I like to delete all the region nodes in FacetModeller once I'm happy with them so that I don't mistakenly export them with the entire model into the `.poly` file in the next step. You can always load them back in from the `.node` file that you saved in the previous step.
9. Export all components of the model except for those region nodes to a `.poly` file.
10. Using whatever ASCII text file editor you like, manually paste the region coordinates from the `.node` file to the `.poly` file (the files that were exported in the previous steps).
11. Add any additional meshing constraint information in the list of regions in the `.poly` file, for example maximum cell volumes.

Chapter 12

Tutorial - Building 3D models: II

A padded VOI cut by topography

This 3D tutorial example relies on lessons learned in the 2D tutorial and first 3D tutorial. You should also refer to the 2D tutorial for basic information about FacetModeller, including installation and running the program.

In this tutorial, you will learn how to generate a simple model containing a padded VOI cut by a topography surface. I won't be providing explicit step-by-step instructions like in the other tutorials so I hope you remember how to do some of the basic tasks.

12.1 Requirements

This tutorial relies on two free utility codes. You will need to use the Triangle meshing program, which can be downloaded here:

http://www.cs.cmu.edu/~quake/triangle.html

Instructions for compiling the Triangle c++ source code are provided on that page.

You will also need to use my program `interpolate_data`.

12.2 Preliminary topography processing

You will need to perform the following processing on your topography data before working with it in FacetModeller. First you need to do some basic processing and get it into the correct file format. **In this tutorial, I have already processed the topography data as required so you can skip this section, but read it anyway!**

1. Remove points outside of the VOI (this can be done in FacetModeller or you can use the PODIUM program `remove_range`; see Section 12.3 below for VOI information).
2. Decimate the points so the spacing is nicer to work with (see aside below).
3. Get your topography data into the column-based ascii file format below (if you have been using the PODIUM programs to perform the topography processing then your file will already be in the required format).

> If the data is too dense then the resulting model may yield more cells than is computationally feasible for forward modelling or inversion. If the data is too sparse then the model may lead to unacceptably large errors in the forward or inverse solutions. Decimation is not currently possible in FacetModeller but you can use the PODIUM utility program `decimate`.

Your topography data will need to be in the column-based ascii file format below:

```
n [...]
1 x1 y1 z1 [...]
2 x2 y2 z2 [...]
...
n xn yn zn [...]
```

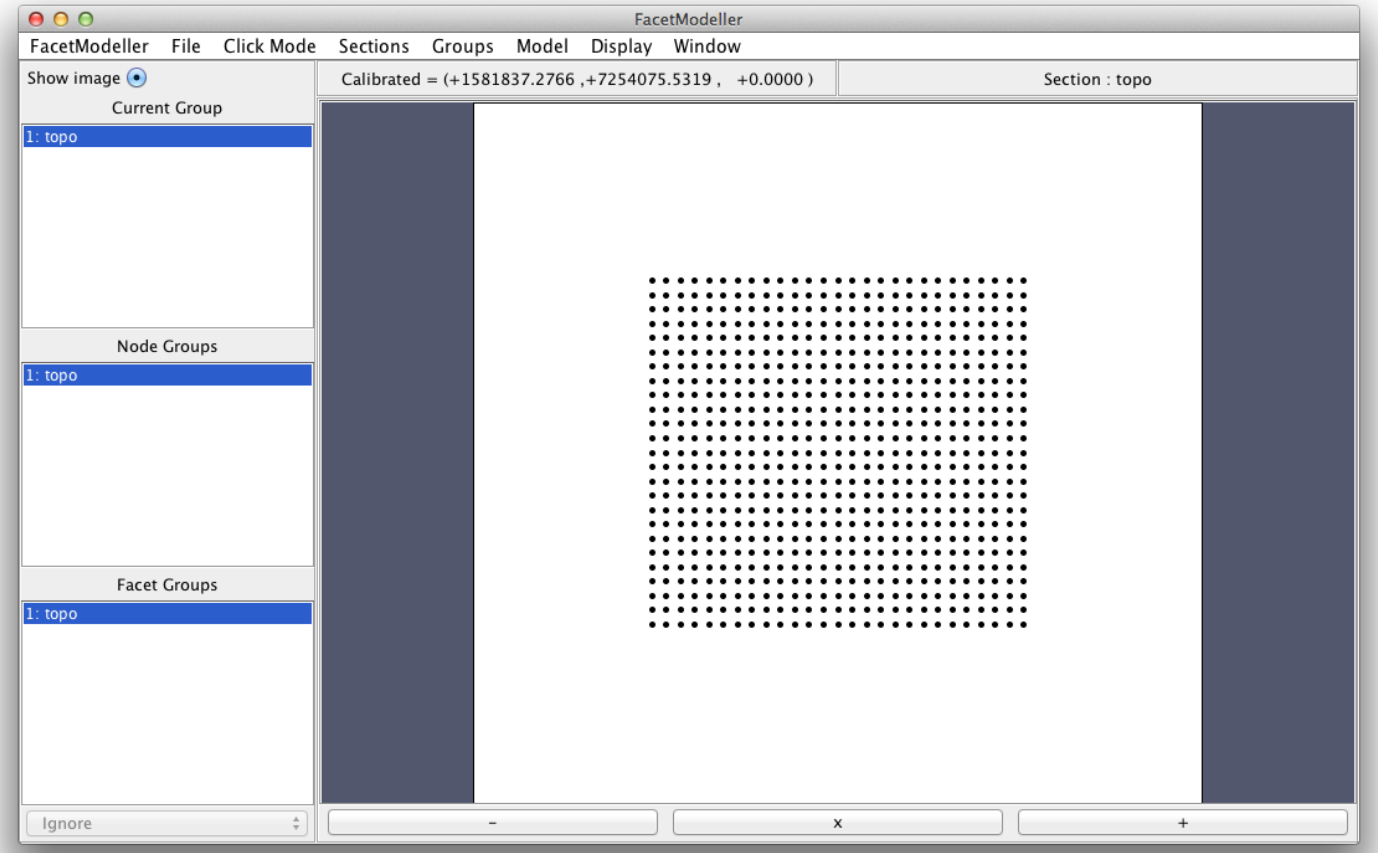
where `[...]` indicates optional additional values on a line of the file. This format is consistent with `.poly` and `.node` files in both 2D and 3D (see the file format pages for Triangle and TetGen).

12.3 Defining the padded VOI in 2D

You will be using Triangle to mesh your topography surface. First however, you need to define your padded VOI in 2D. Why 2D? That will become clear as you go through this tutorial, but the idea here is to use Triangle to generate a topography surface that conforms to the facets in your padded 3D VOI, and to do so you need to move into 2D.

1. Launch FacetModeller for building a 2D model.
2. Create a new group named `topo`, coloured the default black.
3. Select the menu item
   `Sections > New depth section without image`
   and call the section `topo`, coloured the default white.
4. When prompted for the coordinates of the bottom left corner of the image, enter `"1588400 7184850 0"`.
5. When prompted for the coordinates of the top right corner of the image, enter `"1670000 7263250 0"`.
6. Select the menu item
   `File > Load from .node/.ele files`
7. When prompted for the `.node` file, select the file `topography_decimated32.node`.
   Those in my research group can find this file here
   `[repos]/peter/NetBeansProjects/examples/FacetModeller3Db/`
   and public users here
   `[repos2]/facetmodeller/Tutorial_3D_Part2/`
8. When prompted for the `.ele` file, click `Cancel` (or whatever button your operating system gives you on the file open dialog window). This does not cancel the loading, rather it indicates that there is no `.ele` file associated with the `.node` file).

Don't forget to save early and often! You should now see the following:



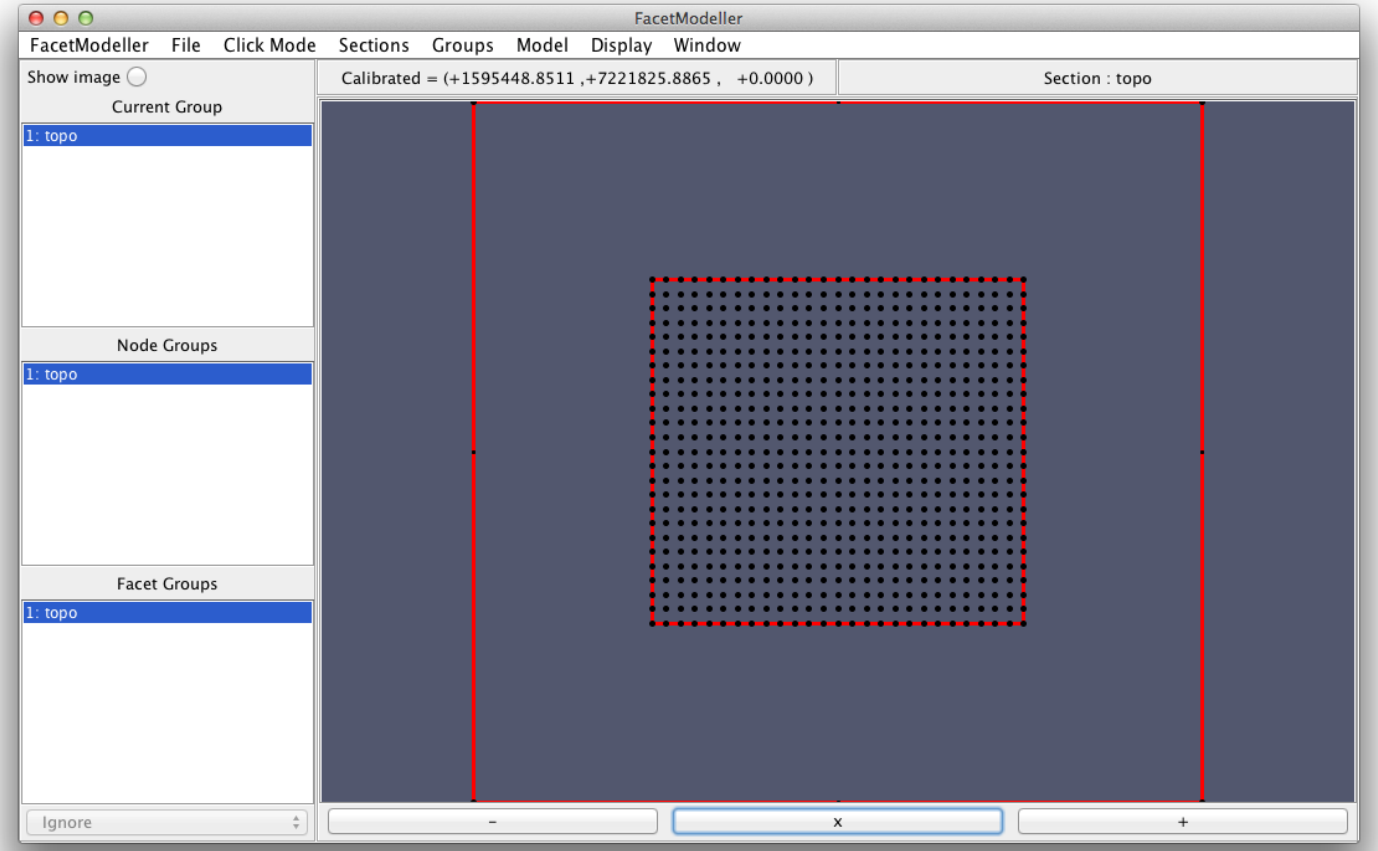
The black dots are the topography points, projected into map view. These points are gridded: you do not need gridded topography in general. You will now define the facets in the boundary of our padded VOI:

9. Add 4 new nodes on the corners of the section.
10. Snap those nodes to the calibration points.
11. Select the menu item
    `Groups > Change facet colour`
    and select red. This will make the facets show up more easily on the black-and-white backdrop.
12. Hide the section image (unselect the `Show image` radio button at top left) to help the facet overlays show up more easily in the following steps.
13. Select the menu item
    `Interaction > Picking/snapping distance` and change from the default `10.0` to `500.0`. This spatial distance (in whatever calibration units you are using; metres here for the UTM coordinates) defines how close the curser must be to a node before it can be selected, for example to attach it to a facet.

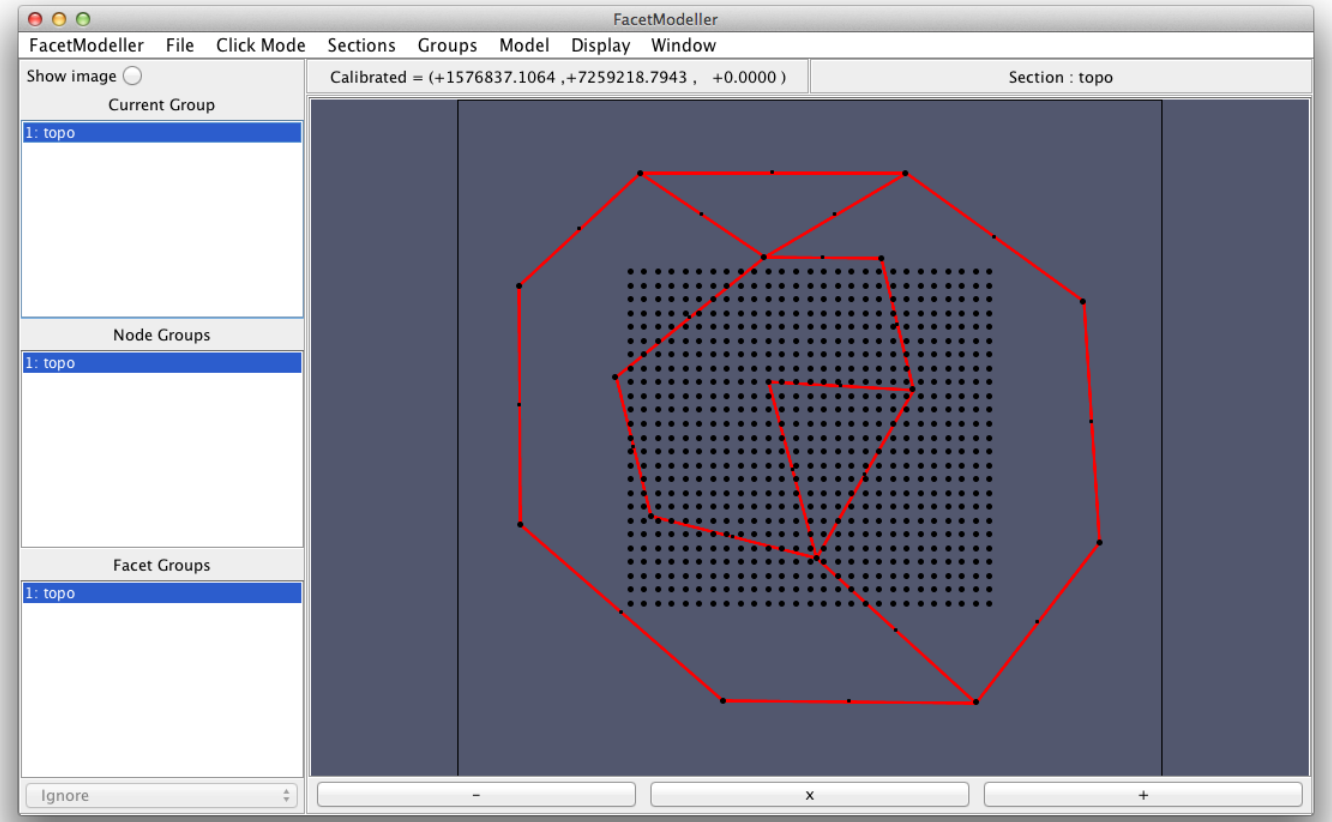
> This distance is also used when snapping nodes to, for example, the extents of the VOI. Having a smaller picking distance is advisable so that you are only selecting nodes within some small proximity of the cursor. You can tell when a node is selectable when a white circle is drawn around it, which will only happy once your cursor is close enough to it. Also, if the picking/snapping distance is too large then you may inadvertently snap nodes that you did not wish to, e.g. nodes that are not close to the VOI. Conversely, if the picking/snapping distance is too small compared to the spatial range of your model then the spatial distance corresponding to an image pixel may be larger than the picking/snapping distance. In that situation, FacetModeller will not respond to your mouse clicks and nothing will happen why you try to snap nodes.

14. Using the `Define facets node-by-node` click mode, define 4 new facets around the boundary of the section.
15. Define 4 new facets that connect the corners of the topography grid.

You should now see the following (I have changed the line display width here):



The outer red rectangle defines the boundary of the modelling region. The inner red rectangle defines a central region of interest where we want smaller cells in our modelling mesh. The space between the two red rectangles defines the padding region where we can allow larger cells. Generally, you will not have gridded topography, in which case, you would simply add new nodes anywhere you wish to define the inner rectangle. Of course, you could follow that strategy with gridded data too, and there is no reason that the central region of interest needs to conform to the topography. Also, there is no reason these need to be rectangles, and you could have facets that define as many different regions as you like. For example, you could also have done something like this:

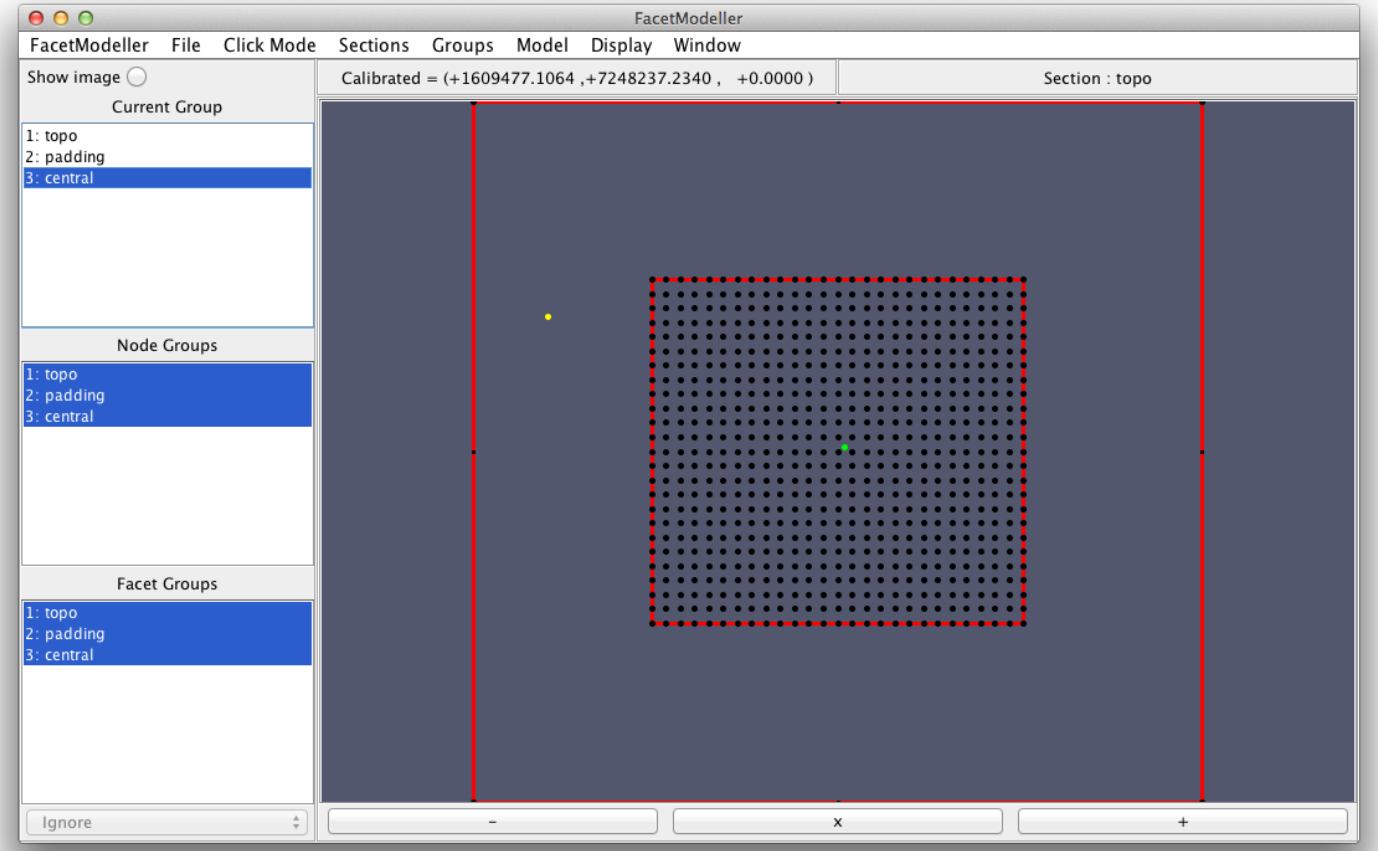


But let's stick to the simpler situation for now!

The final requirement is to add two regions to the model (if you forget how to define regions you can refer to Section 10.8):

16. Define a new group named `padding` and define a new region point anywhere inside the padding region (between the two red rectangles).
17. Define a new group named `central` and define a new region point anywhere inside the central region of interest (inside the inner red rectangle).

You should now see the following (I have coloured the two regions yellow and green):



The rest of the 2D processing goes as follows:

18. Export your model to a `.poly` file and call it `topo2d.poly`.
19. Quit FacetModeller.
20. Open the `topo2d.poly` file in your favourite text editor.
21. Scroll to the bottom and add area constraints for Triangle to use when meshing. The last two lines of the file should read something like this:
    `1 1596698.8936170214 7239201.773049645 1 1000000000`
    `2 1629894.4680851065 7224606.028368794 2 2560000`
22. Mesh the `.poly` file with the following commands:
    `triangle -pqDajPCV topo2d.poly`
    The result will be files named `topo2d.1.node` and `topo2d.1.ele` holding the triangulated 2D mesh.

> In this example you have specified an area constraint of `2560000` for the central region of interest. This corresponds to the area of the topography grid squares for this example (the grid spacing is 1600 = $\sqrt{2560000}$). The area constraint of `1000000000` is set so large that the constraint is essentially ignored by Triangle.

The final task is to interpolate the topography at the new nodes generated by Triangle. I have provided the output file `interpolated.node` in the tutorial directory if you want to skip these steps.

23. The header of the topo2d.1.node file needs to be changed to turn it into a 3D file, which you can do manually or us my program `node3d`:
    `node3d topo2d.1`
24. Then you interpolate the topography using: `interpolate_data topo_decimated32 null topo2d.1 interpolated.node 1 0`
    The output file is named `interpolated.node`.

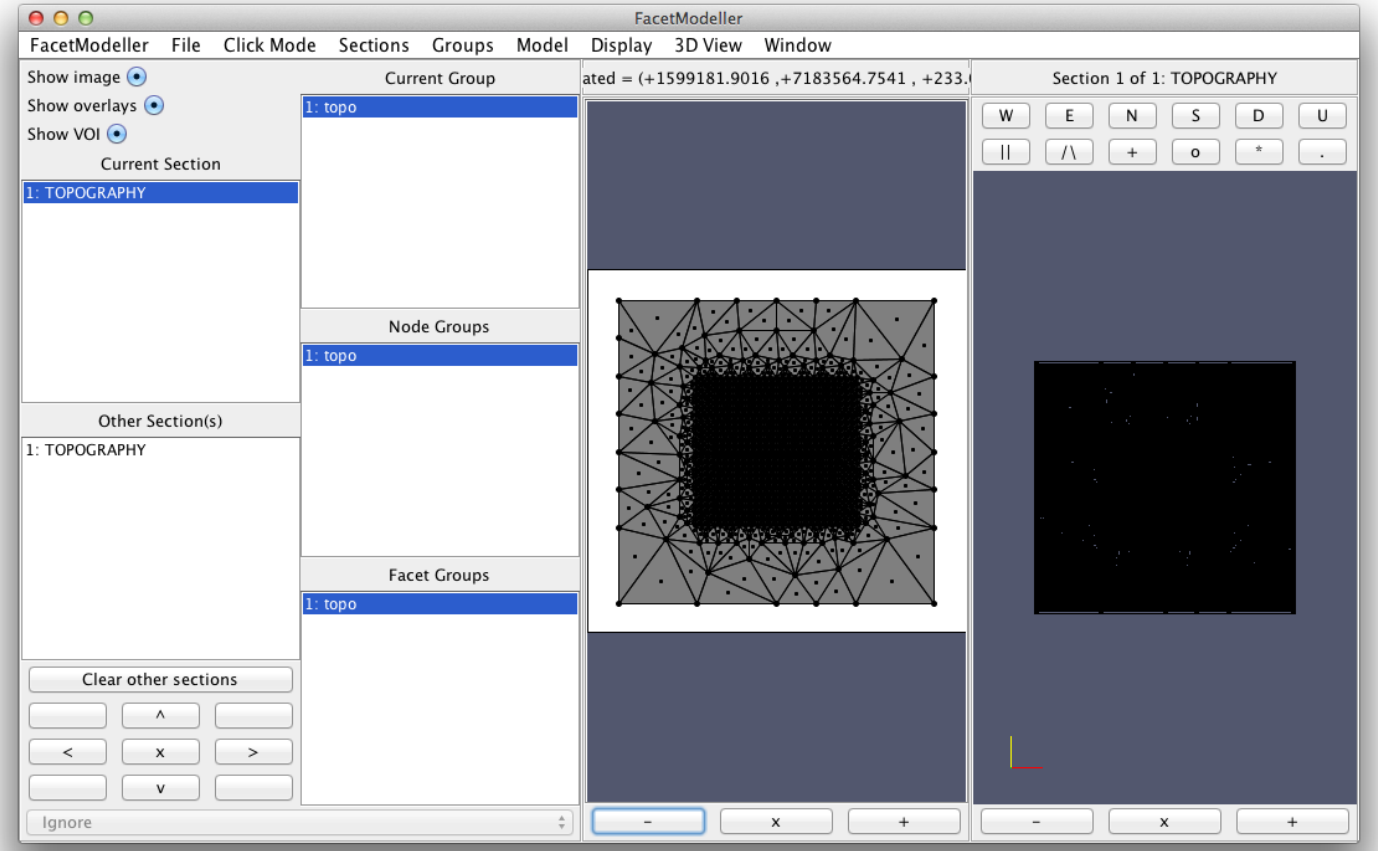
12.4 Defining the padded VOI in 3D

You now have a topography surface that conforms to the boundary facets that you will be creating in your 3D model.

1. Launch FacetModeller for building a 3D model.
2. Define the following VOI:
   Easting limits: `1588400 1670000`
   Northing limits: `7184850 7263250`
   Elevation limits: `-40000 10000`
   Note you have used the same easting and northing limits here as when calibrating the 2D section.
3. Define a new group named `topo`.
4. Create a new **depth** section without image named `TOPOGRAPHY`:
   the first calibration point is `"1580240 7177010 233"`
   and the second is `"1678160 7271090 233"`.
5. Select the menu item
   `File > Load from .node/.ele files`
6. Select and open the file `interpolated.node` when prompted.
7. Select and open the file `topo2d.1.ele` when prompted. Although this file came from the 2D meshing program Triangle, it defines triangular facets that can also be used in a 3D model.

> What you have done here is generated a depth section without image (named `TOPOGRAPHY`) onto which you have then loaded the nodes and facets from the `.node` and `.ele` files. The nodes are held in memory in 3D. When you select the `TOPOGRAPHY` section as the `Current Section`, those nodes are projected onto the section in the 2D viewing panel. You can use this approach for loading any other set of nodes and facets that need not be associated with the topography surface. For example, you may have `.node` and `.ele` files that define the surface of a more-or-less vertical fault. You could then define a vertical cross section without image and load your files onto that section. Of course, you need to be careful to calibrate the section so that any nodes you load from file will be projected onto the section.

Again, don't forget to save early and often! You should now see the following:



The 3D viewer in FacetModeller is fairly rudimentary and can be quite slow if you are displaying a large model. We hope to develop a more robust and more efficient 3D viewer in the future. For now, to avoid speed issues for this example, hide the facets:

8. Unselect the `topo` facets in the `Facet Groups` selection box (control- or command-click, depending on your operating system).

You now need to define some new sections and groups to add new nodes and facets to.

9. Create a new depth section (DEPTH section!) without image and name it `bottom`, for the bottom of the VOI:
   the first calibration point is `"1588400 7184850 -40000"`
   and the second is `"1670000 7263250 -40000"`.

10. Create another new depth section without image and name it `centre`, for the bottom of the central block (where you want finer cells):
    the first calibration point is `"1608400 7204850 -20000"`
    and the second is `"1650000 7243250 -20000"`.

11. Define new groups named `outer` and `inner` coloured green and red respectively.

12. Oh, how annoying, the topography facets came back. Hide them again.

You now need to define nodes on those two new sections.

13. Select the `outer` group as the `Current Group`.

14. Select the `bottom` section as the `Current Section`.

15. Add nodes to the corners of that section.

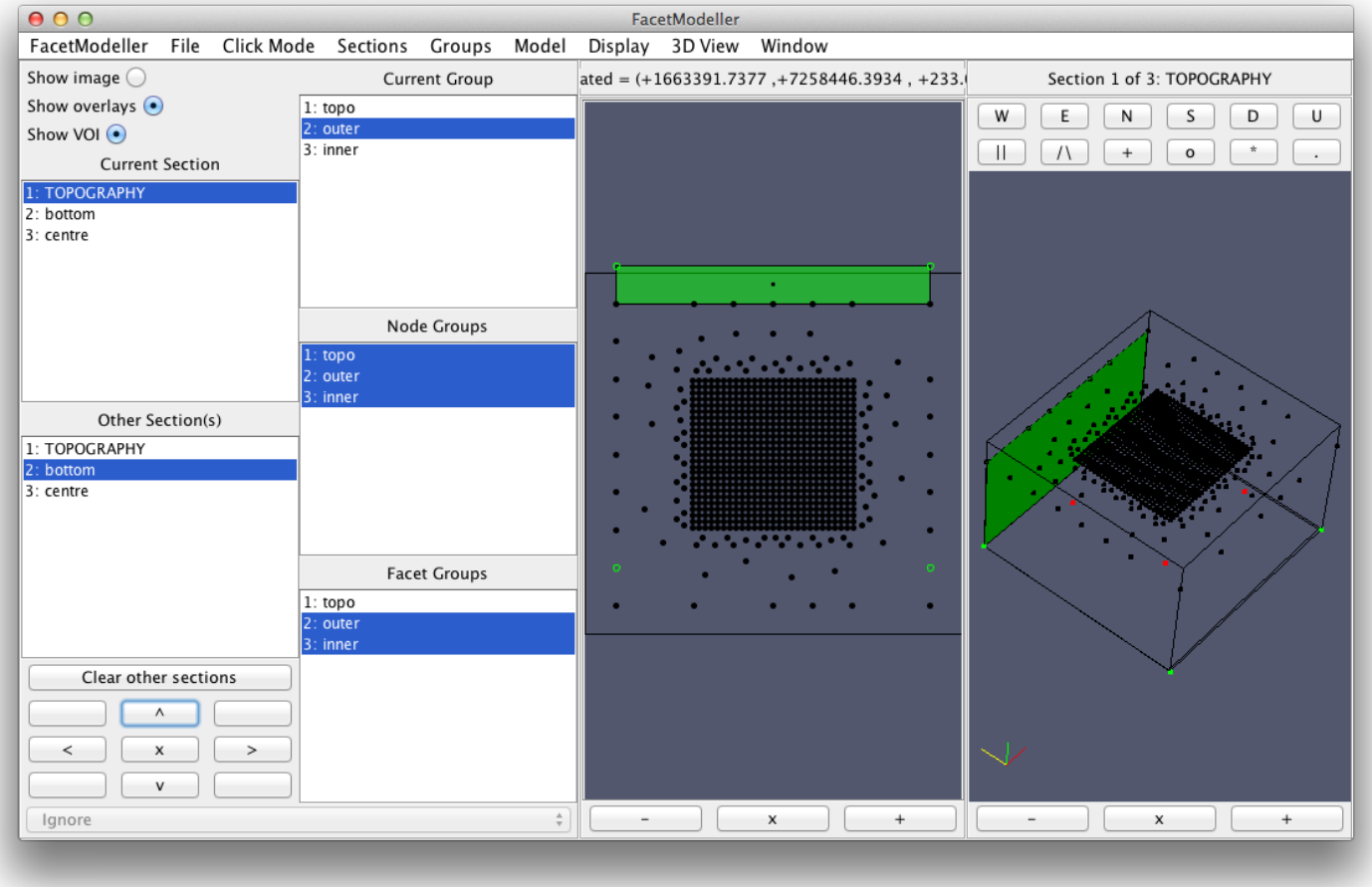
16. Snap nodes to calibration points.

17. Repeat steps 13 through 16 above but with group `inner` and section `centre`.

Wiggle your 3D model around in the 3D viewing panel and you should see those new nodes drawn green and red. The final step is to define the facets on the outer VOI boundary and on the inner central section.

18. Hide the section image (unselect the `Show image` radio button at top left).
19. Select the `outer` group as the `Current Group`.
20. Select the `TOPOGRAPHY` section as the `Current Section`.
21. Select the `bottom` section as the `Other Section`.
22. Shift the other section overlays up one step (using the top middle button in the 3-by-3 grid on the bottom left of the FacetModeller window).
23. Using the node-by-node approach, start defining a polygonal facet that contains all the black topography nodes along the northern side of the VOI, from left to right in the 2D viewer, then moves to the top right green node in the `bottom` section, then the top left green node, then back to the first black topography node you started with. In this facet definition click mode, you can actually drag across the nodes you want to add to the facet, or simply click as usual. However, when dragging, if you come close to a node it will include that node in the facet, and you may end up selecting nodes you didn't want. If that happens, change the click mode to ignore and then back to the node-by-node facet definition click mode and start again.

You should now have created a facet that defines one side of the outer VOI boundary. Your model should now look like this:



In this example, the sides of the outer VOI boundary are all aligned with the Cartesian axes. This ensures that the facet you just created is planar. Recall that in a previous aside I said you could define your 2D VOI boundary however you liked. Although that is true, using boundary facets that are not aligned with the Cartesian axes makes it far more difficult to ensure that the facets on the sides of the boundary are planar. Well, it shouldn't be too difficult given the right tools, but I haven't developed them yet. Therefore, I highly suggest you always use a VOI that is aligned with the Cartesian axes!

To finish defining the facets in your model:

24. Utilizing the overlay shift buttons as required, define the other 3 facets on the remaining sides of the outer VOI boundary.

25. Select the `bottom` section as the `Current Section` and define a rectangular facet on the bottom of the VOI boundary.
26. Repeat steps 19 through 25 above to define the side and bottom facets for the central block (you will use the `inner` group instead of `outer`, and the `center` section instead of `bottom`). You may have to hide the 3D viewing panel, or zoom in on the 2D viewing panel, to help you define these facets.

To check that you have done this correctly:

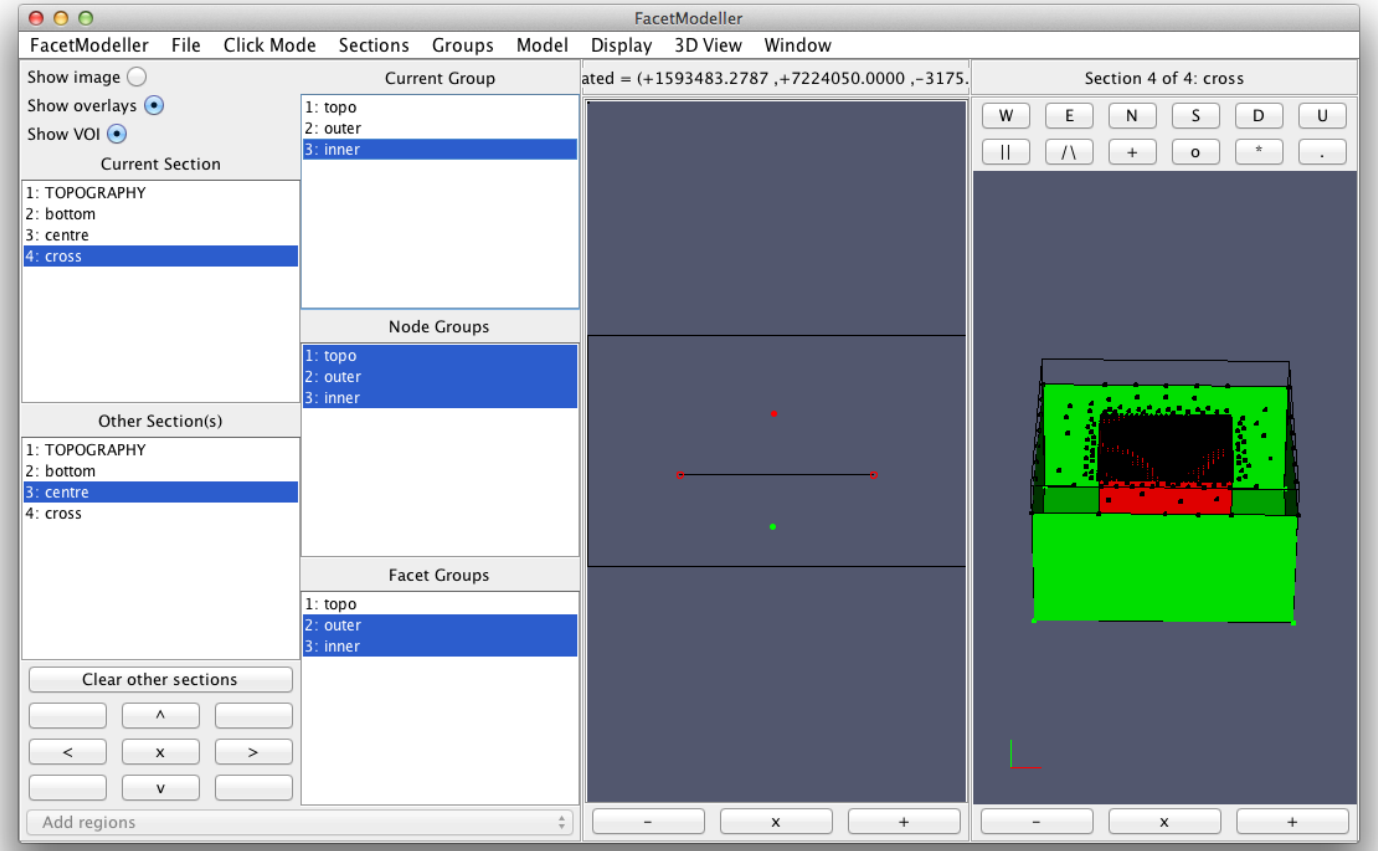
27. Select the menu item
    `Model > Model information`

This will provide a dialog box showing you information about the number of nodes and facets in the model. The total number of facets should now be 1556, 10 more than the number of topography facets loaded from file. Those 10 facets include 4 on the sides of the boundary, 4 on the sides of the inner region, and 2 to close the bottoms of the outer boundary and inner region.

Finally, you need to define the two regions in your model. Defining regions in 3D is identical to the process in 2D but of course you have to now think in 3D: each region point is still attached to a specific section and you have to make sure that the region points are within the correct parts of the model volume. For this example, you will create a new vertical cross section that will slice the model in half. The region points will be attached to that new vertical cross section. You will view the model projected onto that section (in the 2D viewing panel) and you will overlay some of the other model objects such that it will become clear where you have to define the region points.

28. Create a new vertical cross section without image and name it `cross`:
    the first calibration point is `"1588400 7224050 10000"`
    and the second is `"1670000 7224050 -40000"`.
29. Select that `cross` section as the `Current Section`.
30. Select all the sections in the `Other Section(s)` selection box.
31. Select the `outer` group as the `Current Group`.
32. Add a new region point in the outer padding region. This can be anywhere inside the VOI and below the bottom of the inner region.
33. Select the `inner` group as the `Current Group`.
34. Add a new region point in the inner central region. Remember that the model contains topography and above the topography surface is air. Hence, the VOI contains some air and you need to make sure you add this region point somewhere below the topography surface, but still above the bottom of the inner region.

Your model should now look like this (keep in mind the topography facets are still hidden):



You can now export your model to a `.vtu` file for viewing in ParaView, or to a `.poly` file for meshing with TetGen.

Chapter 13

Tutorial - Building 3D models: III

Dealing with outcropping units

This 3D tutorial example relies on lessons learned in the 2D tutorial, 3D tutorials. You should also refer to the 2D tutorial for basic information about FacetModeller, including installation and running the program.

In this tutorial, you will learn how to sew outcropping rock unit contacts to the topography. I won't be providing explicit step-by-step instructions like in the other tutorials so I hope you remember how to do some of the basic tasks.

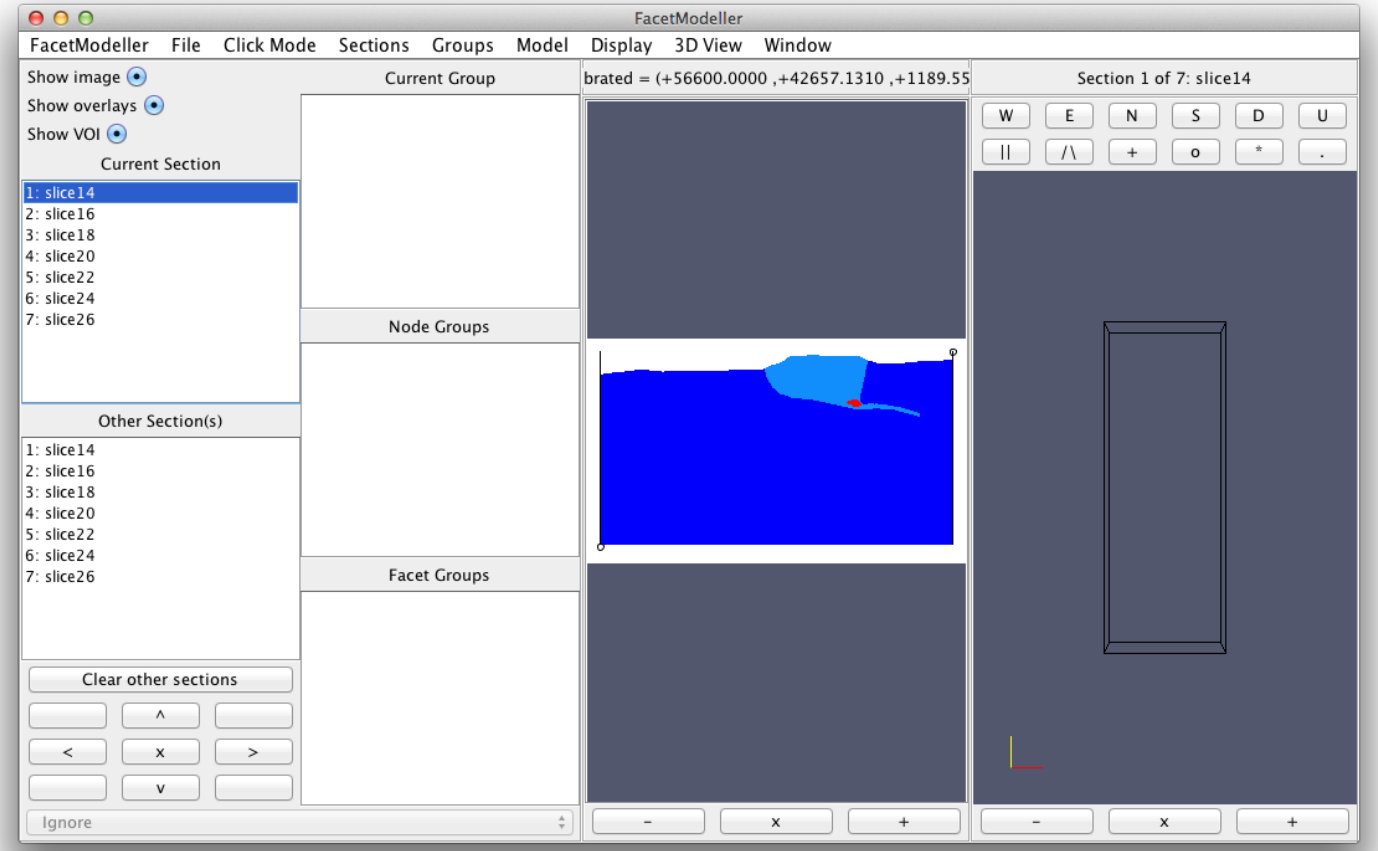
13.1 Preliminary instructions

1. Launch FacetModeller for building a 3D model.
2. Define the following VOI:
   Easting limits: `56600 57800`
   Northing limits: `40300 43600`
   Elevation limits: `-1600 225`
3. Load the image files named `slice##.png` where `##` runs from `14` to `26` (by 2). Those in my research group can find these files here
   `[repos]/peter/NetBeansProjects/examples/FacetModeller3Dc/`
   and public users here
   `[repos2]/facetmodeller/Tutorial_3D_Part3/`
   The vertical cross-sections are normal to and stacked along the x (easting) direction.
4. Calibrate section `slice14` with the following calibration information:

Click here	Corresponding spatial coordinates
bottom left of the black axes in the image	`"56600 40300 -1600"`
top right of the black axes	`"56600 43600 225"`

5. Copy the calibration to all the other sections, using a separation of `200` between sections.

Don't forget to save early and often! You should now see the following:



For this example, there are three rock units:

1. gneiss (dark blue in the images)
2. troctolite (light blue in the images)
3. sulphides (red in the images)

You'll ignore the sulphides in this tutorial and focus on modelling the troctolite body. Define new groups with names and colours indicated in the table below (you can use whatever colours you like but I find these colours helpful for this example, and I'll be referring to them in the text so I suggest you stick with these):

Name	Colour
topography	grey
troctolite	orange
outcrop	red
newTopo	green

13.2 Opening topography files

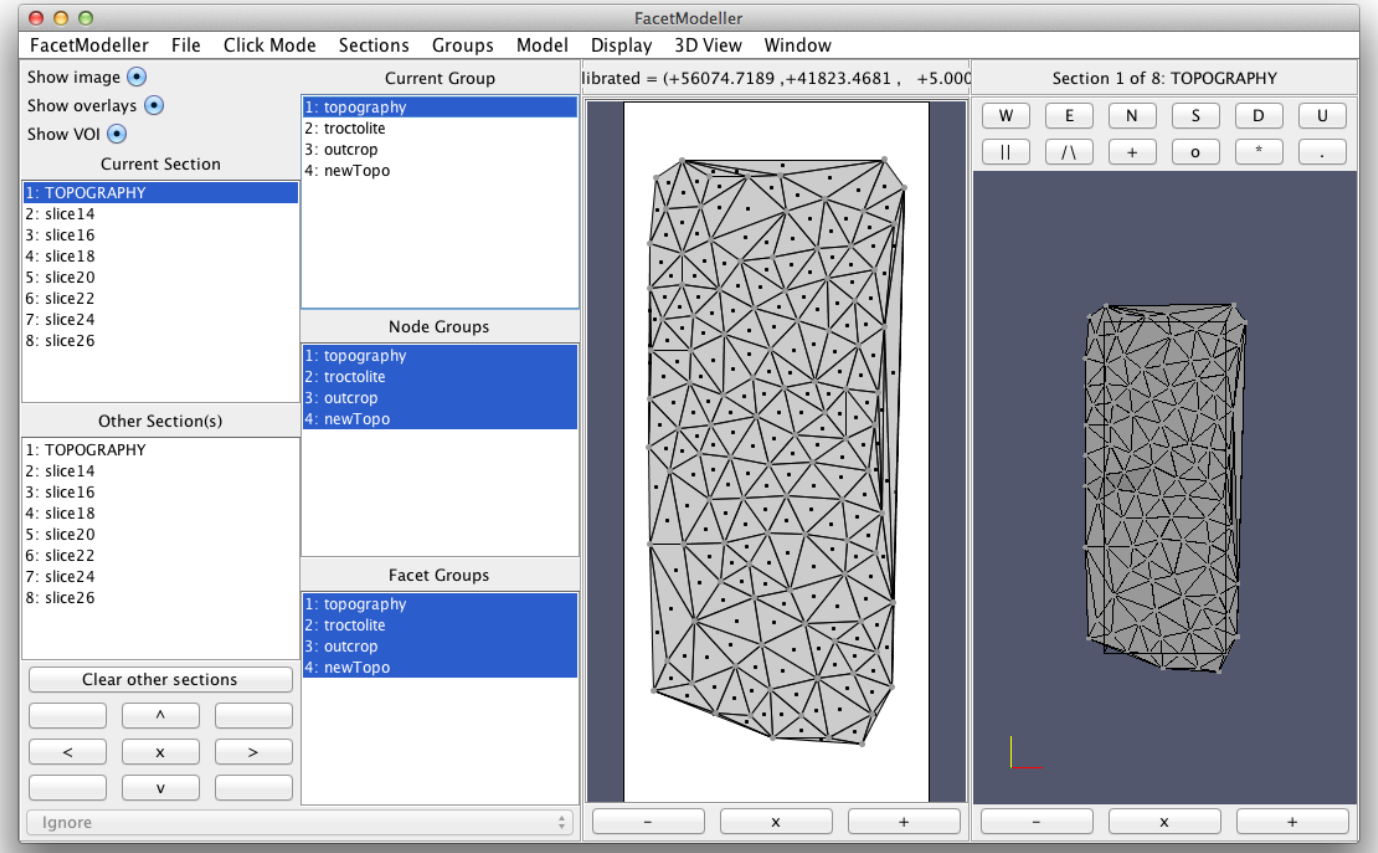
Getting your topography surface ready may take considerable effort. The process is explained in the second 3D tutorial. In this tutorial, you will not bother sewing the topography surface to the VOI.

When there are rock units that outcrop, leading to surfaces in your model that need to be sewn to the topography surface (as in this tutorial), you will want the spacing of the topography nodes to be fairly consistent with the spacing in the other surfaces to which it must be sewn. Otherwise you can end up with skinny triangles that are detrimental to mesh quality.

In this tutorial I have processed the topography surface as required (but not bothered to sew it to the VOI). To load it:

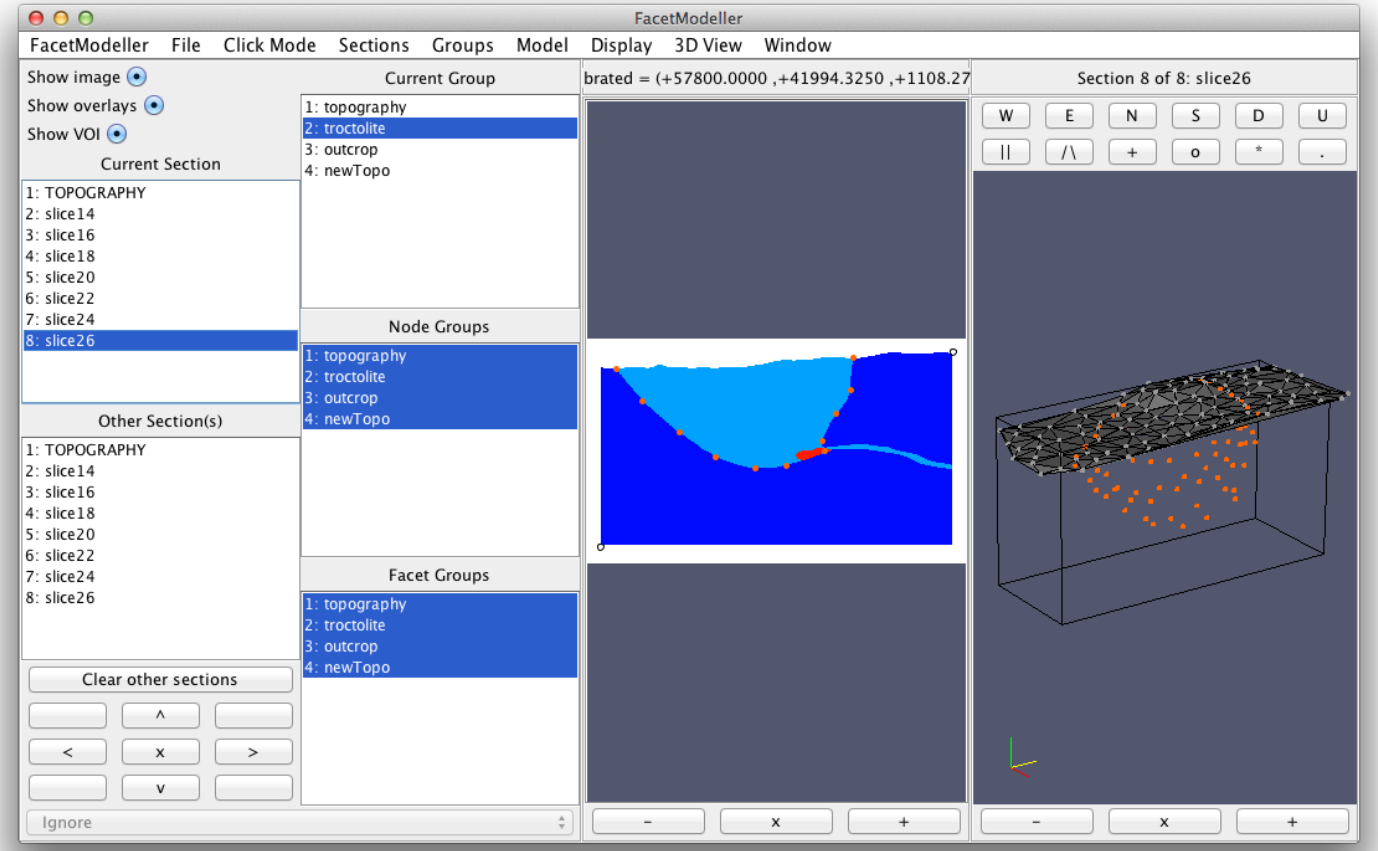
1. Make sure the `topography` group is the `Current Group`.
2. Create a new **depth** section without image named `TOPOGRAPHY`:
   the first calibration point is `"56241 39740 5"`
   and the second is `"58160 44141 5"`.
3. Select the menu item
   `File > Load from .node/.ele files`
4. Select and open the file `topo_dec3d.node` when prompted (in the same directory as the section images).
5. Select and open the file `topo_dec3d.ele` when prompted (in the same directory).
6. When asked for a colour, select the default white.

You should now see the following:



13.3 Adding nodes

As in the first 3D tutorial, you will define some nodes around the anomalous body (the troctolite in this example). Select the `troctolite` group and, for every `slice##` section, add nodes to define the gneiss-troctolite contact (between the dark and light blue colours in the images). Make sure you add nodes where this contact outcrops (connects with the topography). You can ignore the long wing-like feature at depth. Your model should now look like this:



13.4 Changing node groups

The complication in this model building example is that the gneiss-troctolite contact outcrops and must be sewn correctly to the topography surface. Before you can start creating facets, you need to make sure you can distinguish between the nodes on the contact that are below the topography and those at the topography surface.

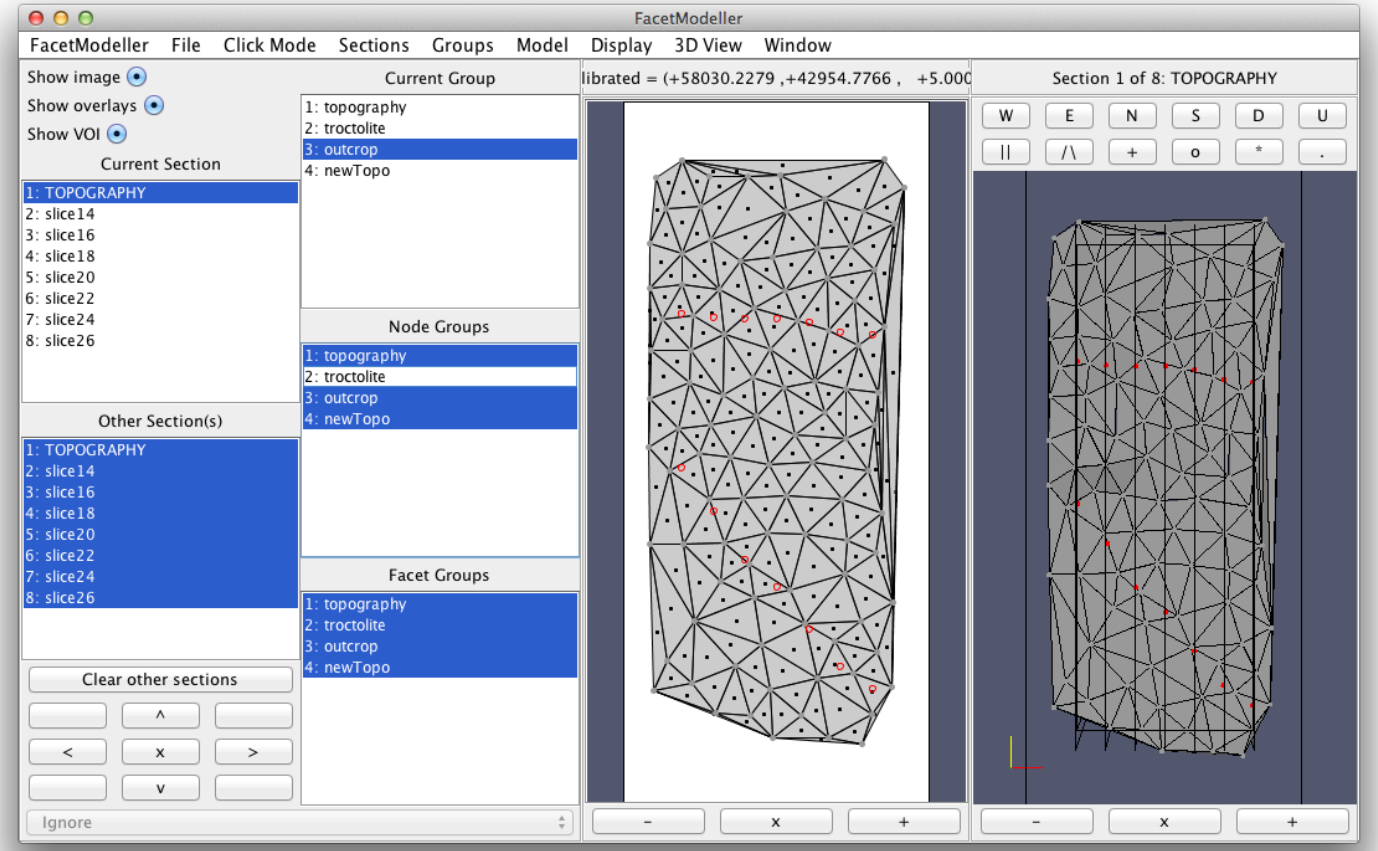
1. Select the `outcrop` group as the `Current Group`.
2. Select the menu item
   `Click Mode > Change node groups`
   Any node you now click on will be moved into the selected `outcrop` group.
3. For every `slice##` section, click on the two nodes where the gneiss-troctolite contact outcrops. You should see them change colour to red.

13.5 Defining facets

Before you continue, you will need to change the view in each view panel to an overhead view and change what is being displayed:

4. Select the `TOPOGRAPHY` section as the `Current Section`.
5. Select all the sections in the `Other Section(s)` selection box.
6. Show the 3D viewer panel and press the button labelled `D` (view downward) on the top of the 3D viewer panel.
7. Zoom in a couple times on the 3D viewer.
8. Unselect the `troctolite` group in the `Node Groups` selection box.

You should now see this:

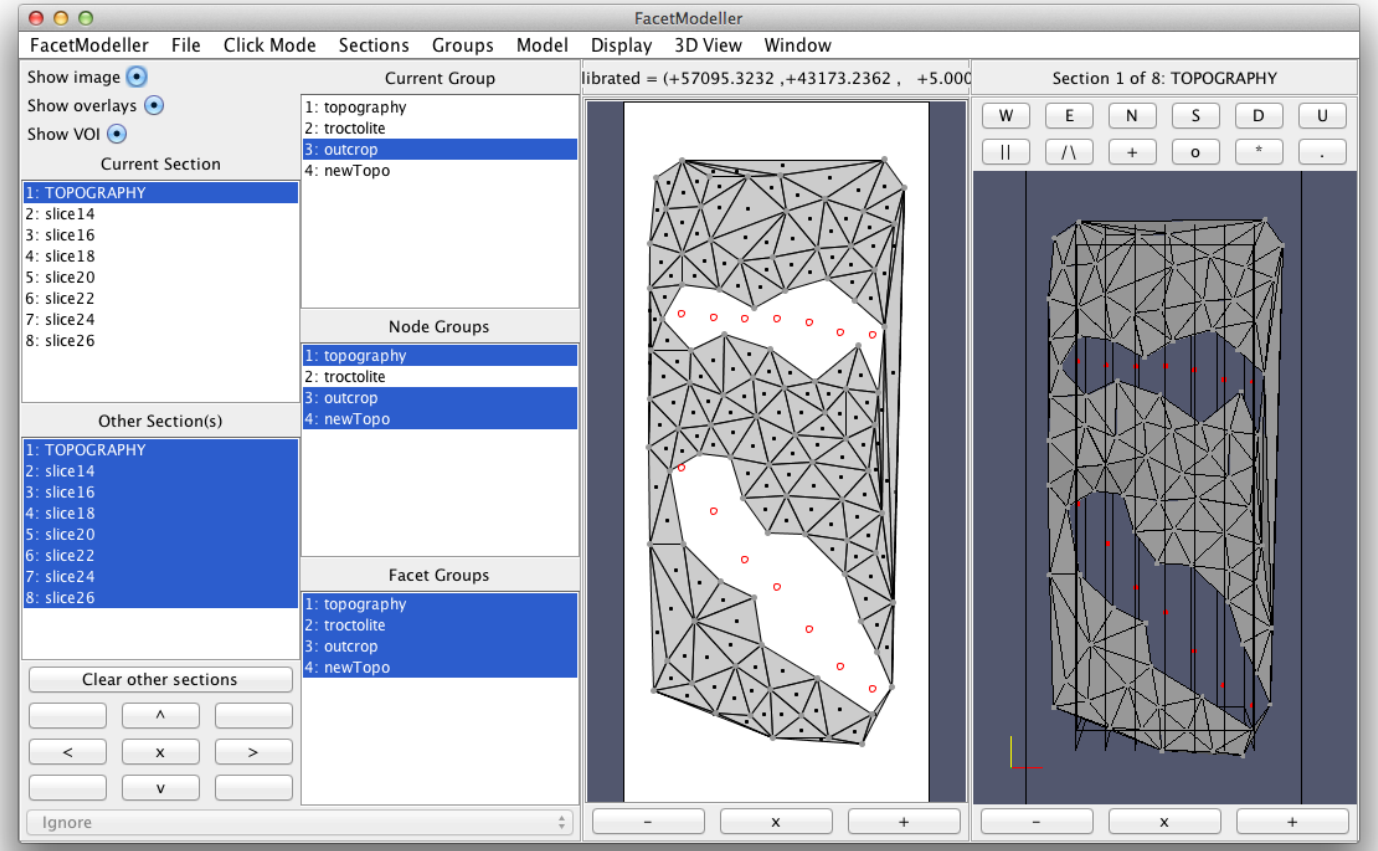


To sew the gneiss-troctolite contact to the topography surface you'll need to make sure that the nodes in the `outcrop` group are incorporated into the topography. To create the gneiss-troctolite contact surface, the red nodes will need to be connected between adjacent parallel sections. This will create two lines of red nodes along the topography surface where the gneiss-troctolite contact outcrops. Any topography facets that interfere with those connections must be removed. Refer to the next screen-shot for guidance with these steps:

9. Delete any topography nodes close to the red `outcrop` nodes or close to lines that connect those red nodes across adjacent sections.

10. Delete any remaining topography facets that enclose the red `outcrop` nodes or have edges that would intersect the lines of red nodes required to define the outcrop of the gneiss-troctolite contact.

Your model should now look like this

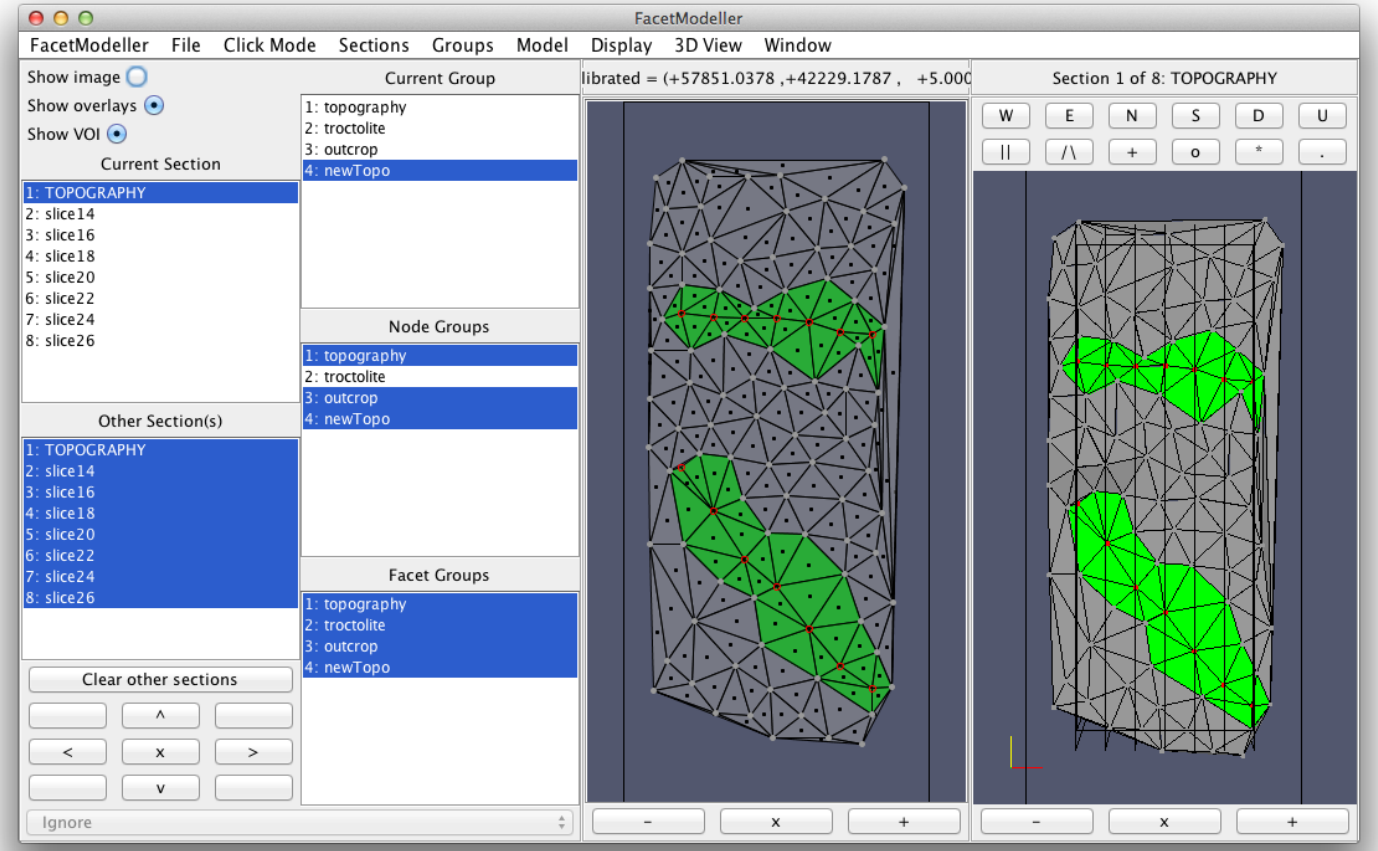


You now need to fill in the holes in the topography with new facets.

11. Select the menu item
    `Click Mode > Define triangular facets`
12. Select the `newTopo` group as the `Current Group`.
13. Unselect the `Show image` radio button radio button.
14. Define new triangular topography facets to fill in the holes. **Make sure there are facet edges linking all of the red nodes in the `outcrop` group.**

> Sometimes it may not be clear how to best connect the nodes to provide the best quality mesh (which may or may not be important to you, depending on how you intend to use your model). To help with this, the text bar above the 2D viewer panel will display the minimum vertex angle of each candidate triangular facet: for a quality mesh, you will want to avoid small vertex angles.

This task takes a while to complete. I suggest you practice defining a few facets and move on once you understand what you are doing and why you are doing it. If you do manage to get through it, your model should look like this from overhead:

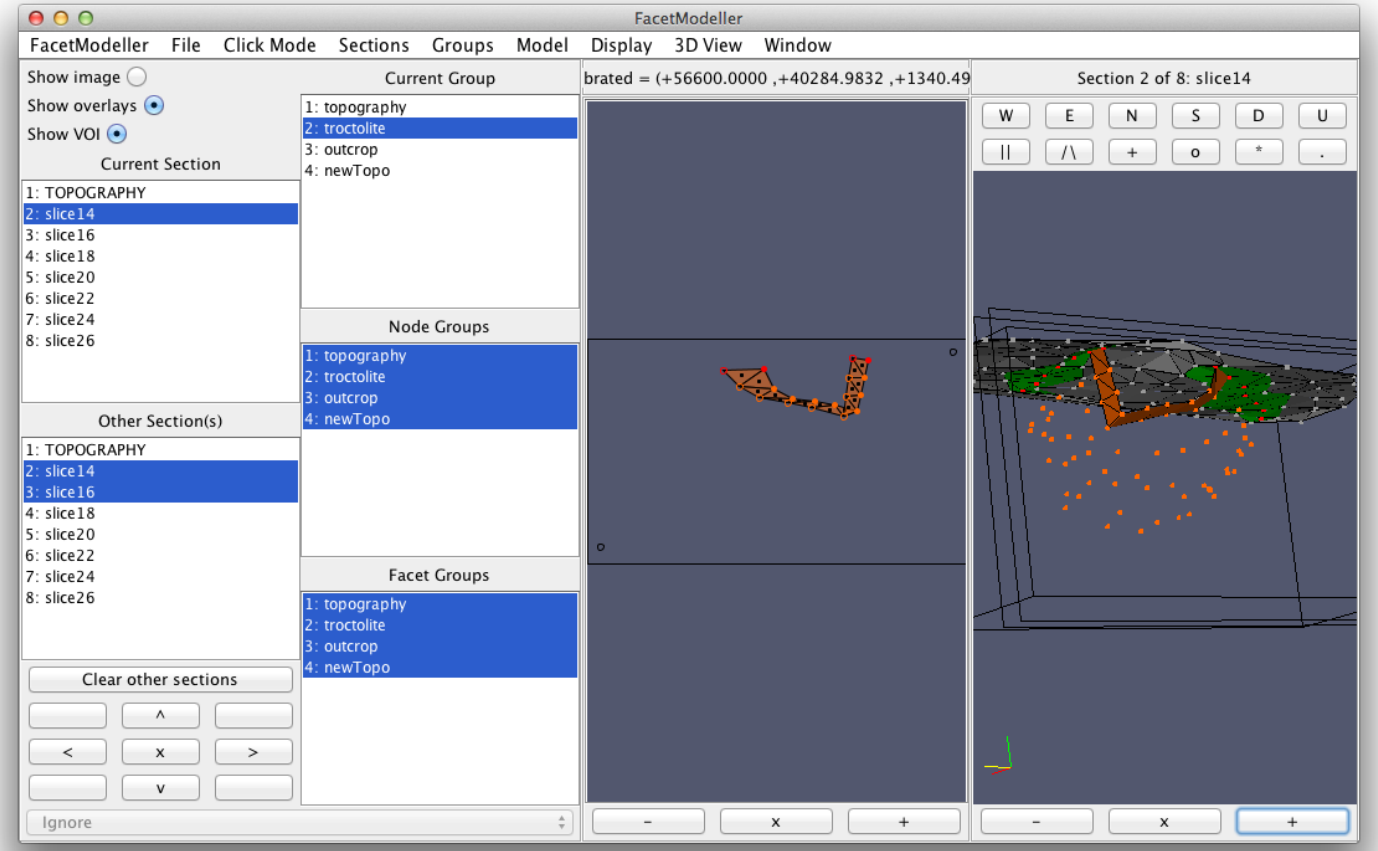


Again, the important part is that there are edges linking all of the red nodes in the `outcrop` group.

> To make this example even more complicated, you could also consider adding new nodes along the topography surface on each vertical section, then delete/define facets to merge those nodes into the existing topography surface.

13.6 Finishing the model

Now that the topography is taken care of, you can begin to create the gneiss-troctolite contact surface below the topography. To do so, follow the steps you took in the first 3D tutorial where you created facets between parallel sections. Add these facets to the `troctolite` group. Once you have finished this step with the first pair of sections, your model should look like this:



13.7 Wrap-up

You should now know how to complete this model (if you really want to). I haven't covered every possible 3D model-building task in these tutorials, nor have I shown you all that FacetModeller can do. My hope is that the application is simple enough to use (once you get the basic idea of how it works from these tutorials) that you can discover the other functionality by yourself while sifting through the menu items. I am, of course, always happy to provide support.

Future major development plans are to allow you to:

- Add new nodes to a topography surface, with the elevation of the new nodes interpolated from the existing information.
- Smoothly subdivide all the facets in a model to generate a finer smoother model, so you'll be able to create a coarse approximation of your model and then refine it automatically.

Please let me know if there are any other enhancements to FacetModeller that would help you with your model building needs. Thanks for playing!

Part IV

A collection of programs for

Multi-modal Applied Geophysical Numerical modelling on Unstructured Meshes

Chapter 14

MAGNUM preliminaries

14.1 About MAGNUM

MAGNUM is a collection of three programs (`FOGO`, `VIDI` and `DYNO`) for Multi-modal Applied Geophysical Numerical modelling on Unstructured Meshes. Despite the name, rectilinear meshes are also supported by this package. MAGNUM includes only forward or inverse modelling programs; for utility programs, see the PODIUM package.

14.2 Policies and conditions of use

Refer to Chapter 1 for general disclaimers and conditions of use. The programs are available to those in my research group but we are not yet providing these programs to the world at large. The programs are available on request for academic use: please contact the main developer.

14.3 Obtaining MAGNUM

Refer to Chapter 2 for general instructions on obtaining the software described in this documentation.

14.4 Installing MAGNUM

Source code is NOT available. MAGNUM is provided as a collection of compiled executables for particular operating systems. Hence, no installation is required once the programs have been obtained from the repository. We make efforts to test the compiled programs on various operating systems but without direct access to users' computing systems we can not guarantee that these codes will execute on your system. The `mac` version is currently being compiled on OS X 10.13.2 (High Sierra), the `win` version on Windows 7 and the `lin` version on Ubuntu, all using `gfortran` as the compiler.

14.5 Running the programs

The programs are run in the command line, e.g. from a Terminal window on a Mac or a Command Prompt window on a Windows machine. No graphical user interfaces have been developed for two reasons: 1) it is simple to develop batch scripts using programs developed for command line use; 2) we feel that the significant development time required for a GUI is better spent developing new functionality. The documentation below should be reviewed carefully before running the programs. Any program can be run with no input parameters (that is, simply type the name of the program in a command window and hit return/enter) to obtain a short reminder of the command line usage.

Many of these programs will run in parallel (via OpenMP directives). Remember to set the OMP_NUM_THREADS environment variable. On a Mac you use the following command in the terminal window before running this program:

```
> export OMP_NUM_THREADS=value
```

Chapter 15

Forward modelling of geophysical data: FOGO (formerly FWD)

15.1 Program summary

FOGO is a program for **FO**rward modelling of **G**e**O**pysical data. It provides support for the following data types:

- scalar gravity data (vertical component other single Cartesian component)
- gradiometry (tensor) gravity data
- total magnetic field data
- first-arrival seismic traveltimes

To calculate magnetic data with remanence you may have to perform two calculations, e.g. one for the induced field and another for the remanent component. Program sum_node can be used to combine the two data responses.

The model can be build on a rectilinear or unstructured mesh, or can involve unstructured wireframe surfaces representing contacts between rock units.

This program will run in parallel. Remember to set the OMP_NUM_THREADS environment variable. On a Mac you use the following command in the terminal window before running this program:

```
> export OMP_NUM_THREADS=value
```

Potential field modelling uses the methods of Okabe (1979). Seismic modelling uses the methods of Lelièvre et al. (2011).

15.2 Running the program

15.2.1 Command line usages:

```
fogo meshinp propinp datainp

fogo meshinp propinp datainp outroot

fogo meshinp propinp datainp outroot dosen

fogo meshinp propinp datainp outroot dosen ai
```

15.2.2 Command line parameters

`meshinp`

- A model discretization specification file.
- See the documentation on model discretization specification files for more information on the file format.

`propinp`

- A physical property specification file.
- See the documentation on physical property specification files for more information on the file format.

`datainp`

- The name of the geophysical data input file (the format is described below).

`outputroot`

- The names of the output files have this root (the output files are described below).
- If `outputroot` is absent then it is set to the root of the input file, that is, the input file name with extension and any path information stripped off: for example, if `inputfile` is set to `mydirectory/myfile.ext` then the output files will be named `myfile*` and placed in the current working directory.

`dosen`

- If set to `"t"` (true) then sensitivity information is calculated.
- The default is `"f"` (false).

`ai`

- Only used for a mesh-based model.
- Specifies the index of the attribute to use for the model.
- If `ai` ≤ 0 then this input is ignored.
- If `ai` > 0 then this input overrides all other specifications of what model to use (the attribute names in the model file header are ignored).

15.3 Data input file format

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.

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

Name	Default	Brief description
		PARAMETERS COMMON TO ANY DATATYPE
`datatype`	`""`	this must be the first parameter specified in the file
`datafile`	`""`	file containing the data response (for inversion only)
`gamma`	`1.0`	multiplier on the data misfit term (for inversion only)
`lambdainit`	`0.0`	initial trade-off parameter value (for inversion only)
`chifact`	`1.0`	normalized target misfit (for inversion only)
`chitol`	`0.05`	relative tolerance on the target misfit (for inversion only)
`measure`	`"ell2"`	specifies the measure for the misfit (for inversion only)
`ekblomp`	`2.0`	the p-value for the Ekblom measure (for inversion only)
`ekblome`	`1.0E-9`	the epsilon value for the Ekblom measure (for inversion only)
`huberc`	$\gg 0$	the c-value value for the Huber measure (for inversion only)
		PARAMETERS SPECIFIC TO GRAVITY OR MAGNETICS DATA
`obsfile`	`""`	file containing the observation locations
`approx`	`"f"`	perform approximate modelling or not
`move`	`"f"`	allows you to copy the data to the x or z coordinate
`mtxroot`	`""`	defines the file(s) containing the sensitivity matrix/matrices
`compmeth`	`"none"`	compression method; set to `"none"`, `"noco"` or `"wave"`
`compdir`	`"row"`	compression direction; set to `"row"` or `"col"`
`wavelet`	`"null"`	type of wavelet compression: set to `"daub1"` – `"daub6"`, `"symm4"` – `"symm6"`, or `"null`
`tol`	`0.0`	relative wavelet threshold
`level`	`0.0`	value added to the observed data before inverting (for inversion only)
`autolevel`	`"f"`	if `"t"` (true) then an attempt is made to automatically level the data (for inversion only)
		PARAMETERS SPECIFIC TO GRAVITY DATA
`comp`	`"z"`	specifies which gravity component to use; set to "x", "y" or "z")
		PARAMETERS SPECIFIC TO GRAVITY GRADIOMETRY DATA
`comps`	`tttttt`	specifies which tensor components to use

Name	Default	Brief description
		PARAMETERS SPECIFIC TO MAGNETICS DATA
magdata	""	magnetics data type; set to "tmi", "xyz" or "amp"
magmodel	""	magnetics model type; set to "sus", "pst" or "atp"
igeo	0.0	geomagnetic field inclination in degrees
dgeo	0.0	geomagnetic field declination in degrees
sgeo	1.0	geomagnetic field strength in nT
idir	0.0	measurement inclination in degrees
ddir	0.0	measurement declination in degrees
		PARAMETERS SPECIFIC TO FIRST ARRIVAL TRAVELTIME DATA
sourcesfile	""	.node file specifying the source locations
receiversfile	""	.node file specifying the receiver locations
combosfile	"null"	.ele file specifying the source/receiver combinations, or set to "all" or "match"
recip	"f"	set to "t" (true)to perform reciprocal modelling
radius	0.0	the initialization radius in the fast marching
thresh	0.0	a threshold on the sensitivity values
tracemode	"none"	specifies the type of tracing to perform (if any)
sloray	0.0	homogeneous slowness value to remove when calculating traveltimes along ray paths
senfullflag	"f"	set to "t" (true) to use a full sensitivity matrix instead of sparse
writettimes	"t"	if "t" (true) then the traveltimes are written to an output file
writettypes	"f"	if "t" (true) then the traversal types are written to an output file
		PARAMETERS SPECIFIC TO MUOGRAPHY DATA
method	"sph"	"sph" pr "cart" for spherical or Cartesian formulations respectively
obsfile	" "	muography observation locations (see below for specific file format)
receiversfile	" "	a .node file specifying the receiver location(s)
nsub	-1	Used differently depending on the method (see below for more information)
mtxroot	""	defines the file(s) containing the sensitivity matrix/matrices
level	0.0	value added to the observed data before inverting (for inversion only)
autolevel	"f"	if "t" (true) then an attempt is made to automatically level the data (for inversion only)
alphac	0.0	weighting for additional inversion parameter for solving for the level

15.3.1 Units

If all spatial quantities are in m (metres) then:

- For gravity data, if density is in g/cm^3 (grams per cubic centimetre) then the data is in $mGal$ (milligal).
- For gravity gradiometry data, if density is in g/cm^3 (grams per cubic centimetre) then the data is in E (eotvos).
- For magnetic data, if susceptibility is S.I. (unitless), or magnetization is expressed as an effective susceptibility, then the data is in nT (nanotesla).
- For seismic data, if slowness is in s/m (seconds per metre) then the data is in s (seconds).
- For muography data, if density is in g/cm^3 (grams per cubic centimetre) then the data is also in g/cm^3.

15.3.2 Specifying the type of data response

datatype

- Set to "fat" for first-arrival traveltime data.
- Set to "gz" for gravity data (vertical component or other single Cartesian component).
- Set to "gg" for gravity gradiometry data.
  See more below under parameter datafile for specifications on the data file format for this case.
- Set to "mag_tmi_sus" for total field magnetic intensity data with susceptibility model.
- Set to "mag_tmi_pst" for total field magnetic intensity data with Cartesian vector magnetization model.
- Set to "mag_tmi_aid" for total field magnetic intensity data with spherical vector magnetization model.
- Set to "mag_xyz_sus" for multi-component magnetics data with susceptibility model.
- Set to "mag_xyz_pst" for multi-component magnetics data with Cartesian vector magnetization model.

- Set to "mag_xyz_aid" for multi-component magnetics data with spherical vector magnetization model.
- Set to "mag_amp_sus" for magnetic amplitude data with effective susceptibility model.
- Set to "muog" for muography data.

igeo, dgeo, sgeo, idir, ddir

- Inclination, declination and strength of the geomagnetic inducing field and measurement direction.
- Enter the angles in degrees and strength in nT.
- For total field data set the measurement direction equal to the inducing field:
  idir = igeo
  ddir = dgeo.
- dgeo and ddir are ignored for 2D problems.
- For 2D problems, any magnetization component in the strike direction (normal to the plane of the vertical section) will not contribute to the data measurements, assuming that the 2D model is constant to infinity in the strike direction. Any magnetic data component measured in the strike direction will also be zero. Hence, **for 2D problems, igeo and idir should be specified as inclinations projected onto the 2D section.** This can be achieved as follows:

$$\vec{v}_{proj} = \vec{v}_{geo} - (\vec{v}_{geo} \cdot \hat{n})\,\hat{n}$$

where $\vec{v}_{geo}$ is the geomagnetic field vector, $\vec{v}_{proj}$ is the projected vector and $\hat{n}$ is a unit vector normal to the 2D section plane. To calculate $\vec{v}_{geo}$ you can use:

$$\begin{aligned}\vec{v}_{geo,x} &= \texttt{sgeo}\cos(\texttt{igeo})\cos(\texttt{dgeo})\\ \vec{v}_{geo,y} &= \texttt{sgeo}\cos(\texttt{igeo})\sin(\texttt{dgeo})\\ \vec{v}_{geo,z} &= \texttt{sgeo}\sin(\texttt{igeo})\end{aligned}$$

which is in a $+z$-down system. For vertical 2D section planes (horizontal normal vector), the projected inclination, θ_{proj}, can be calculated using:

$$\begin{aligned}r &= \sqrt{\vec{v}_{proj,x}{}^2 + \vec{v}_{proj,y}{}^2}\\ z &= \vec{v}_{proj,z} = \vec{v}_{geo,z}\\ \theta_{proj} &= \arctan(z/r).\end{aligned}$$

Furthermore, if you know the declination angle of the 2D section, ϕ_{sect}, then the projected inclination, θ_{proj}, can be calculated using:

$$\begin{aligned}r &= \texttt{sgeo}\cos(\texttt{igeo})\cos(\Delta\phi)\\ z &= \texttt{sgeo}\sin(\texttt{igeo})\\ \theta_{proj} = \arctan(z/r) &= \arctan\left(\frac{\sin(\texttt{igeo})}{\cos(\texttt{igeo})\cos(\Delta\phi)}\right) = \arctan\left(\frac{\tan(\texttt{igeo})}{\cos(\Delta\phi)}\right)\end{aligned}$$

where $\Delta\phi$ is the angle made between the geomagnetic field direction and the 2D section, e.g.

$$\Delta\phi = \texttt{dgeo} - \phi_{sect}$$

but be careful when making such a calculation if the two angles straddle the 0/360 degree fence.

comps

- If the i^{th} character of the value string is "t" then the i^{th} tensor component is calculated in the forward modelling or is used in the inversion.
- If inverting, all tensor components are assumed to exist in the data file, even if all are not being inverted.
- **See more below under parameter datafile for specifications on the data file format for gravity gradiometry data.**

approx

- If "t" (true) then the triangular or tetrahedral cells in a mesh are approximated as dipoles (circles or spheres).

move

- If this parameter is set to `"x"` or `"z"` then the data are copied to the x or z coordinate column of the output `.vtu` file for viewing purposes.
- This **only** applies to the `.vtu` output file.
- The copying is only performed if the x or z observation values are identical, e.g. for horizontal synthetic airborne data or vertical down-hole data.

15.3.3 Data observations

`obsfile`

- This file is used for gravity or magnetics data only. For traveltime data, use `combosfile`.
- A `.node` file defining the data observations.
- For synthetic modelling, you can use program `make_obs` to generate this file.
- When forward modelling gravity or magnetics data, the observation locations are read from this file.
- When inverse modelling, this file is ignored and the observation locations and data responses read from the `datafile`.

`combosfile`

- This file is used for traveltime data only. For gravity or magnetics data, use `obsfile`.
- This parameter can specify a `.ele` file with two columns of indices that define the source/receiver pairs.
- You can use program `make_obs` to generate this file.
- Specify `"all"` if you want to use all source/receiver combinations.
- Specify `"match"` if you want to match the i^{th} source with the i^{th} receiver.
- When forward modelling seismic traveltime data, the source/receiver pairs are read from this file.
- When inverse modelling, the `combosfile` is ignored and the source/receiver pairs and traveltimes are read from the `datafile`.

`datafile`

- A `.node` or `.ele` file (depending on the data type) containing the data response.
- When forward modelling, this file is ignored.
- When inverse modelling, observation locations (source/receiver pairs for traveltime data) are read from this file.
- For tensor data:
  - The first 6 columns (node attributes) in the data file are the components in this order:
    xx - xy - xz - yy - yz - zz
  - If inverting, the uncertainties should be in columns 7 through 12 in that same order.
  - **The coordinate system for the tensor data components is always a right-handed +z down system (+x north, +y east).** Hence, if you have your data in a right-handed +z up system (+x east, +y north) then you will need to reorder your data columns as follows:
    yy - yx - yz - xx - xz - zz
    and you will also need to change the sign of the xz and yz measurements, but please see the final item below.
  - If, on the other hand, you have your data in a left-handed +z down system (+x east, +y north) then you will still need to reorder your data columns as above but you do not need to change the sign of any measurements.
  - For complete clarity, the components should be in this order (E=east, N=north, D=down):
    NN - NE - ND - EE - ED - DD
  - Note that the NE component equals the EN component, so does not appear in the list of components specified above.
  - If you are at all unclear about the coordinate system and need to convert, especially any uncertainty about having to change the sign on any measurements, then I highly suggest you invert all components separately before continuing with an inversion of multiple components together. When inverting components separately, compare the results, make sure they are all consistent, particularly the sign of the densities in the results, and make sure all inversions converge as expected. If there are any inconsistencies then fix as necessary before inverting components together.

15.3.4 Specifying the modelling grid

`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.
- The `zdir` parameter in the `meshinp` file (see Section 3.3) is applied to the data file. Hence, the code assumes that the coordinate system for the mesh and data files are consistent.
- The `zdir` parameter only affects the observation locations: parameters `datafile`, `obsfile`, `sourcesfile`, `receiversfile`. The coordinate system for tensor gravity components and vector magnetometry are always $+z$ down.

15.3.5 Specifying the model

`modelfile, facetunitsfile, regionunitsfile, regionsfile`

- These are specified in the `meshinp` file.
- The required attribute name is either `Density`, `MagSus`, `Slowness` or `Conductivity` depending on the data type. The case is important. This can be present in the `modelfile` or `regionunitsfile` (explained further below).
- Remember that the property values are scaled using the `mmul` and `madd` parameters (specified in the `propinp` file) to obtain the values used in the modelling.
- For a voxel model:
  - Unless `ai` is specified, the required attribute name should be present in either the `modelfile` or `regionunitsfile` but not both. If it is found in both then an error is thrown and the program quits.
  - Specifying `ai` overrides the required attribute name and the following three bulleted items are irrelevant.
  - Placing the required attribute in the `modelfile` allows you to have different physical property values in every cell.
  - Placing the required attribute in the `regionunitsfile` is an alternate approach if you want to have different physical property values in several homogeneous regions of a model. In this case, the first attribute in the `modelfile` is assumed to contain rock-unit IDs that are linked to physical properties via the `regionunitsfile`. If no attribute columns exist in the `modelfile` then rock-unit IDs of `1` are used.
  - If the required attribute name is not found in the `modelfile` or `regionunitsfile` then the first attribute in the `modelfile` is used instead, regardless of any attribute names in the file. If there are no attributes in the `modelfile` then: a zero-valued model is used if `dosen` is `"t"` (true), or a unit-valued model is used if `dosen` is `"f"` (false).
- For an unstructured surface model:
  - The required physical property attribute must be present in the `regionunitsfile`.
  - For traveltime data (straight or curved rays), the surface model is meshed before performing the fast marching solution. The slowness in every cell of the mesh is determined by taking the region attribute values in the `regionsfile`, which are assumed to be region unit IDs, and matching those to the information in the `regionunitsfile`. The coordinates in the `regionsfile` are used by the meshing program to specify different regions in the surface model. The 3D problem is not yet supported.
  - For gravity or magnetics data, an intermediate mesh is not required and the data are calculated directly from the geometry of the surfaces in the wireframe model. To do so, the program must know which regions are on each side of every surface facet:
    - The first attribute in the `modelfile` should specify facet IDs. If the `modelfile` contains no attributes then default facet IDs of `1` are used.
    - Attributes `Region1` and `Region2` must be present in the `facetunitsfile`. Those attributes specify region unit IDs on each side of the surface facets. For each facet, the facet ID is taken from the `modelfile` and matched to an ID in the `facetunitsfile`, which provides the two region unit IDs on either side of the facet. Physical property values on either side of the facet are obtained from the `regionunitsfile` file. The difference (second minus first) is used to calculate the gravity or magnetic response. The ordering of the nodes in the facets is important ...
    - In 2D, if the nodes define an inward normal, using a right-hand-rule that crosses the vector along the line element facet (edge) and the y-axis, then the 1st region should be inside and the 2nd region outside. For example, if you have a single blob defined by a polygonal outline, a clockwise node ordering of the polygon in a +z down

coordinate system will define inward normals, in which case the 1st region would be the blob and the second the background.

* In 3D, if the nodes define an inward normal, using a right-hand-rule around the nodes as ordered, then the 1st region should be outside and the 2nd region inside. Another way to think of this is that a vector pointing from the 1st region into the 2nd region (across some face) should point in the same direction (i.e. positive dot product) as the normal vector defined by the node ordering.
* Program surface_normals can be used to help order the nodes in a wireframe surface model to aid with forward modelling.

15.3.6 Data misfit and trade-off parameters

`measure`

- set to `"ell2"` for the standard, sum-of-squares, ℓ_2 norm
- set to `"ekblom"` for the Ekblom measure
- set to `"huber"` for the Huber measure

`gamma`

- This parameter is a user-defined scalar that multiplies the data misfit term during an inversion.
- The value does not change during the inversion.

See

`lambdainit`

- In an inversion, the trade-off parameter(s) multiply the data misfit term(s).
- This parameter specifies the initial value of all trade-off parameters in an inversion.
- If this parameter is greater than zero then it takes precedence over the `lambdainit` parameter in the main input file for a `VIDI` inversion (see Section 16.4).

15.3.7 Sensitivity matrices and compression

`mtxroot`

- For linear forward problems: when forward modelling, the sensitivity matrix is usually calculated and written to file(s); when inverting, the sensitivity matrix is usually read from file(s).
- If `mtxroot="null"` or `=""` then: the sensitivity matrix is not written to file(s) in the forward modelling; the sensitivity matrix is calculated in an inversion, instead of being read from file(s).
- The files named `mtxroot_*.mtx` hold the sensitivity matrices (there are multiple files for tensor data).
- For scalar gravity data, and the magnetics problem with a susceptibility model and a single data component, there is only a single sensitivity matrix and the file is called `mtxroot.mtx`.
- For tensor gravity data, there are 6 matrices and the files are called `mtxroot_xx.mtx`, `mtxroot_xy.mtx`, etc. When forward modelling with `dosen` set to `"t"` (true), all of these sensitivity matrices are calculated and written to the different files.
- For the `"amp"` and `"atp"` magnetics model types with a single data component, there are 3 matrices and the files are called `mtxroot_x.mtx`, `mtxroot_y.mtx` and `mtxroot_y.mtz`. These should be in a +z down coordinate system. When forward modelling with `dosen` set to `"t"` (true), there is currently no way to calculate all of these sensitivity matrices at once. Instead, you have to run `FOGO` three times with the `"sus"` model type, and each time specify a different Earth's field direction for the three different Cartesian directions:
  - x-direction = northing: `dgeo` = 0, `igeo` = 0
  - y-direction = easting: `dgeo` = 90, `igeo` = 0
  - z-direction = down: `dgeo` = 0, `igeo` = 90.

  However, it is probably easier to just run an inversion and use the `mtxroot` parameter in the inversion input file: see Section 16.4.11.

- For the "pst" magnetics model type, there are 3 matrices and the files are called mtxroot_p.mtx, mtxroot_s.mtx and mtxroot_t.mtz. When forward modelling with dosen set to "t" (true), there is currently no way to calculate all of these sensitivity matrices at once. Instead, you have to run FOGO three times with the "sus" model type, and each time specify a different Earth's field direction for the three different components. The p-component should be in the direction of the Earth's field and the s- and t-components perpendicular to that, and all three orthogonal. A simple way to achieve this is to set the s-component direction to have the same declination as the Earth's field but with inclination rotated 90 degrees, then the t-component is horizontal and has declination rotated 90 degrees away from that of the Earth's field. For example:
  - p-direction: dgeo = 14.9, igeo = 64.8
  - s-direction: dgeo = 14.9, igeo = -25.2
  - t-direction: dgeo = 0, igeo = 104.9.

  However, it is probably easier to just run an inversion and use the mtxroot parameter in the inversion input file: see Section 16.4.11.

compmeth

- Set to "none" for no sensitivity compression.
- Set to "noco" for compression by reduced precision.
- Set to "wave" for wavelet compression, which uses parameters wavelet and tol.

compdir

- Set to "row" for sensitivity compression along rows (the most sensible choice for rectilinear meshes).
- Set to "col" for sensitivity compression along columns (the only sensible choice for unstructured meshes).
- **Too much compression of a sensitivity matrix can lead to spurious artifacts in models recovered from inversion. Hence, whenever sensitivity compression is used it is important to analyze the forward modelling errors associated with the compression and compare the magnitude of those errors against the uncertainties assigned to the observed data.**

wavelet

- If set to "null" then the default is used, which is "daub2".

tol

- For wavelet compression, wavelet coefficients with absolute value less than $|w_{max}| * \texttt{tol}$ are set to zero, where w_{max} is the coefficient with the largest absolute value in the row or column being compressed.

15.3.8 Seismic traveltime modelling options

recip

- If set to true ("t") then the sources and receivers are switched.
- The fast marching method will run faster the fewer sources you use
- (the number of receivers has a much smaller impact on the solution time).
- The data should be identical (within numerical error) due to the reciprocity theorem.

radius

- If radius > 0 then traveltimes at any nodes within that radius are calculated as slowness-times-distance using the slowness in the cell containing the source. This assumes homogeneity in the near-source region and helps reduce the modelling error.
- If $|$radius$| > 0$ then any rays being back-traced from receiver to source are traced directly to the source once they get within abs(radius) distance from the source. This is required because small numerical errors in the ray tracing means that rays may not actually hit the source exactly.
- When forward modelling, it is best to set radius > 0 and as large as possible as allowed for my your model such that the assumption of homogeneity still holds. However, this assumes homogeneity around the source so you may have to be careful when inverting with radius > 0.
- This parameter is not used when tracemode is straight.

tracemode

- If set to `"none"` then no ray-tracing is performed.
- If set to `"straight"` then straight rays are traced directly between source and receiver (make sure `dosen` is `"t"` (true) in this case).
- If set to `"source"` then rays are back-traced through the traveltime field and at sonic points it takes the direction that points the closest towards the source.
- If set to `"straightest"` then rays are back-traced through the traveltime field and at sonic points it takes the direction that is most similar to the previous ray direction.
- If set to `"exhaustive"` then rays are back-traced through the traveltime field and at sonic points all possible ray paths are tested and only the shortest path accepted (this option has the potential to be very time consuming).
- If tracing is performed then the traveltimes at the receivers are calculated as the traveltime integral along the trace path.
- If tracing is not performed then traveltimes are interpolated at receiver locations. The data will be about the same.
- The ray tracing in 3D is somewhat problematic. In 2D it is absolutely fine.
- If all you care about are the nodal traveltimes across the grid then you don't need to worry about this parameter (set to `"null"` or remove from input file).

`writetypes`

- The travel types are integer values that specify what kind of waves (traversal, head, diffraction) happened just before hitting a node:
  - 0 corresponds to near-source initialization,
  - 1 traversal along a tetrahedral edge element or triangular face (i.e. a 1D element),
  - 2 traversal across a tetrahedral face or triangular cell (i.e. a 2D element),
  - 3 traversal through a tetrahedral cell (i.e. a 3D element - only used in 3D).

`sloray`

- If rays are traced then this value is removed from the slowness model when calculating the traveltimes along the ray path. The ray paths themselves are still determined using the original slowness model.
- This can be helpful if you want to extract an anomalous component of the data.

15.3.9 Muography traveltime modelling options

The formats of the input files have been taken directly from the formats of the information given to me when I worked on muography data for the first time, with Barnoud et al. We can adjust these formats as desired in the future.

`method`

- The spherical formulation (`sph`) should be used for any production work.
- The `cart` option was used during research to provide a check of the `sph` option, but the `sph` option is superior.

`obsfile`

- This is a column-based file with each line of the format
  $x\ y\ z\ \phi\ \theta$
  where the first three columns specify the Cartesian coordinates of the corresponding receiver, and ϕ and θ define the azimuthal and altitude angles at the centre of the bin (solid angle).
- The `datafile` should have two additional columns:
  $x\ y\ z\ \phi\ \theta\ d\ \sigma$
  where d is the data value and σ is an assigned uncertainty.
- ϕ is the azimuthal angle in degrees from 0 towards north and positive east of north, i.e. our standard definition for declination.
- θ is the elevation in degrees from 0 horizontally to 90 vertically, i.e. opposite sign to our standard definition for inclination.

`receiversfile`

- This `.node` file should contain three coordinate columns for the receiver locations, plus two attribute columns specifying $\Delta\phi$ and $\Delta\theta$, the bin solid angle width and height respectively.
- There will be duplicated coordinate information in the `obsfile` and `receiversfile`. It is important that this information matches: each set of receiver coordinates in `obsfile` must match a set of receiver coordinates in `receiversfile`.

`nsub`

- If the method is `"sph"` then this specifies the number of integration sub-bins across a single data bin (solid angle).
- If the method is `"cart"` then this specifies the number of times to subdivide each mesh cell edge.
- If a negative value is provided and the method is `"sph"` then a default value of 128 is used for `nsub`.
- If a negative value is provided and the method is `"cart"` then a default value of 3 is used for `nsub`.

15.4 Output files

The output files are named `outroot*` with various suffixes and extensions, as listed below. The output file directory is the directory from which the program is being run, unless specified differently by adding path information to `outroot`.

`*.node OR *.ele`

- Contains the calculated data as the first attribute, named `VerticalGravity`, `GravityXComponent`, `GravityYComponent`, `GravityTensor`, `TotalFieldMag` or `TravelTime` depending on the data type.
- A `.ele` file is used for traveltime data and a `.node` file for all others.
- For traveltime data, the two index columns in the `.ele` file specify the source/receiver pairs for each traveltime.

`*.vtu`

- As above but for viewing the data response in ParaView.

`*_sen.vtu`

- Holds the model and another cell attribute equal to the ℓ_2 norm along each sensitivity matrix column (each cell has a value corresponding to a specific matrix column).
- Only written if parameter dosen is set to true (`"t"`).

For traveltime data, the following additional files are written:

`*_traces.*`

- Hold seismic traces for each source-receiver pair.
- Only written if parameter `tracemode` is not `"none"`.

`*_ttimes.vtu`

- Only written if either parameter `writettimes` or `writettypes` is `"t"` (true).
- If `writettimes` is `"t"` (true) then this file holds the traveltimes at all grid nodes in the modelling grid.
- If `writettypes` is `"t"` (true) then this file holds the traversal types at all grid nodes in the modelling grid:
  - 0 means the node was initialized with a traveltime
  - 1 means a traversal along a tetrahedral edge element or triangular face
  - 2 means a traversal across a tetrahedral face or through a triangular cell
  - 3 means a traversal through a tetrahedral cell (only used in 3D)

Chapter 16

Mesh-based physical property and lithological inversion: VIDI (formerly VINV)

16.1 Program summary

VIDI (**V**oxel**I**zed **DI**scretization) is a program for mesh-based inversion that determines the physical property values or lithologies inside the mesh cells. This is a flexible, modular and highly functional program:

- The discretization is voxellized (many space-filling cells) on a 2D or 3D rectilinear or unstructured (triangular or tetrahedral) mesh. See Section 16.4.2 for related input file parameters.
- Two different inversion approaches are available that use very different optimization algorithms:
  - the inversion can treat the model values as physical properties on continuous, but possibly bounded, ranges
  - a lithological inversion can be designed such that only specific physical property values are allowed for each of the known or assumed rock types.
- The supported geophysical data types are:
  - vertical component (scalar) gravity data
  - gradiometry (tensor) gravity data
  - total magnetic field data
  - magnetic amplitude data
  - first-arrival seismic traveltimes.
- Single or multiple data-types can be inverted (independent or joint inversion). See Section 16.4.8 for related input file parameters. Joint inversion can include:
  - multiple geophysical data types responsive to a single physical property
  - a single geophysical data type responsive to multiple physical properties
  - multiple geophysical data types responsive to multiple physical properties.

  There are various possible joint coupling strategies for any pair of physical property models:
  - explicit linear relationship (a specific mathematical relationship between two physical properties)
  - implicit linear relationship (based on the concept of correlation from statistics)
  - structure-based coupling using the cross-gradient measure
  - petrophysical coupling using fuzzy c-means to specify clusters
  - petrophysical coupling using a combination of Gaussian functions.
- There are many options for regularization. See Section 16.4.5 for related input file parameters. Regularization includes the traditional smallness and smoothness/roughness measures with general norms. The smoothness axes can be rotated to any arbitrary orientations. See Section 16.5.2 for a mathematical description of the model objective function.
- For magnetic inversion there are several possible forms:
  - inversion of total field data for a susceptibility model (a scalar quantity in each mesh cell)
  - inversion of magnetic amplitude data for an effective susceptibility model
  - inversion of total field data for a magnetization model (a vector quantity in each mesh cell)

For the latter there are further options for the format of the magnetization model:

- Cartesian components in arbitrary directions (typically one is in the direction of the Earth's field)
- spherical/polar coordinates, i.e. magnetization amplitude plus orientation angle(s).

See Section 16.4.4 for related input file parameters.

- There are many options for controlling the minimization. See sections 16.4.9 and 16.4.10 for related input file parameters.

This program will run in parallel. Remember to set the OMP_NUM_THREADS environment variable. On a Mac you use the following command in the terminal window before running this program:

```
> export OMP_NUM_THREADS=value
```

16.2 Running the program

16.2.1 Command line usages

```
vidi inputfile

vidi inputfile outputroot

vidi inputfile outputroot sensibility

vidi inputfile outputroot sensibility restart

vidi inputfile outputroot sensibility restart seed
```

16.2.2 Command line parameters

`inputfile`

- The name of the input file (the format is described below).
- See the note at the start of Chapter 3 regarding relative file paths.

`outputroot`

- The names of the output files have this root (the output files are described below).
- If `outputroot` is absent then it is set to the root of the input file, that is, the input file name with extension and any path information stripped off: for example, if `inputfile` is set to `mydirectory/myfile.ext` then the output files will be named `myfile*` and placed in the current working directory.

`sensibility`

- Set to `"t"` (true) to perform sensibility checks on the input files without actually running the inversion.
- The inversion will stop after calculating the misfit and model objective measures for the initial model. This can be helpful if you simply want to calculate that information for a particular model.

`restart`

- Set to `"t"` (true) to restart an inversion.
- This simply appends the output to any existing `.log` and `.aux` output files. It does not automatically determine which trade-off value to start from or which initial model to use: you must specify that information manually in the input files.

`seed`

- Set to some integer value to use as the seed for random number generation.
- This is only important if global optimization is used, not for local optimization.

16.3 Outputs

16.3.1 Output files

The output files are named `outroot*` with various suffixes and extensions, as listed below. The output file directory is the directory from which the program is being run, unless specified differently by adding path information to `outroot`.

There are many possible output files. For each dataset there will be predicted data files written in `.node` or `.ele` format, and in `.vtu` format. Those file names will contain the geophysical data type name, e.g. `"vertical_gravity"` or `"total_field_mag"`, and the data residuals calculated as predicted data minus observed data, divided by the data uncertainties.

If there are multiple physical property models or multiple model components involved, e.g. for a magnetization inversion or a joint inversion, the output files named `outputroot_models.*` contain those models; for a single dataset inversion with a single physical property model, those files are named similarly but with `"models"` replaced by the physical property name(s), e.g. `"density"` or `"magsus"`. The contents of the output files should be easy to assess by these naming strategies.

For joint inversions, the output `outputroot_models.*` file contains the physical property models and also the joint coupling constraint vector information. If there is only a single joint coupling measure then attribute `"UnscaledConstraintVector"` contains the constraint vector values before any weighting, and `"ScaledConstraintVector"` is multiplied by the `wjvalue` but **not** any weights specified in the `wjfile`. If there is more than one joint coupling measure then attribute `"CombinedConstraintVector"` contains a sum of the constraint vectors weighted by the corresponding `wjvalue` and `rho` values but **not** any weights specified in the `wjfile`.

For a magnetization inversion, the model names are as follows:

- `MagSus` is the effective magnetic susceptibility (vector magnitude divided by Earth's field strength)
- `MagP`, `MagS` and `MagT` are the vector components with the p-component in the direction of the Earth's field and the other two perpendicular.
- `MagST` is the RMS of the s- and t-components.
- `MagX`, `MagY` and `MagZ` are the vector components in a +z-down coordinate system (+x is North and +y is East).
- `MagQ` is an approximate Koenigsberger ratio calculated as the RMS of the s- and t-components divided by the absolute value of the p-component.
- `MagInc` and `MagDec` are the inclination and declination angles: inclination is zero at horizontal and positive below horizontal; declination is zero to the North and positive East of North.

For inversion on rectilinear meshes, the files containing the recovered models are named `*.txt` and contain multiple columns of values with a first header line indicating the model attributes in each column. These files can not be read directly by the UBC-GIF model visualization program MeshTools3d. You will need to extract a single column of the file into another file: you can use program xyz2xyz for this purpose.

16.3.2 Log file and terminal output

All information displayed in the terminal/command window during execution is written to the output log file. Below is a brief description of some of the items displayed:

- `totit` - the total iteration number increments each time the lambda trade-off parameter(s) change(s)
- `stage` - the stage of a joint inversion (the coupling weights are increased each stage)
- `lambit` - like `totit` but it resets to zero when `stage` increments
- `objfun` - the total objective function value
- `lambda1` (and `lambda2` if a joint inversion) - the trade-off parameter value(s)
- `omega1` (and `omega2` if a joint inversion) - omega is equal to the misfit divided by the target misfit
- `norm1` (and `norm2` if a joint inversion) - the model objective function value(s)
- `gtg` - the gradient norm
- `slope` - the slope of the objective function in the direction of the model perturbation
- `alpha` - the length of model perturbation taken
- `nact` - the number of active bound constraints (the number of model parameters that have hit their bounds)
- `ncgi` - the number of iterations taken in the conjugate gradient algorithm when calculating the model perturbation direction
- `rcg` - the residual for the conjugate gradient solution for the model perturbation.

Additional items are displayed for joint inversions:

- `alphaj` - the `alphaj` value used

- `rho ...` - the current value(s) of the `rho` parameter(s) being heated
- `jnorm` - the sum of the joint coupling measure(s) after all multipliers (`alphaj`, `rho` and joint coupling weights) have been applied
- `jnormi ...` - the individual joint coupling measure(s) with joint coupling weights applied but without other multipliers applied (`alphaj` and `rho`).

16.4 Input file format

Each line of the input file should be of the format

```
name value
```

where `name` is the name of some inversion parameter and `value` is the value for that parameter.

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

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

The possible parameters and default values are listed in the table below. The capitalized headings in the table link to sections where the parameters are discussed in more detail.

Name	Default	Brief description
`revision`	`0`	a code revision number
		MESH INFORMATION
`meshinp`	`""`	a model parameterization specification file
		PHYSICAL PROPERTY INFORMATION
`propinp`	`""`	input file for a particular physical property
		DATA-RELATED INFORMATION
`datainp`	`""`	input file for a particular geophysical data set
`normalized`	`"t"`	if `"t"` (true) then the data misfit term(s) are normalized by the number of data
`normalizec`	`"t"`	if `"t"` (true) then the data misfit term(s) are normalized by the `chifact` value(s)
		REGULARIZATION
`normalizem`	`"f"`	if `"t"` (true) then the regularization terms are normalized by the mesh volume
`rotate`	`"f"`	set to `"t"` (true) to rotate the smoothness axes
`maskfile`	`""`	file that masks the regularization across the mesh
`maskindex`	`""`	attribute index for the `maskfile`
`wmfile`	`""`	file containing smoothness weights and axis rotation information
`wmindex`	`0`	attribute index for the `wmfile` smoothness weights file
`wxindex`	`0`	attribute index for the `wmfile` for cell-centred smoothness weights in x-direction
`wyindex`	`0`	attribute index for the `wmfile` for cell-centred smoothness weights in y-direction
`wzindex`	`0`	attribute index for the `wmfile` for cell-centred smoothness weights in z-direction
`wnindex`	`0`	an alias for `wxindex` (“n” for north)
`weindex`	`0`	an alias for `wyindex` (“e” for east)
`wvindex`	`0`	an alias for `wzindex` (“v” for vertical)
`strikeindex`	`0`	attribute index for the `wmfile` for the strike rotation angle
`dipindex`	`0`	attribute index for the `wmfile` for the dip rotation angle
`tiltindex`	`0`	attribute index for the `wmfile` for the tilt rotation angle
`strikevalue`	`0.0`	strike rotation angle for the entire mesh
`dipvalue`	`90.0`	dip rotation angle for the entire mesh
`tiltvalue`	`0.0`	tilt rotation angle for the entire mesh
`gradtol`	`0.0`	tolerance on min. vertex/dihedral angle when generating the gradient operators
`alpham`	`1.0`	across-face smoothness regularization multiplier (see parameter `rotate` for usage!)
`alphax`	`1.0`	smoothness regularization multiplier for x-direction (see parameter `rotate` for usage!)
`alphay`	`1.0`	smoothness regularization multiplier for y-direction (see parameter `rotate` for usage!)
`alphaz`	`1.0`	smoothness regularization multiplier for z-direction (see parameter `rotate` for usage!)
`alphan`	`1.0`	smoothness regularization multiplier for north/south direction (an alias for `alphax`)
`alphae`	`1.0`	smoothness regularization multiplier for east/west direction (an alias for `alphay`)
`alphav`	`1.0`	smoothness regularization multiplier for vertical direction (an alias for `alphaz`)
`measure0`	`"ell2"`	specifies the type of measure for the smallness regularization term
`measure1`	`"ell2"`	specifies the type of measure for the smoothness regularization term

Name	Default	Brief description
ekblomp	2.0	the p-value for the Ekblom measure or total-variation measure
ekblome	1.0E-9	the epsilon value for the Ekblom measure or or total-variation measure
smooth_mod_diff	"f"	specifies whether or not the reference model appears in the smoothness terms
smooth_wts_inside	"f"	specifies where the smoothness weights should appear in the smoothness terms
		CONSTRAINTS
lowerunit	0	lower bound for rock-unit IDs when discrete is "t" (true)
upperunit	0	upper bound for rock-unit IDs when discrete is "t" (true)
		JOINT INVERSION
alphaj	0.0	multiplier on the sum of joint measures
jointinp	""	input file for a particular joint coupling
stageinit	0	the joint inversion will start at this stage
searchr	"t"	set to false ("f") to avoid ratio search for lambda
jchitol	0.05	relative tolerance on the joint pareto misfit
		OPTIMIZATION
optiminp	""	input file for optimization engine options
doparinv	"t"	determines where parallelization occurs
		TIKHONOV SEARCH
lambdainit	0.0	initial lambda trade-off parameter value
minlambdasteps	4	minimum number of steps in lambda-search
maxlambdasteps	48	maximum number of steps in lambda-search
lambdafactmin	1.01	minimum multiplication factor when adjusting lambda
lambdafactmax	2.0	maximum multiplication factor when adjusting lambda
lambdamult	1.0	increasing this factor will lead to larger adjustments when close to the target
ratiomult	1.0	increasing this factor will lead to larger adjustments when close to the target
tikinterp	"f"	set to "t" (true) to attempt interpolation for tradeoff parameter
		OUTPUTS
writeinter	"t"	set false to not output inversion results at intermediate iterations
totitprefix	"f"	set to "t" (true) to adjust prefix of intermediate output files to indicate total iteration #
mtxroot	""	set to some non-empty file root to write sensitivity matrix file(s) (linear problems only)

16.4.1 revision

The `revision` parameter is used to assess whether you are re-running an inversion that was originally run on an older version of the program. If the specified revision number is less than the built-in version number of the program then a warning message may be issued and the user will be prompted to continue or quit. The idea is that you will specify the program revision number at the time that you set up the inversion. Then, if at a future date some changes have been made to the program defaults, you will be informed of the changes. Otherwise, your work may not be reproducible unless you change your older input files.

For users in my research group, you can simply run `svn update` and use the quoted revision number. For public users, your SVN revision numbers are different because you are using a different repository. Therefore you will have to refer to the change log. If in doubt, you can always put a huge revision number into your files, e.g. 1000000.

16.4.2 Mesh-related parameters

The mesh can be a 2D or 3D rectilinear or unstructured (triangular or tetrahedral) mesh.

`meshinp`

- A model discretization specification file.
- See the documentation on model discretization specification files for more information on the file format.

16.4.3 Physical property-related parameters

`propinp`

- Each physical property has its own input file (`propinp`).
- Specify the `propinp` parameter as many times as required.
- See the documentation on physical property files for more information on the file format.

16.4.4 Data-related parameters

`datainp`

- Each dataset has its own input file (`datainp`).
- Specify the `datainp` parameter as many times as required.
- See the documentation for program `FOGO` for the data-specific parameters.

`normalized`

- If `"t"` (true, the default) then the data misfit term(s) are normalized by the number of data.
- This normalization occurs in the objective function and is most helpful in a joint inversion when trying to balance multiple geophysical data sets.

`normalizec`

- If `"t"` (true, the default) then the data misfit term(s) are normalized by the `chifact` value(s).
- If any of those values are zero, which can occur during testing or when using a global optimization, then this normalization does not occur.
- This normalization occurs in the objective function and is most helpful in a joint inversion when trying to balance multiple geophysical data sets.

16.4.5 Regularization parameters

Some regularization parameters are specified in the individual physical property definition files instead of the inversion input file. See the documentation on physical property files for more information.

Some regularization parameters specify directions: pay careful attention to the Cartesian coordinate system definitions in Section 16.4.6.

`rotate`

- For rectilinear meshes, if `rotate` is `"f"` (false, the default) then there are three smoothness operators that act along each Cartesian axis direction; those operators calculate model differences across cell faces. If `rotate` is `"t"` (true) then those smoothness operators can be rotated to some arbitrary orthogonal orientations, in which case the operators no longer calculate differences across cell faces but rather look at the spatial gradients in small packages of cells (see Lelièvre & Oldenburg, 2009). In either case, parameters `alphax`, `alphay` and `alphaz` are used to weight the three different smoothness terms. Refer to the equations for the model objective function in Section 16.5.2.
- For unstructured meshes, if `rotate` is `"f"` (false, the default) then there is a single smoothness operator that calculates differences across cell faces and parameter `alpham` is the associated weight. If `rotate` is `"t"` (true) then there are three smoothness operators that look at the spatial gradients in small packages of cells (see Lelièvre & Farquharson, 2013) and parameters `alphax`, `alphay` and `alphaz` are used to weight the three different smoothness terms. **Hence, with unstructured meshes, you must specify different weighting parameters depending on the value specified for parameter `rotate`.** Refer to the equations for the model objective function in Section 16.5.2.
- See the notes below for parameter `gradtol` if using rotated smoothness axes on unstructured meshes.
- Section 16.4.6 provides further instructions on how to perform an inversion with the smoothness axes rotated.
- If you wish to use the total variation measure for the smoothness term, you must set `rotate` to `"t"` (true) if you are using an unstructured mesh. However, the `alphax/y/z` parameters are ignored (refer to the notes below for parameter `measure1`).

`maskfile, maskindex`

- The `maskfile` specifies the mask for the regularization across the mesh. You will want to mask the regularization if, for example, you have a rectilinear mesh with cells above the topography surface, for which program `toposplit` can be used to generate the required `maskfile`. In that situation, if you do not mask the regularization then smoothness values will be calculated across the earth-air interface.
- The `maskfile` should be a `.ele` format file or a simple text file with one or more columns of values.
- Attribute index `maskindex` of file `maskfile` is used as the mask. Cells with values ≤ 0 are not included in the regularization. For example, no smoothness regularization occurs across a mesh face if either adjacent cell has value ≤ 0.

`wmfile, wmindex`

- The `wmfile` specifies the smoothness weights. This should be a `.ele` format file or a simple text file with one or more columns of values.
- If `rotate` is `"f"` (false, the default) then, for either rectilinear or unstructured meshes, the number of required weights is equal to the number of internal faces in the mesh (faces other than those lying on the outer boundary of the mesh). The `wmfile` should list the smoothness weights for the easting, northing and then vertical directions:

$$
\begin{array}{c}
w_{x1} \\
w_{x2} \\
\ldots \\
w_{xL} \\
w_{y1} \\
w_{y2} \\
\ldots \\
w_{yM} \\
w_{z1} \\
w_{z2} \\
\ldots \\
w_{zN}
\end{array}
$$

  where L, M and N denote the number of internal mesh faces in each Cartesian direction in the mesh. For an unstructured mesh, the ordering of the weights should be as specified in the `.faces` file generated by TetGen. For a rectilinear mesh, the ordering of the weights should be as in the UBC-GIF format for smoothness weighting files. Program `make_face_weights` writes a file in the correct format, with the correct ordering, and may be of help.
- For legacy considerations, if `rotate` is `"f"` (false) and you are using a rectilinear mesh then the `wmfile` may contain smallness weights w_s before the smoothness weights:

$$
\begin{array}{c}
w_{s1} \\
w_{s2} \\
\ldots \\
w_{sK} \\
w_{x1} \\
\ldots \\
w_{zN}
\end{array}
$$

  where K denotes the number of mesh cells. That is the standard UBC-GIF format for smoothness weighting files. However, any smallness weights present in the `wmfile` are ignored by program `VIDI` when generating the smoothness operators. Instead, the smallness weights should be specified in the physical property input files using the `wsfile` parameter. This behaviour allows you to specify the same file for both the `wsfile` and the `wmfile`, in which case all the weights in the file will be used by `VIDI`.
- If `rotate` is `"t"` (true) then, for either rectilinear or unstructured meshes, the number of required weights is equal to the number of cells in the mesh. In this case, the `wmfile` can be built as you would any other model on your mesh. For an unstructured mesh, parameter `wmindex` specifies which attribute in the `.ele` file, or which data column in the column-based data file, to use for the weights. For a rectilinear mesh, if `rotate` is `"t"` (true) then the file should be in the standard UBC-GIF model format, with one value for each cell in the mesh, and each value on a new line.
- If `wmfile` is specified as `"null"` then the smoothness weights are set to 1.0 for the entire mesh.

`wxindex, wyindex, wzindex, strikeindex, dipindex, tiltindex, strikevalue, dipvalue, tiltvalue`

- These parameters are only used if `rotate` is `"t"` (true).
- Parameters `wxindex`, `wyindex` and `wzindex` specify the columns in the `wmfile` is used for the cell-centred smoothness weights for the three Cartesian directions.
- The `strikeindex`, `dipindex` and `tiltindex` columns in the `wmfile` are used to specify the rotation of the smoothness axes.
- If `wmfile` is specified as `"null"` then `strikevalue`, `dipvalue` and `tiltvalue` specify the rotations for the entire mesh.
- The rotation parameters are only used if rotate is set to `"t"` (true)).

`gradtol`

- When using rotated smoothness axes on unstructured meshes, if the minimum vertex (2D) or dihedral (3D) angle defined by the cell centroids in a particular package of neighbouring cells is less than this tolerance then an approximate gradient

calculation is used for that package of cells. The alternative would be to remove that package of cells from the gradient calculation completely. However, if many such packages of cells were removed from the calculation and `alphas` = 0 then some cells may be completely absent from the regularization, leading to an ill-posed problem. Hence, the code uses the approximation option as necessary to avoid ill-posed problems.

- If `gradtol` is set too low, e.g. using the default value of 0, poor quality meshes may lead to division by small values in the gradient calculations, possibly leading to artifacts in recovered models. If `gradtol` is set too high, the approximate gradient calculation may lead to inversion results that do not follow the intended character, e.g. orientations in incorrect directions. Hence, when using rotated smoothness axes on unstructured meshes, users should attempt to use meshes with as good quality as is feasible (higher quality often means more cells) such that the value of this parameter can be set low.
- You will need to experiment with the `gradtol` parameter. If the mesh is good quality then it can be set fairly low. Maybe even 0. Try an inversion with a low value, e.g. 0, and if there are clearly spurious artifacts then try it higher. If you go too high then you won't be using true gradient operators and the result may not recover the orientations that you prescribed. It is a trade-off.

`measure0, measure1`

- set to `"ell2"` for the standard, sum-of-squares, L2 norm
- set to `"ekblom"` for the Ekblom measure; parameters `ekblomp` and `ekblome` then come into effect
- set to `"totvar"` for the total variation measure; parameters `ekblomp` and `ekblome` then come into effect; can only be used for the `measure1` and then only if directional smoothness operators are being used; parameters `alphax/y/z` are ignored when the total variation measure is used.

16.4.6 Instructions for rotating the smoothness directions

If you want to emphasize pipe- or rod-like features, you will want to rotate the smoothness axes so that they measure the smoothness along the major and minor axes of the desired features. Set parameter `rotate` to `"t"` (true). If using an unstructured mesh, pay attention to parameter `gradtol`.

If you want to specify a homogeneous orientation across the entire inversion mesh then set parameter `wmfile` to `"null"` and use parameters `strikevalue`, `dipvalue` and `tiltvalue` to specify your desired orientation. If you want to specify heterogenous orientations then construct an appropriate file for use with parameter `wmfile` and use parameters `strikeindex`, `dipindex` and `tiltindex` to specify your desired orientations. When you rotate the smoothness axes you end up with gradient operators that calculate quantities at cell centres instead of at the faces between cells. This means that your w`mfile` specifies orientations and weights for every cell in the model. You can therefore build that weighting information like you would any other model on your mesh.

If you want to specify homogeneous smoothness weights across the entire inversion mesh then use parameters `alphax`, `alphay` and `alphaz` to specify the weights for the three rotated smoothness directions. If you want to specify heterogenous smoothness weights then use parameters `wxindex`, `wyindex` and `wzindex`.

For 2D, the only angle used is the dip. The coordinate system definition used inside the code has $+x$ right and $+z$ down. A dip of zero is along the x-axis, to the right. A positive dip is a clockwise rotation from $+x$ towards $+z$. Hence, a dip of 90 degrees is downwards.

For 3D, the strike is the clockwise angle from north towards east: a zero strike is north, a 90 degree strike is east. If you are standing and looking in the strike direction then the dip is an angle to your right below horizontal: a dip of 0 is horizontal, a dip of 90 degrees is straight down. The coordinate system definition used inside the code has $+x$ north, $+y$ east and $+z$ down. The rotations define a new set of rotated axes with the rotated x-axis along the strike direction and the z-axis in the down-dip direction. **The parameters `alphax`, `alphay` and `alphaz` are weights along the rotated smoothness axes. If the axes are not rotated, then `alphax` controls the smoothing in the north-south direction, `alphay` in the east-west direction, and `alphaz` vertically.**

A positive tilt is a rotation around the new y'-axis that moves x' towards z'. If the dip is 90 degrees then a small positive tilt rotation moves the x' axis beneath the ground surface. i.e. it is a plunge below horizontal.

Refer to Section 16.5.2 for a description of the rotated model objective function. Here I define w_x to be a particular cell's weight for the x-smoothness direction, with similar definitions for w_y and w_z. If you want to emphasize pipe-like features, you rotate the smoothness axes so that the z-smoothness axis is along the major axis of the pipe and you set

$$w_x = w_y$$
$$w_z \gg w_x$$

which specifies higher smoothness along the pipe major axis, the result being an elongation of features along the major axis.

If you want to emphasize plate-like (or disk-like) features, you rotate the smoothness axes so that the z-smoothness axis is perpendicular to the plate and you set

$$w_x = w_y$$
$$w_z \ll w_x$$

which specifies lower smoothness perpendicular to the plate, the result being elongation parallel to the plate and sharp jumps perpendicular to it.

You may have to experiment with the ratio w_z/w_x to obtain appropriate results. I generally start with a ratio of 10 for pipe-like features and 0.1 for plate-like features.

If you want to emphasize features that do not contain symmetry around a major or minor axes then you will generally have

$$w_x \neq w_y \neq w_z.$$

16.4.7 Constraint parameters

Some constraint parameters are specified in the individual physical property definition files instead of the inversion input file. See the documentation on `physical property files` for more information.

16.4.8 Joint inversion options

`alphaj`

- The multiplier on the sum of joint measures is typically only set to 1.0 or 0.0.
- Use 1.0 to include the coupling specified in the `jointinp` files.
- Use 0.0 to exclude any coupling. This can be helpful before a joint inversion to assess relative weights for each model norm term before running a coupled joint inversion with `alphaj` set to 1.0: see parameter `alphab` in Section 3.4 and the model objective function Section 16.5.2.

`jointinp`

- Each joint coupling measure you wish to include has its own input file (`jointinp`).
- Specify the `jointinp` parameter as many times as you have different couplings (pairs of coupled models).
- See the documentation on `joint coupling measure files` for more information on the file format.

`stageinit`

- A joint inversion proceeds in several stages, heating the coupling multiplier values in the objective function (see ρ_k in Section 16.5.3) from 0.0 at stage 0 up to the final value `rho` at stage `nsteps` (both specified in the `jointinp` files). You may wish to skip stage 0, or you may need to use the `stageinit` parameter to restart a joint inversion.

  A joint inversion proceeds in several stages, heating the `rho` values in the objective function (see Section 16.5.3) from 0.0 at stage 0 up to the specified final value the `jointinp` file. This heating occurs over `nsteps` stages (also specified in the `jointinp` file.
-

`searchr`

- If set to `"f"` (false) to avoid ratio search for lambda then the final misfits may not be as close to their targets as requested.

The suggested standard procedure for running joint inversions is:

1. Run the independent inversions.
2. Run a joint inversion with no coupling (set `alphaj` to 0.0) and compare to the results from the previous step. Pay particularly attention to the model objective function values `norm1` and `norm2` specified in the output `.log` files. If the results are significantly different then the model objective function terms may need to be differentially weighted in the joint inversion: see parameter `alphab` in Section 3.4 and the model objective function Section 16.5.2.
3. Re-run the joint inversion with no coupling, as necessary to assess appropriate relative weights for each model norm term.
4. Run a joint inversion with coupling, using the result from the previous step to initialize. This means setting `stageinit` to 1, setting appropriate values for `lambdainit` in the data input files (see Section 15.3) and specifying the initial models in the `physical property input files`.

16.4.9 Optimization options

`optiminp`

- See the documentation on optimization engine specification files for more information on the file format.

`doparinv`

- If `doparinv` is `"t"` (true, the default) then the GA or PMOGA algorithm (if used) calculates the objective function for each candidate solution in parallel. This means that many forward modelling solutions occur in parallel. If `doparinv` is `"f"` (false) then the parallelization will only occur inside the forward modelling solution (if the forward modelling of a given type data is also parallelized), and only one forward modelling solution occurs at any one time.

`discrete`

- This parameter is specified in the `optiminp` file.
- If `"f"` (false) then the physical properties in each cell are continuous variables (between bounds). This is always the approach used when `engine` is set to `"local"`.
- If `"t"` (true) then the model values are integers representing rock-unit IDs that are linked to physical properties via the `regionunitsfile` specified in the `meshinp` file (see Sections 3.7 and 15.3.5 for more information). This option is only used when `engine` specifies a global optimization, i.e. set to `"pso"`, or `"ga"`. Bounds, specified using `lowerunit` and `upperunit`, should be used to limit the model values to the allowed range of rock-unit IDs and all rock-unit IDs must represent the set of all integers between and including those bounds.

16.4.10 Tikhonov search options

These parameters are only used if performing a single-objective optimization, i.e. `multiobj` is `"f"` (false) in the `optiminp` file. Currently, they are not used unless `engine` is `local`.

`lambdainit`

- When there are multiple datasets, all lambda values are set to `lambdainit` by default.
- However, any lambda values specified in the data input files take precedence: see parameter `lambdainit` in Section 15.3.

`tikinterp`

- If `"t"` (true) then the inversion will attempt an interpolation for the tradeoff parameter once straddling the target misfit value. This will only occur once, after which the usual heating search will take over.
- If `"t"` (true) then the other Tikhonov search options will be ignored until after the single interpolation has been performed.
- Currently, only a linear interpolation in log-log space is supported.
- If there is more than one dataset then this input parameter is ignored.

16.4.11 Output options

`writeinter`

- For very fast inversions, your machine's runtime library I/O buffer may not flush between the different intermediate inversion iterations. This means that if an output file is open by another process, e.g. in a text editor, the file may not be written correctly each iteration. However, measures have been put in place such that the final output file will be written correctly: the approach I have taken is to pause execution by one second, allowing the I/O buffer to catch up - this should work on all but the most archaic computers (think vacuum tubes) but if you want to be safe, don't have the output files open while running an inversion.

`totitprefix`

- The names of the files are `*_totit###.*` where `###` is an integer value indicating the iteration number (use the log file to cross-reference with stage or lambda iteration numbers).

`mtxroot`

- The names of the output sensitivity matrix file(s) are `mtxroot*.mtx`.
- These are overwritten if they exist already.

- This is helpful if you plan to run multiple inversions with the same sensitivity matrix. You could use program `FOGO` to calculate the sensitivity matrix. However, by using this parameter you can have that happen automatically on the first inversion you run, then alter the `physical property input files` to specify that matrix file for later inversions.

16.5 The objective function

The objective function for an inversion with a single data type and a single causative physical property model is

$$\Phi = \lambda\gamma\Phi_d + \Phi_m \tag{16.1}$$

The objective function for an inversion with a single data type and multiple physical property models (for example, inversion of total field magnetic data for three components of the vector magnetization) is

$$\Phi = \lambda\gamma\Phi_d + \Phi_{m1} + \Phi_{m2} + \dots \tag{16.2}$$

The objective function for a joint inversion with two data types and a single physical property model (for example, inversion of vertical gravity and gravity gradiometry data for density) is

$$\Phi = \lambda_1\gamma_1\Phi_{d1} + \lambda_2\gamma_2\Phi_{d2} + \Phi_m \tag{16.3}$$

In this situation, the inversion attempts to fit both data sets to their assigned targets by altering the two λ (lambda) trade-off parameters.

The objective function for a joint inversion with two data types and two physical property models (for example, inversion of vertical gravity data for density, and total field magnetic data for magnetic susceptibility) is

$$\Phi = \lambda_1\gamma_1\Phi_{d1} + \lambda_2\gamma_2\Phi_{d2} + \Phi_{m1} + \Phi_{m2} + \Psi \tag{16.4}$$

In this situation, the inversion attempts to fit both data sets to their assigned targets by altering the two λ (lambda) trade-off parameters. Function Ψ is the joint coupling term.

The objective function for a joint inversion with more than two data types is

$$\Phi = \lambda\left(\gamma_1\Phi_{d1} + \gamma_2\Phi_{d2}\right) + \Phi_{m1} + \dots \tag{16.5}$$

In this situation, the inversion does not attempt to fit both data sets to their assigned targets.

The γ (gamma) values are user-defined constants specified in the data input files: see section 15.3. Note the λ (lambda) trade-off parameters multiply the data misfit terms, Φ_{d1} and Φ_{d2}, not the model objective terms as is sometimes seen in literature. The initial values of the trade-off parameters can be specified in the inversion and data input files: see sections 15.3 and 16.4.

16.5.1 The data misfit terms

Each misfit term is of the form

$$\Phi_d = \frac{1}{N\,C}\mathbf{e}^{\mathrm{T}}\rho\Big(\mathbf{W}_d\left(\mathbf{d}_{pred} - \mathbf{d}_{obs}\right)\Big)$$

where N is the number of data, C is the target `chifact` value, $\mathbf{e}$ is a vector of ones, $\mathbf{W}_d$ is a diagonal matrix holding the inverses of the assigned uncertainties, $\mathbf{d}_{pred}$ is the predicted data (the forward modelled response of some candidate model), $\mathbf{d}_{obs}$ is the observed data, and the function ρ defines a particular measure: $\rho(\dots)$ is a vector. See below for more information on the possible options for the function ρ. When an ℓ_2-norm is used, the result is the classic χ-squared data misfit:

$$\Phi_d = \frac{1}{N}\sum_{i=1}^{N}\frac{\left(d_{i,pred} - d_{i,obs}\right)^2}{\sigma_i^2}$$

where σ_i is the uncertainty assigned to the i^{th} data measurement. The normalization by the number of data and chifact value can be removed using the `normalized` and `normalizec` parameters in the inversion input file: see section 16.4.

16.5.2 The model objective function

The model objective function is

$$\Phi_m = \Phi_s + \Phi_f + \Phi_x + \Phi_y + \Phi_z + \Phi_t$$

Each term is of the form

$$\Phi_- = \alpha \mathbf{v}^{\mathrm{T}} \mathbf{W} \rho(\boldsymbol{\omega})$$

where $\mathbf{v}$ holds integration volumes, $\mathbf{W}$ is a diagonal matrix of weights, $\boldsymbol{\omega}$ is some function of the model $\mathbf{m}$, and the function ρ defines a particular measure: $\rho(\boldsymbol{\omega})$ is a vector. The options for the $\rho(\boldsymbol{\omega})$ function are the ℓ_2-norm

$$\rho(\omega) = \omega^2$$

and the Ekblom measure

$$\rho(\omega) = \left(\omega^2 + \epsilon^2\right)^{p/2}$$

which reduces to the ℓ_2-norm for $p = 2$ and $\epsilon = 0$.

The smallness term is

$$\Phi_s = \alpha_b \alpha_s \mathbf{v}_c^{\mathrm{T}} \mathbf{W}_s \rho(\boldsymbol{\omega}_s)$$
$$\boldsymbol{\omega}_s = \mathbf{m} - \mathbf{m}_{ref}$$

where

- $\mathbf{m}_{ref}$ is a reference model (set to zero if absent),
- $\mathbf{v}_c$ holds cell volumes,
- $\mathbf{W}_s$ is a diagonal matrix containing any user-defined smallness weights combined with any depth, distance or sensitivity-based weights.

The remaining terms are all smoothness terms. **Only one may be used!** The first is currently only used for unstructured grids:

$$\Phi_f = \alpha_b \alpha_m \mathbf{v}_f^{\mathrm{T}} \mathbf{W}_f \rho(\boldsymbol{\omega}_f)$$
$$\boldsymbol{\omega}_f = \mathbf{D}_f \mathbf{m}$$

where

- $\mathbf{D}_f$ is a general difference matrix across cell faces,
- $\mathbf{v}_f$ holds the appropriate integration volumes,
- $\mathbf{W}_f$ is a diagonal matrix containing any user-defined smoothness weights combined with any depth, distance or sensitivity-based weights.

The next three smoothness terms are always used for rectilinear grids and are only used for unstructured grids when rotating the smoothness directions:

$$\Phi_x = \alpha_b \alpha_m \alpha_x \mathbf{v}_x^{\mathrm{T}} \mathbf{W}_x \rho(\boldsymbol{\omega}_x)$$
$$\boldsymbol{\omega}_x = \mathbf{G}_x \mathbf{m}$$

where

- $\mathbf{G}_x$ is a gradient operator for the x direction,
- similar definitions exist for the y and z terms.

Note the additional α_m multiplier! When the gradient directions are rotated, the volume vectors $\mathbf{v}_x$, $\mathbf{v}_y$ and $\mathbf{v}_z$ are identical and equal to $\mathbf{v}_c$.

The final smoothness term is a total variation term:

$$\Phi_t = \alpha_b \alpha_m \mathbf{v}_c^{\mathrm{T}} \mathbf{W}_c \left(\boldsymbol{\omega}_t^2 + \epsilon^2\right)^{p/2}$$
$$\boldsymbol{\omega}_t = \mathbf{Q}_x \left((\mathbf{G}_x \mathbf{m})^2\right) + \mathbf{Q}_y \left((\mathbf{G}_y \mathbf{m})^2\right) + \mathbf{Q}_z \left((\mathbf{G}_z \mathbf{m})^2\right)$$

- $\mathbf{W}_c$ is a diagonal matrix containing any user-defined smoothness weights combined with any depth, distance or sensitivity-based weights,

- the ρ function is always the Ekblom measure for this term,
- the exponential powers are applied element-by-element,
- $\mathbf{Q}_x$ interpolates the x-direction squared gradient values at cell centres.

For rectilinear grids, the gradient operators calculate gradients on cell faces and interpolation is required to calculate values at cell centres. For unstructured grids, the gradient operators calculate gradients at cell centres and the interpolation operators are identity matrices.

For the smoothness terms, if `smooth_mod_diff` is `"f"` (false, the default) then the operators are applied to the model, as indicated in the equations above. If `smooth_mod_diff` is set to `"t"` (true) then the operators are applied to $\mathbf{m} - \mathbf{m}_{ref}$ (replace $\mathbf{m}$ with $\mathbf{m} - \mathbf{m}_{ref}$ in the equations above. The appropriate method will depend on the characteristics of the reference model: please see Williams (2008) for guidance.

For the smoothness terms, if `smooth_wts_inside` is `"f"` (false, the default) then the smoothness weights are applied outside the derivatives (to the left of the gradient operators), as indicated in the equations above. If `smooth_wts_inside` is set to `"t"` (true) then the operators are applied inside the derivatives (to the immediate left of the model $\mathbf{m}$, and therefore to the right of the gradient operators, in the equations above). This latter use is consistent with the UBC-GIF inversion codes **but is not recommended**, especially if non-ℓ_2 norms are used.

16.5.3 The joint coupling term

The joint coupling term is a weighted sum of any couplings specified in the input files:

$$\Psi = \alpha_j \sum_k \rho_k \psi_k \tag{16.6}$$

where the sum is over all coupling measures specified, ψ_k.

Scalar quantities used in the equations above correspond to inversion input parameters as indicated below:

symbol	parameter(s)
α_b	`alphab`
α_s	`alphas`
α_m	`alpham`
α_x	`alphax`
α_y	`alphay`
α_z	`alphaz`
α_j	`alphaj`
$\mathbf{W}_s$	`wsfile`, `wsindex`
$\mathbf{W}_f$	`wmfile`, `wmindex`
$\mathbf{W}_x$	`wmfile`, `wxindex`
$\mathbf{W}_y$	`wmfile`, `wyindex`
$\mathbf{W}_z$	`wmfile`, `wzindex`
ϵ	`ekblome`
p	`ekblomp`

Chapter 17

Surface geometry inversion: DYNO (formerly WINV)

17.1 Program summary

DYNO (**DY**namic **NO**de inversion) is a surface geometry inversion program that determines the positions of surfaces in a geological model that represent the contacts between rock units. The discretization is a wireframe surface of tessellated triangles. Single or multiple data-types can be inverted.

The surface(s) can be parameterized with Cartesian coordinate nodes, polar/spherical coordinate nodes, or using circular/spherical harmonics. In Cartesian mode, the (x,y,z) positions of the surface nodes are solved for directly. In spherical mode, the surface is solved for in spherical coordinates (angles fixed, radius determined). In harmonics mode, the real and imaginary values of the spherical (3D) or circular (2D) harmonics are solved for and the Cartesian positions of the surface nodes are calculated indirectly. In spherical and harmonics mode, the centroid(s) remain fixed. The harmonics mode only works for a single closed 2D outline.

A spline can be fit through knot points in 2D Cartesian or polar coordinates. 2D or 3D Cartesian or polar/spherical control points on a coarse wireframe can be refined via surface subdivision.

Gravity (vertical and gradiometry), magnetics and straight-ray seismic data are currently supported. Curved-ray seismics and DC/IP are not supported yet but are planned.

This program will run in parallel. Remember to set the OMP_NUM_THREADS environment variable. On a Mac you use the following command in the terminal window before running this program:

```
> export OMP_NUM_THREADS=value
```

17.2 Running the program

17.2.1 Command line usages

```
dyno inputfile

dyno inputfile outputroot

dyno inputfile outputroot stage

dyno inputfile outputroot stage seed
```

17.2.2 Command line parameters

`inputfile`

- The name of the input file (the format is described below).
- See the note at the start of Chapter 3 regarding relative file paths.

`outputroot`

- The names of the output files have this root (the output files are described below).
- If `outputroot` is absent then it is set to the root of the input file, that is, the input file name with extension and any path information stripped off: for example, if `inputfile` is set to `mydirectory/myfile.ext` then the output files will be named `myfile*` and placed in the current working directory.

`stage`

- Set to 1 to perform the minimization, e.g. using the genetic algorithm solver.

- Set to 2 to perform the burn-in of the Metropolis-Hastings MCMC algorithm (before statistics calculation).
- Set to 3 to perform the statistics calculation (using the Metropolis-Hastings MCMC algorithm).

The general procedure is to first run stage 1 to find a good starting model for the later stages. Stage 2 is an MCMC sampling used as the burn-in. No statistics are calculated in stage 2. The assumption is that after the burn-in stage 2, the Markov Chain has reached equilibrium such that the statistics can then be calculated, which happens in stage 3.

`seed`

- Set to some integer value to use as the seed for random number generation (important for global optimizers).

17.3 Outputs

17.3.1 Terminal output

The output printed to the terminal/command window remains rudimentary and can be altered as users require. Please contact the developers with any suggestions.

Currently, the terminal outputs for a single-objective global search with GA (Genetic Algorithm optimization engine) are:

```
Generation ParamSpread ObjBestEver ObjBestPop VioBestEver VioBestPop NFea NInf
```

where

- `Generation` is the current GA iteration number (generation number)
- `ParamSpread` is the parameter spread
- `ObjBestEver` is the best objective function value (see more below) every obtained by any solution at any iteration (any individual in the GA population at any generation)
- `ObjBestPop` is the best objective function value for the solutions at the current iteration (the individuals in the GA population at the current generation)
- `VioBestEver` is the best constrain violation measure every obtained by any solution at any iteration (any individual in the GA population at any generation)
- `VioBestPop` is the best constrain violation measure for the solutions at the current iteration (the individuals in the GA population at the current generation)
- `NFea` is the number of feasible solutions (which honour the constraints) in the current GA population.
- `NInf` is the number of infeasible solutions (which do not honour the constraints) in the current GA population.

17.3.2 Output files

The output files are named `outroot*` with various suffixes and extensions, as listed below. The output file directory is the directory from which the program is being run, unless specified differently by adding path information to `outroot`.

Output files named `*_best.*` are for the model that fits the data best. When running an MCMC search, additional output files named `*_mean.*` will be written that hold the mean model; the `*_mean.vtu` file will contain node attributes indicating the standard deviation of the node locations. The predicted data files are also written.

17.4 Input file format

Each line of the input file should be of the format

```
name [index] value
```

where `name` is the name of some modelling parameter and `value` is the value for that parameter. Some parameters require that an index is specified to link the parameter to a specific dataset and associated physical property (currently, it is assumed that each dataset is associated with a different physical property). The possible parameters and default values are listed below. Those that require the index specifier are indicated with `"[ ]"`.

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

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

Parameters `revision`, `propinp` and `datainp` are explained further in Section 16.4.

Name	Default	Brief description
`revision`	`0`	a code revision number (see Section 16.4 for more information)
`meshinp`	`"null"`	a model parameterization specification file
`propinp`	`""`	input file for a particular physical property (see Section 16.4 for more information)
`datainp`	`""`	input file for a particular geophysical data set (see Section 16.4 for more information)
`optiminp`	`""`	input file for optimization engine options
`doparinv`	`"t"`	determines where parallelization occurs
`parMPI`	`"f"`	when parallelization occurs, determine if use MPI
`surffile`	`""`	file describing the surface parameterization and any related inversion parameters
`minangle`	`200.0`	minimum facet angle
`colldet`	`"f"`	set to `"t"` (true) to perform collision detection
`collall`	`"t"`	if `"t"` (true, the default) then all facet collisions are counted; otherwise stops counting after the first
`collfile`	`"null"`	a `.ele` file specifying possible groups of facets than could collide (intersect)

`meshinp`

- See the documentation on model discretization specification files for more information on the file format.
- The `meshfile` and `modelfile` should be `.node` and `.ele` format files respectively defining the initial wireframe surface model.
- The `nodeunitsfile` should include the following attributes:
  - `Dynamic` defines whether or not (values 1 or 0) the node can move.
  - `XLower`, `XUpper`, `YLower`, etc. define the upper and lower Cartesian node coordinate bounds. These can be absolute or relative to initial positions (see parameters `relative` and `relativez` - default `"t"` (true)). These bounds are only used in `cart`, `casp` and `casd` mode. They should be specified in the input coordinate system (+z up or down, whichever you are using).
  - `Centroid` specifies centroid indices used to move the nodes in `sphr` and `spsp` mode.
  - `RadiusLower`, `RadiusUpper` and `RadiusScaling` define the bounds and scaling for the radius used in `sphr`, `spsp` and `harm` mode. These can be absolute or relative to initial positions (see parameter `relative` - default `"f"` (false)).
- The `facetunitsfile` should include the attributes `Region1` and `Region2` if you are including gravity or magnetics data. See program `FOGO` for more information.
- The `regionunitsfile` should include the physical property attributes (e.g. `Density`, `MagSus`, `Slowness`, `Conductivity`) required for the data types included.

`optiminp`

- See the documentation on optimization engine specification files for more information on the file format.
- You must use a global optimization engine, e.g. PSO or GA.

`doparinv`

- If `doparinv` is `"t"` (true, the default) then the GA or PMOGA algorithm (if used) calculates the objective function for each candidate solution in parallel. This means that many forward modelling solutions occur in parallel. If `doparinv` is `"f"` (false) then the parallelization will only occur inside the forward modelling solution (if the forward modelling of a given type data is also parallelized), and only one forward modelling solution occurs at any one time.

`parMPI`

- When `doparinv` is `"t"` (true) the GA or PMOGA parallelization is achieved by OpenMP (Open Multi-processing) by default. If `parMPI` is also `"t"` (true) then this task is achieved by MPI (Message Passing Interface) rather than by OpenMP. `parMPI` is `"f"` (false, the default).

`surffile`

- As for the input file, each line of the `surffile` should be of the format `"name value"`.

Name	Default	Brief description
		OPTIONS FOR ALL MODES

Name	Default	Brief description
surftype	""	defines the type of wireframe surface parameterization
ndim	0	the number of dimensions
ztopo	∞	upper bound on z coordinates for subdivided models
coordinatescaling	1.0	scaling applied to the Cartesian node/centroid coordinates
		OPTIONS FOR MODES CART, CASP & CASD
strike	0.0	rotation angle applied to obtain the model for forward modelling
dip	90.0	rotation angle applied to obtain the model for forward modelling
tilt	0.0	rotation angle applied to obtain the model for forward modelling
relative	t	determines whether coordinate bounds are relative to initial node positions
relativez	t	determines whether z-coordinate bounds are relative to initial node z-position
		OPTIONS FOR MODES SPHR, SPSP & HARM
centroidsfile	""	regions format file containing the centroids for each surface
centroidunitsfile	""	rock-units format file that specifies radius bounds and scaling
relative	f	determines whether radius bounds are relative to the initial node positions
		OPTIONS FOR MODES CASP & SPSP
npk	4	the number of nodes per knot
		OPTIONS FOR MODE CASD
nsub	4	the number of subdivisions to perform
stat	t	whether or not the control nodes remain stationary during subdivision smoothing
		OPTIONS FOR MODE HARM
radiusinit	1.0	the initial value of the radius (the initial model is a circle/sphere)
amplitudeboundslower	0.0	lower bound for the 1st and higher harmonic coefficients
amplitudeboundsupper	1.0E6	upper bound for the 1st and higher harmonic coefficients
amplitudescaling	1.0	scaling applied to the amplitudes
maxcoeff	0	the maximum harmonic coefficient to keep

17.4.1 Parameters for all modes

surftype

The possible options are:

- cart to parameterize the 2D or 3D surface(s) in Cartesian coordinates. The parameters are x, y, z for each vertex.
- sphr to parameterize the 2D or 3D surface(s) in spherical coordinates. Nodes move towards or away from their centroid control points. The parameters are r (radius away from centroids) for each vertex. The angles are determined from the initial model and kept constant during the inversion.
- harm to parameterize the 2D or 3D surface(s) in spherical coordinates but stored via harmonic coefficients. The parameters are the centroid(s) and harmonic coefficients of radius versus angle. This mode should generally be avoided, except for research purposes. It is a method taken from medial imaging, where the bodies of interest can be easily fit by a spherical harmonics representation with some small number of harmonics. For geological models, however, a large number of harmonic coefficients is required to accurately describe the model shapes, and this method is then not particularly useful.
- casp to parameterize the 2D surface(s) with knots in Cartesian coordinates and additional interpolated nodes around a spline outline. The parameters are as for the cart mode.
- spsp to parameterize the 2D surface(s) with knots in spherical coordinates and additional interpolated nodes around a spline outline. The parameters are as for the sphr mode.
- casd to parameterize the 2D or 3D surface(s) with the control surface in Cartesian coordinates which gets refined via surface subdivision. The parameters are as for the cart mode.
- spsd to parameterize the 2D or 3D surface(s) with the control surface in spherical coordinates which gets refined via surface subdivision. The parameters are as for the sphr mode.

ztopo

- this parameter is only used when subdividing a model
- it can be used to ensure that no nodes in a subdivided model are above the topography surface
- this parameter does NOT affect non-subdivided models (or the coarse control surface in a subdivision scenario): you should always use appropriate bounds if not subdividing!

- this parameter acts through projection rather than the constraints: any active model nodes above this elevation are projected onto this elevation

`coordinatescaling`

- scaling applied to the Cartesian node/centroid coordinates when taking a random step in the MCMC search
- In MCMC mode the Metropolis–Hastings algorithm is used. In that algorithm, random perturbations to the model are considered. The resulting perturbed candidate models are always accepted if they lower the objective function, and are accepted with non-zero probability if they raise the objective function. If the model perturbations are too large (changes to the model parameters are too large) then it takes too long to converge because the rejection rate is too large: the model perturbations may not be accepted for a long time, during which the chain is stationary and you aren't doing any sensible sampling of the parameter space. If you make too small changes to the model parameters then it also takes too long to converge because it takes a long time to adequately sample all regions of the parameter space. There is a fine balance here. There is some automatic scaling that happens in MCMC but if you can get a handle on the appropriate value of the "coordinatescaling" option then this can help.

17.4.2 Parameters for `cart` mode

Also see the following subsection for parameters shared with other modes.

`writepar`

- Set to `"t"` (true) to write files "par2tot.txt" and "par2dim.txt" containing information on how to convert parameter indices to node and dimension indices.
- Set to `"f"` (false, the default) to not write those files.

17.4.3 Parameters for `cart`, `casp` and `casd` modes

`strike, dip, tilt`

- these rotation angles are applied to the model before calculating the data response
- these are helpful if you are trying to control the movement of the nodes in an object that does not have major axes aligned with the Cartesian axes

`relative, relativez`

- Parameter `relative` is applied to all coordinates.
- Parameter `relativez` is only applied to the z-coordinate.
- Parameter `relativez` overrides parameter `relative`. Hence, if you want all coordinates absolute then both of these parameters must be specified as `"f"` (false) in the input file.

17.4.4 Parameters for `sphr`, `spsp` and `harm` modes

`centroidsfile`

- each node can be moved towards or away from a different centroid
- for `harm` mode, only a single centroid is allowed

`centroidunitsfile`

- this rock-units format file must contain the attributes `RadiusLower`, `RadiusUpper` and `RadiusScaling`
- the scaling is applied to the radius value(s) when taking a random step in the MCMC search

17.4.5 Parameters for `casp` and `spsp` modes

`npk`

- for example, npk=4 results in 3 interpolated nodes between pairs of knots (the nodes in the input model)

17.4.6 Parameters for `sphr`, `spsp` and `spsd` modes

The idea for these modes is that several nodes may share a single centroid that controls their movement, so that centroid information (e.g. radius bounds) needs to be associated with the centroids, not the nodes. Each centroid is defined in the `centroidsfile` (a `.node` format file). Then, for each centroid there may be identical radius bounds, so there is an additional link required from the `centroidsfile` to a `centroidunitsfile`. The `nodesfile` will have an attribute that links each node to a centroid defined in the `centroidsfile`. The `centroidsfile` will have an attribute that links each centroid to a "unit" in the `centroidunitsfile`. The `centroidunitsfile` defines the radius bounds.

17.4.7 Parameters for `casd` mode

`stat`

- 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

17.4.8 Parameters for `harm` mode

`amplitudescaling`

- scaling applied to the amplitudes when taking a random step in the MCMC search

`maxcoeff`

- the 0th harmonic coefficient is treated as the "radius" so is controlled by parameter `radiusinit` and the bounds and scaling on the radius (see parameter `centroidunitsfile`), although the radius scaling is current hardwired to 1.0

17.4.9 Defining constraints

Bound constraints are specified using various parameters mentioned above, e.g. `XLower`, `XUpper`. Additional constraints can be included using either built-in options or by writing your own subroutines.

Built-in constraints

Angle constraints: The second built-in constraint available is to avoid small angles between facets. 2D models with any facets joining at a node and making an angle less than or equal to `minangle`, in degrees, are considered infeasible. This constraint is only applied if `minangle` is on $(0, 180)$, meaning greater than zero and less than 180. This constraint is currently only supported for 2D problems.

Intersection constraints: The third built-in constraint available is associated with collision detection. You can turn on the collision detection using parameter `colldet`. When collision detection is used, `DYNO` counts the pairs of facets in the model that intersect. Collision detection may take considerable time and should be avoided if possible through smart use of bound constraints on the individual surface geometry parameters. However, when collisions are possible, use parameter `collfile` to help limit the number of facet-facet intersection tests: each line of the `collfile` (an `.ele` format file) should specify two indices that are integer facet ID's linking to those in the `.ele` file that specifies the surface-based model (parameter `modelfile` specified in the `meshinp` input file). The `collfile` should specify pairs of facet ID's that COULD possibly intersect and SHOULD be checked during the inversion; any pairs not specified will not be checked. For example, if the `modelfile` contains facets with ID's (first attribute) ranging from 1 to 4 then you might have a `collfile` that looks like this:

```
6 2 0 # <number of lines below=6> <number of indices per line=2> <number of attributes=0>
1 1 1
2 1 2
3 1 3 # <index> <facet ID> <facet ID>
4 2 2
5 2 3
6 3 3
```

which indicates that no intersections are possible with any facet with ID equal to 4 but all other intersections are possible, including between pairs of facets with equal ID values (other than for ID's equal to 4). If the `collfile` is not provided then

collision detection will check every possible pair of facets for intersection, which may take considerable time. To help design the `collfile`, the program `check_dyno_bounds` can be used to determine the facet group pairs that could intersect.

To use these built-in constraints, you must make sure that you set the `ncon` parameter in the optimization engine specification file to a consistent value on $[0, 3]$. If parameter `conobjs` is `"t"` (true) then `ncon` should be incremented by 1; if parameter `minangle` is on $(0, 180)$ then `ncon` should be incremented by 1; if parameter `colldet` is `"t"` (true) then `ncon` should be incremented by 1.

Writing your own constraint subroutines

Please contact me for further instructions.

The objective function

Currently, the objective function minimized in a surface-geometry inversion only includes the data misfit term. If `chifact` equals zero, the `ObjBestPop` value in the terminal output indicates the misfit value. In that case, you probably want to allow the inversion to continue minimizing the misfit term until the maximum number of iterations are reached, or the stall convergence tests (if used) are triggered.

If `chifact` is non-zero, the `ObjBestPop` value in the terminal output indicates the following quantity:

$$\texttt{ObjBestPop} = \frac{|\Phi_d - \texttt{chifact}|}{\texttt{chifact}}$$

where Φ_d is defined in Section 16.5.1 as the standard sum-of-squares misfit divided by the number of data. Hence, `ObjBestPop` is now the relative misfit value (relative to the target value). In this case, you probably want to stop the inversion once the `ObjBestPop` value gets below some tolerance, equivalent to parameter `chitol` for a VIDI inversion, however for `DYNO` you would use the `targ` parameter instead, which is specified in the optimization input file `optiminp`. For example, set `targ` to 0.05 if you want the final misfit within 5% of the desired target.

Writing your own regularization subroutines

Please contact me for further instructions.

Chapter 18

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
2018/01/08	3260	Programs `fwd`, `vinv` and `winv` are now named `fogo`, `vidi` and `dyno`.
2016/01/25	2243	Added `usez` and `icol` command line parameters to program `remove_duplicates`.
"	"	Added program `decimate_by_mesh`.
"	"	Added `atts` command line parameter to program `interpolate_data`.
"	"	Added support for ESRI grid files in program `convert_format`.
"	"	Program `combine_node` now uses `mapflag="f" (false)` when removing duplicates.
2016/03/02	2276	Changes to program `vinv` to move the weights outside the regularization measures by default.
2016/07/18	2399	New inversion option `smooth_wts_inside` for program `vinv`.
"	"	Added `iat` command line parameter to program `remove_trend`.
"	"	Renamed program `interpolate_topography` to `interpolate_data`.
"	"	Added `iat` command line parameter to program `interpolate_data`.
"	"	Added `tol` command line parameter to program `add_noise`.
"	"	Absorbed programs `combine_mesh`, `combine_node` and `combine_poly` into `combine_files`.
"	"	Added support for `.ele` files in program `combine_files`.
"	"	Absorbed program `mesh2solid` into program `convert_format`.
"	"	Added new coupling weight parameters `wjfile` and `wjindex` in `coupling input files`.
"	"	Added `zrev2` command line parameter to program `node2vtu` to control field vectors.
"	"	The smoothness weighting file for rectilinear meshes in program `vinv`
"	"	can now be the UBC-GIF format, with or without smallness weights.
"	"	Added support for Geosoft GXF grid files in program `convert_format`.
"	"	Extended program `smooth_node` to work with 3D surface-based models.
"	"	Extended program `fwd` to calculated sensitivities for first arrival traveltime data.
"	"	Added new program `pad_mesh` for padding a rectilinear mesh.
"	"	Added `ai` command line parameter to program `fwd`.
"	"	Implemented panning for the `FacetModeller` 2D and 3D viewers.
"	"	Added program `blocks2vtu`.
"	"	Added programs `node2d` and `node3d`.
"	"	Program `subdivide` now uses dynamic information from a node units file.
"	"	Added new parameter ailsig to program `data_difference`.
"	"	Added functionality for multi-component mag (new formulation xyz).
"	"	Added more outputs to program `data_difference`.
2016/11/07	2566	Added program `clean_surface`.
"	"	Extended program `convert_format` to convert `.node/.ele` to `.off` and `.obj` 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. This is currently only important when running the inversion programs (see parameter `revision`).

Part V

Workflow for 3D real data inversion

This part of the documentation provides checklists and instructions for performing 3D potential field data inversions with the MUN software. Throughout this part you will see some required reading instructions, which look like this. These should be followed closely. Williams (2008) is cited in several places and you should obtain that reference before you move further.

In this workflow I'll be using ParaView for visualization. I will be providing instructions for creating files for loading into ParaView. If you are using alternative software for visualization then you will need to adjust the instructions accordingly.

◷ Estimated times to complete the tasks look like this. These are conservative upper estimates. They do not include the time needed for the required readings.

Chapter 19

Data checklist

🕑 Estimated time: 60 minutes. This stage may take a lot of back-and-forth emailing between the inversion service provider and the data provider. Encourage them to carefully complete this checklist themselves - time is money. This checklist is available as a separate document.

Use this checklist to make sure you have everything required before continuing further with the inversion procedure. Some omissions from this checklist can be overcome with non-standard procedures but this may introduce significant uncertainty into the inversion procedure.

Legend:

- **Bold text denotes information of critical importance. Inversion can not be completed without that information.**
- Regular text denotes information that is not critical but if it is missing then assumptions must be made that may introduce error into the procedure.
- Grey text denotes information that is not critical but if it is missing then additional processing steps may be required.

19.1 General items for any data

- ☐ **Helpful file names and additional notes that indicate where all the required information resides. Provide a thorough description of each file, including the file format and what information is present, e.g. what each column represents.**
- ☐ **All files in ASCII format.**
- ☐ Ideally, data files are in some column-based format with columns ordered
  Easting - Northing - Elevation - Other quantities.
  Otherwise, conversion will be required.
- ☐ **Units for all quantities.** Ideally, SI units are used; otherwise, conversion will be required.
- ☐ All values provided with as high precision as possible.
- ☐ Ideally, locations provided to at least one decimal place precision.
- ☐ **An explanation of the coordinate system(s) for all information.**
- ☐ UTM or other Cartesian coordinate system. Otherwise, conversion will be required.
- ☐ All data in the same coordinate system. Otherwise, conversion will be required.
- ☐ **Easting and northing for all spatial observations.**
- ☐ Absolute elevations above some datum, e.g. sea level, for all spatial observations. If altitudes above terrain are provided instead then elevations can be calculated using topography information, but this requires interpolation of the topography at the data measurement locations, which may introduce some error.

19.2 Geophysical data

19.2.1 General

- [ ] Estimated uncertainties on all measurements. Otherwise, rule-of-thumb default values can be used but it will be more difficult to determine an appropriate level of data fit.
- [ ] Minimally processed data, e.g. not gridded. Otherwise, spurious inversion artifacts may be generated.
- [ ] Description of all processing applied, e.g. gridding, filtering and corrections. Otherwise, assumptions must be made on what processing steps have already been applied, which may introduce significant error.
- [ ] If vector components are present then they are in the same coordinate system mentioned above. Otherwise, conversion will be required.

19.2.2 Topography data

- [ ] Ideally, data files are in some column-based format with columns ordered
  Easting - Northing - Elevation.
  Otherwise, conversion will be required.
- [ ] Topography data across the lateral extents of interest and ideally extending further (far enough to cover any padding cells added to the mesh). If topography data is not provided then we will assume a flat Earth across the region of interest, which may introduce some error. If topography data does not extend beyond the region of interest then it will be extrapolated, which may introduce minor error if there is significant topography outside the region of interest.

19.2.3 Gravity data

- [ ] Ideally, data files are in some column-based format with columns ordered
  Easting - Northing - Elevation - Gravity Measurements - Uncertainties.
  Otherwise, conversion will be required.
- [ ] Preferred units are mGal for vertical gravity and eotvos for gradiometry. Otherwise, conversion will be required.

19.2.4 Magnetic data

- [ ] Ideally, data files are in some column-based format with columns ordered
  Easting - Northing - Elevation - Total Field Measurements - Uncertainties.
  Otherwise, conversion will be required.
- [ ] Preferred units are nT. Otherwise, conversion will be required.
- [ ] **The geomagnetic field (inclination, declination and strength) at the location and date of the survey, or at least the location and date of the survey so that the geomagnetic field information can be determined.**

19.3 Inversion mesh design

- [ ] **Lateral extents and depth for subsurface region of interest.**

Chapter 20

General Guidelines

Follow the following general guidelines throughout this workflow:

- Keep a record of all processing commands that you use, e.g. in a file named `processing.txt`. Thoroughly comment that file so that all of your work can be reproduced.
- Store all data files in a dedicated directory, e.g. in a directory named `data`.
- Store all model files in a dedicated directory, e.g. in a directory named `models`.
- Store all petrophysical data files in a dedicated directory, e.g. in a directory named `petro`.
- Store all inversion files in a dedicated directory, e.g. in a directory named `inv`, and use a single subdirectory for each inversion run.
- Delete files you no longer need. This reduces clutter and helps avoid using an incorrect file in the future. However, deleting files can be dangerous and you should therefore make sure that you keep a thorough record of your processing steps so you can regenerate the deleted files if required. Alternatively, instead of deleting the files you could move them to some quaranteened directory, but make sure you never use those files for future processing. I prefer the deletion approach.

Read Williams (2008) sections 3.3.1 (pp68–), 3.3.2 (pp70–) and 3.4 (pp77–) before continuing.

Chapter 21

Prepare topography and potential field data for inversion

21.1 Requirements

◷ Estimated time: 15 minutes.

Make sure you have all the following information in place before you continue:

- Topography and potential field data, with all requirements listed in the checklist of Chapter 19.
- An understanding of the uncertainty in the data. See Williams (2008) Section 3.3.2 (pp70–).
- Lateral extents of the local volume of interest. See Williams (2008) Section 3.3.2 (pp70–).
- Lateral extents of the regional volume of interest. See Williams (2008) Section 3.4 (pp77–).
- Resolution of interest. See Williams (2008) Section 3.3.2.

21.2 Data coverage and mesh design

It is important to note that I simplify the design of my inversion meshes compared to the complicated scenario presented by Williams (2008) Figure 3.2 (pp81). I define the following terms:

- The core volume of interest (COI) is what Williams (2008) calls the "local core" volume. This is the main volume of investigation within which you ultimately want to perform finely discretized inversions. You should have finely spaced data available across the lateral extents of this region.
- The regional volume of interest (ROI) contains the COI at its centre and is what Williams (2008) calls the "regional core" volume. You should have data available across the lateral extents of this region but that data does not need to be as finely spaced as over the COI because the regional scale inversions will use coarser meshes.
- The padded regional volume of interest (POI) contains the ROI at its centre and is what Williams (2008) calls the "regional padding" volume. There is no need to have data above the padding zone that lies outside the ROI.

21.3 File format conversion

◷ Estimated time: 60 minutes. This stage may take considerable time, depending on how well the data provider followed the checklist of Chapter 19. The estimated 120 minutes for both the checklist and the file format conversion will likely present a trade-off: the more information we have about the data, the less time the file format conversion will take.

If the data file format is supported by program `convert_format` or `xyz2node` then use one of those programs to convert the data files to 3D `.node` file format. Otherwise, you will have to manually alter the format to one that is supported. Depending on the data file format, you may also need to make use of programs `node2d`, `node3d` and `combine_files`. Programs `remove_nodes` and `remove_range` may also be required to remove null data values. When finished the conversion, use program `node2vtu` to generate `.vtu` files and visualize the data in Paraview.

QC tasks:

- Visually inspect the files in a text editor before and after the conversion to check this was done correctly.
- Visualize the data in ParaView before and after removing null values to check this was done correctly.

21.4 Remove duplicates

◷ Estimated time: 15 minutes.

Duplicate data points should be removed using program `remove_duplicates`. First run

```
> remove_duplicates noderoot null outroot
```

where

- `noderoot` is the root name of the data `.node` file
- `outroot` is a root name you specify for the output `.node` file.

If duplicates exist, this will generate a file named `outroot.node` with duplicate points removed.

The output of the program will indicate how many duplicates were removed. If there were no duplicates then you are finished. Otherwise, now run

```
> remove_duplicates noderoot null outroot 0 t
```

and specify a different `outroot` name than above. This generates another output `.node` file containing only the duplicate points.

QC tasks:

- Check which points were removed and assess why duplicates existed.
- If the duplicates at a particular location have different uncertainties assigned, for example if your data is an amalgamation of different data sets, then you may wish to re-run the duplicate removal using the `icol` parameter of program `remove_duplicates` as follows:

```
> remove_duplicates noderoot null outroot 0 f t icol
```

where you set `icol` to some integer to specify the attribute column that holds the uncertainties in the input data file (`noderoot.node`).

21.5 Convert altitudes to elevations for the potential field data

◷ Estimated time: 15 minutes.

If altitudes were provided instead of elevations then use program `interpolate_data` to interpolate new elevations as follows:

```
> interpolate_data toporoot null origroot origroot_interpolated 1 0 t
```

where

- `toporoot` is the root name of the gridded topography data `.node` file
- `origroot` is the root name of the original potential field data `.node` file.
- The output file is named `origroot_interpolated.node`.

That command sets the `drape` parameter to `"t"` (true) such that the altitudes are added to the interpolated topography values, resulting in proper elevations.

> Program `interpolate_data` can take considerable time. To reduce the run time, you may wish to decimate your potential field data before performing the interpolation: see instructions below in Section 21.8. You can reduce the run time further by also decimating the topography data before performing the interpolation: see instructions below in Section 21.7. However, decimating the topography before interpolation may introduce errors.

21.6 Interpolate gridded potential field data back to the original non-gridded observation locations

◷ Estimated time: 30 minutes.

This is only required if some processing, e.g. filtering, has been performed on gridded potential field data and it is important to use the processed data in the inversion. It is not wise to use the gridded data directly in an inversion because spurious artifacts

can be recovered in regions where the original non-gridded survey data had low data coverage, e.g. between flight lines. The solution is to interpolate the processed gridded data back to the original survey measurement locations, e.g. flight lines.

Use program `interpolate_data` to perform the interpolation as follows:

```
> interpolate_data gridroot null origroot origroot_interpolated 1 1
```

where

- `gridroot` is the root name of the gridded potential field data `.node` file
- `origroot` is the root name of the original potential field data `.node` file.
- The output file is named `origroot_interpolated.node`.

QC tasks:

- Use program `data_difference` to calculate the difference between the original survey data and the interpolated data.
- Visualize the gridded data, the original survey data, the interpolated data and the differences between the latter two in ParaView and check for inconsistencies. Use program `node2vtu` as required.
- You may wish to use the `pow` parameter in program `interpolate_data` to alter the interpolation strategy and try to improve the results, as follows:

```
> interpolate_data gridroot null origroot origroot_interpolated 1 1 f pow
```

The default value for `pow` is the number of dimensions, 3.

> Program `interpolate_data` can take considerable time. To reduce the run time, you may wish to decimate your potential field data before performing the interpolation: see instructions below in Section 21.8. You can reduce the run time further by also decimating the topography data before performing the interpolation: see instructions below in Section 21.7. However, decimating the topography before interpolation may introduce errors.

21.7 Decimate the topography data

◷ Estimated time: 10 minutes.

Extract a patch of the topography data that covers the entire POI (e.g. POI + 100m on all sides) using program `remove_range`. If you do not have topography data across the entire POI then topography will have to be extrapolated at later stages in this workflow. You should at the very least have topography data covering the COI.

QC tasks:

- Visualize the decimated topography data in ParaView and check the decimation was performed correctly.

21.8 Decimate the potential field data

◷ Estimated time: 15 minutes.

First, extract the data within the lateral extents of both the COI and ROI using program `remove_range`.

The data should now be decimated to a spacing consistent with the resolution of interest. You may want to perform different decimations for use in different inversions, e.g. with fine or coarse meshes. There are several possible methods available for data decimation. They are listed below in order of preference.

1. If your data is an amalgamation of different data sets, for example collected at different times or with different instruments, then the uncertainties may be different for each data set. In this case, use program `decimate_by_mesh` to decimate the data such that there is one data point per cell across the surface of the mesh. Use the `icol` parameter in program `decimate_by_mesh` to preferentially keep data points with lower assigned uncertainties. To perform this type of decimation, you will need to generate a rectilinear mesh. You can follow Section 22.3 but you only need to define a mesh with a single vertical layer of cells.
2. If the data are gridded or are collected along flight lines then use program `decimate` to decimate the data along the grid or flight lines. You may first need to use program `break_lines` to split the data into different lines.
3. If the data are in some sensible order but can not easily be broken into different flight lines then use program `decimate` to decimate the data by index.

4. If all approaches above fail, use program `remove_duplicates` and specify a distance. This approach is simple but not suggested because there is no control over which data points are kept.

QC tasks:

- Visualize the decimated potential data in ParaView and check the decimation was performed correctly.
- If the data were decimated based on a mesh, visualize the mesh and check that the mesh and the decimated data are consistent with your expectations.
- If the data were split into different lines, visualize the line information in ParaView and check the splitting was performed properly.

21.9 Upward continuation of the potential field data

Read Williams (2008) Section 3.5.2 (pp88–).

If the data are at a low elevation, e.g. ground data, then you may choose to upward continue the data. This is only necessary if the data contain the response of small, shallow, short wavelength features that are too small to be accurately reproduced by the size of the cells to be used in the inversion mesh. The inversion will not be able to reproduce the short wavelength data features and they may contribute problematic errors to the data misfit. This may bias the result or introduce noise into the recovered model.

Data is generally upward continued such that the new observation height above topography is at least half the width of the mesh cells. For rectilinear meshes, the width of the mesh cells is generally constant across the core region of interest. For unstructured meshes, use a representative width for the majority of the larger cells.

There is currently only a single method available for upward continuation, which is to use the equivalent layer approach of Li et al. (2014). That approach is complicated and ad hoc. It requires many choices for control parameters; those choices are generally made through rule-of-thumb considerations and can significantly effect the final inversion results. ◷ Estimated time: 120 minutes.

Another more approach is to grid the data and upward continue using Fourier series methods. We do not currently have code for that approach but it could be implemented without much difficulty if required.

Often, upward continuation is not necessary, particularly if the data have been decimated appropriately. Appropriate decimation is the suggested approach and, hence, I do not provide further instructions on upward continuation here. ◷ Estimated time: 0 minutes.

21.10 Preliminary levelling of potential field data

◷ Estimated time: 5 minutes.

Preliminary levelling of gravity data

For gravity data, you want to remove the average value. Use program `remove_trend` as follows:

```
> remove_trend noderoot outroot 1 0
```

where

- `noderoot` is the root name of the gravity data `.node` file
- `outroot` is a root name you specify for the output `.node` file.

Preliminary levelling of magnetics data

For magnetics data, you want to remove the geomagnetic field strength value. Use program `remove_trend` as follows:

```
> remove_trend noderoot outroot 1 0 cval
```

where

- `noderoot` is the root name of the magnetics data `.node` file

- `outroot` is a root name you specify for the output `.node` file
- `cval` is the **negative value of the** geomagnetic field strength: the `cval` value supplied is added to the data, so you must provide a negative value here.

21.11 Assign uncertainties to the potential field data

◷ Estimated time: 5 minutes.

If uncertainties have not been assigned to the potential field data already then you can do so using program `add_noise`. If you are not sure what level of uncertainty to add, a floor of 2% of the data range is what I would consider a sensible minimum:

```
> add_noise nodefile sig 1 2 0.0 2.0 0.0
```

For now, you should stick with using only a floor value. Using some percentage error based on the data values is not sensible until the data levelling has been fine-tuned below.

21.12 Interpolate data elevations

◷ Estimated time: 20 minutes (10 minutes to check if this step is required + 10 minutes to perform the step if required).

21.12.1 Check data observations are above topography

Sections Section 21.13 and Section 21.14 in this chapter discuss data processing steps that rely on inversion. Once you have an inversion mesh ready, e.g. following the instructions in Chapter 22, you should make sure that any data collected above the surface of the earth are in fact above the surface of your inversion mesh. There may be errors associated with the collection of the data elevation information, or the topography information, that might lead to having data lying below the mesh topography surface. However, the culprit is usually the fact that the mesh topography surface is a coarse representation of the actual topography of the Earth. This issue is generally only a problem for ground-based survey data, or for low altitude airborne survey data flown over significant topography.

To check for an issue, you can visually inspect the data in ParaView, but a more thorough approach is as follows. First, use program `interpolate_data` to calculate topography values below the data locations, using the topography surface used to generate the top of the mesh:

```
> interpolate_data meshtopo meshtopo dataroot dataroot_interpolated 1 0 f
```

where:

- Files `meshtopo.node/.ele` are the files that define the topography surface of your inversion mesh. These correspond to files `outlines.1.node/.ele` in the example of Section 22.4.3. If such files do not exist in your processing stream, you may be able to trim off the topography surface of your mesh using program `remove_range`, `remove_cells` or `remove_nodes`.
- File `dataroot.node` is your potential field data file.
- The output file is named `dataroot_interpolated.node`.

Now calculate the difference between those topography values and the data elevations using program `data_difference`:

```
> data_difference dataroot_interpolated.node dataroot.node diff.node 0 0
```

where:

- File `dataroot_interpolated.node` is the output file from running program `interpolate_data` above.
- File `dataroot.node` is your potential field data file.
- File `diff.node` is the output file of program `data_difference`, which you can give any name. You do not need this file and it can safely be deleted.

Look in the printed output of program `data_difference` and make sure all the differences are positive, meaning that the data elevations minus the interpolated topography values are all positive, which for a $+z$-up coordinate system means all the data observation locations are above the topography surface of the inversion mesh.

If that check fails, you should now fix the problem by interpolating new elevations for your data. The procedure is described in the following subsection.

21.12.2 Interpolate new data elevations above the inversion mesh topography surface

The procedure is to first calculate the data altitudes above the Earth's "true" topography surface and then drape those altitudes above the approximated topography surface of the inversion mesh. If you already have the altitudes for your data then you can skip the first two steps below.

First calculate the "true" topography values below the data locations, as we did above in Section 21.12.1 using program `interpolate_data`, but here we want to use the original, e.g. finely gridded, topography information. We will take those to be topography values for the true Earth. For example, you might run

```
> interpolate_data toporoot null dataroot dataroot_interpolated 1 0 f
```

where:

- File `toporoot.node` is your gridded topography data file
- File `dataroot.node` is your potential field data file.
- The output file is named `dataroot_interpolated.node`.

Next, calculate the data altitudes (elevation minus topography) using program `data_difference`, for example:

```
> data_difference dataroot_interpolated.node dataroot.node dataroot_altitudes.node 0 0
```

where:

- File `dataroot_interpolated.node` is the output of program `interpolate_data` above.
- File `dataroot.node` is your potential field data file.
- The output file is named `dataroot_altitudes.node`.

The output file contains the altitudes in the z column usually reserved for elevations. Use program `print_coordinates` to check that all the altitudes are positive:

```
> print_coordinates dataroot_altitudes
```

Now interpolate new elevations using program `interpolate_data` by draping the altitudes over the topography surface of the inversion mesh:

```
> interpolate_data meshtopo meshtopo dataroot_altitudes dataroot_altitudes_interpolated 1 0 t
```

where:

- Files `meshtopo.node/.ele` are the files that define the topography surface of your inversion mesh.
- File `dataroot_altitudes.node` is the output file of program `data_difference` above.
- The output file is named `dataroot_altitudes_interpolated.node`, which holds the new elevations.

Finally, you have to combine different pieces of information in different files. For this running example:

- The elevations you want are in file `dataroot_altitudes_interpolated.node`.
- The data measurements you want are in your original file `dataroot.node`.

To combine them correctly you can use program `combine_files`:

```
> combine_files H 2 node dataroot_altitudes_interpolated dataroot dataroot_new_elevations
```

where the output file is named `dataroot_new_elevations.node` but you can name it whatever you like.

21.13 Further levelling of potential field data

◷ Estimated time: 30 minutes for inversion setup + inversion run time.
Total time may be a few days, depending on how thoroughly you perform the levelling. However, much of that time is spent waiting for inversions to run so there are opportunities to do other work at the same time.

Read the following in Williams (2008):

- **Section 3.5.3 subsection "Data levels" (p97)**
- **Section 3.5.4 subsection "Data levels" (pp100-101)**

- **Section 3.5.5 (pp101–)**
- **Section 3.9.2 (pp129–)**

The procedure is to invert on a mesh with the data at different levels: different constant values added or subtracted from the data response. Chapter 22 provides instructions on how to generate unconstrained meshes. Constrained meshes will likely require the use of FacetModeller.

The inversion mesh can be unconstrained or constrained, e.g. with a priori surfaces incorporated into the mesh. However, it is generally preferable to incorporate as much constraining information when performing the levelling in this chapter. Without any petrophysical constraints, gravity data cannot be levelled beyond simply removing the mean value, as already performed Section 21.10. Without any petrophysical constraints, magnetics data can be levelled further but only if positivity is enforced, which is essential for the levelling in this chapter (this was not mentioned in Williams (2008)).

The inversion mesh can cover the core COI or padded POI. However, it is always preferable to include some padding cells, even if they are only a few added to the COI mesh. This is because the levelling procedure in this chapter can only work if the inversion is provided enough freedom to fit the still existing constant shift in the data. Adding padding cells helps the inversion in this regard and can make the spurious artifacts easier to interpret. For this reason, I suggest you always use the POI mesh.

Generally, the mesh used for the inversion-based regional removal procedure of Chapter 23 is a sensible option for the inversion-based levelling procedure. In fact, the two procedures are related: generally the regional removal procedure is able to remove an appropriate constant shift but, again, only if there are enough padding cells to give the inversion enough freedom. Hence, I do suggest you take the time to go through both procedures thoroughly. Another levelling approach is simply to perform the regional removal procedure, then use program `remove_trend` to calculate the average value of the regional component, then remove that from the original data, then iterate until convergence (until the average value of the regional component is zero).

You may need to perform many inversions here to determine an appropriate level of data fit. When in doubt, use a slightly tighter data fit than you expect you might want to use in any inversions performed after this levelling procedure.

Before each inversion you will use program `remove_trend` to remove a constant shift as follows:

```
> remove_trend inroot outroot 1 0 cval
```

where:

- `cval` should specify the constant shift to add to the data.
- `inroot` should specify the root name of your data node file.
- `outroot` should specify your desired root name for the output file. You should use some sensible prefix on this root name to indicate the constant shift removed, e.g. "_p500" if you have added -500nT to the magnetics data, or "_n500" if you have subtracted 500nT.

QC tasks before inversion:

- Visualize any input mesh and data files in ParaView and check for consistency with your expectations.

QC tasks after inversion:

- Check logged inversion convergence information. You should see a smooth convergence in the misfit from high value down to specified target value. If anything is wrong, alter input parameters and re-run.
- Visualize recovered models in ParaView and check for spurious artifacts.
- Visualize predicted data files in ParaView, compare against observed data, and check for regions of coherent high misfit in the normalized data residuals.

21.14 Regional removal for the potential field data

There are two possible methods available for regional removal. You may wish to combine them in the order indicated or only perform the scooping procedure.

1. Remove a constant, linear, or higher-order polynomial trend using program `remove_trend`. ◷ Estimated time: 10 minutes.
2. Use the scooping procedure of Li & Oldenburg (1998b). This procedure is described in detail in Chapter 23. A specially designed mesh is required; the following Chapter 22 provides instructions on how to generate the required mesh. ◷ Estimated time: refer to the following chapters.

Chapter 22

Generating unconstrained inversion meshes

This chapter provides instructions that can be followed to generate meshes for use in:

- equivalent layer inversion, e.g. for upward continuation
- inversion-based levelling of potential field data using the iterative procedure of Williams (2008)
- regional removal using the scooping procedure of Li & Oldenburg (1998b)
- unconstrained inversion.

Section 22.3 deals with rectilinear meshes and Section 22.4 with unstructured meshes. Each of those sections provides instructions for generating meshes a on core volume of interest (COI) and a padded volume of interest (POI).

Read Williams (2008) section 3.4 (pp77–), which provides detailed considerations for mesh design. He works with rectilinear meshes but many of the same design considerations translate to unstructured meshes.

22.1 Requirements

◷ Estimated time: 10 minutes.

Make sure you have all the following information in place before you continue:

- Topography data, processed following the instructions in the previous Chapter 21.
- Lateral and depth extents of the COI. See Williams (2008) section 3.3.2 (pp70–).
- Lateral and depth extents of the POI. See Williams (2008) section 3.4 (pp77–).
- Resolution of interest. See Williams (2008) section 3.3.2.

22.2 Determine mesh depth extents

◷ Estimated time: 5 minutes.

The lateral extents of the COI are usually determined from exploration goals. The depth extent for the COI mesh should be no greater than half the lateral COI range. For data collected above the Earth's surface, there is not enough information to reliably recover information any deeper than half the lateral data range. If the easting and northing ranges differ, use the largest to determine an appropriate mesh depth. I'll denote this elevation z_{range}. If a COI depth has been determined from exploration goals, which I'll denote z_{goals}, then that depth should be shallower than the largest sensible depth estimated via the lateral data range, i.e. $z_{goals} \leq z_{range}$. Otherwise, the data can not possibly provide enough information about the subsurface to answer the exploration goals down to the desired depth. In that case, the exploration goals should be revisited or more data obtained. If $z_{goals} \leq z_{range}$ then it is safest to use z_{range} as the depth of the COI mesh so that the mesh can model possible deeper sources whose signals may have been captured in the survey data.

You also need to determine the elevation of the lowest and highest topography measurement. This can be achieved using program `print_coordinates`. I'll denote those elevations z_{topo1} and z_{topo2} respectively.

22.3 Generating 3D rectilinear meshes

◷ Estimated time: 30 minutes.

The procedure for generating rectilinear meshes is less involved than for unstructured meshes. However, rectilinear meshes should only be used if there is not significant topography. If in doubt, use program `print_coordinates` to determine the

topography range across the POI. If that range is significantly smaller than the desired resolution then you will likely have mesh cells with significantly larger vertical dimensions than the topography range, in which case you may be able to safely ignore the topography.

22.3.1 Generating a 3D rectilinear mesh for a COI

Use the UBC-GIF mesh file format, described in Section 3.2.1, to generate a mesh that covers the lateral extents of the COI and runs from an elevation of z_{topo1} down to z_{range}. The simplest approach is to begin with a regular rectilinear mesh (all cell dimensions identical). That regular mesh can easily be changed later if desired, e.g. to an irregular rectilinear mesh.

> I suggest using an elevation of z_{topo1} because this will ensure that any data collected above the Earth's surface lie above the surface of the mesh. This is particularly important for ground-based surveys. This contrasts with the UBC-GIF approach, which is to build the mesh up to an elevation of z_{topo2} and then strip away cells that are above the topographic surface. Unfortunately, because of errors representing the topographic surface on a pixellated rectilinear mesh, this does not guarantee that ground-based survey data will actually lie above the resulting mesh surface. This is one reason why data is often upward continued in the UBC-GIF processing workflow. I prefer using unstructured meshes when there is significant topography because it ameliorates these concerns.

QC tasks:

- Visualize the COI mesh in ParaView, making use of program `ubcgif2vtr` as required.
- Perform a visual comparison of the positioning of the COI mesh with the topography and potential field data. Check the positioning is consistent with your expectations.

22.3.2 Generating a 3D rectilinear mesh for a POI

First you should generate the mesh for the COI following the instructions above. Then you can add padding cells to the core volume. This can be done manually or using program `pad_mesh`.

QC tasks:

- Visualize the POI mesh in ParaView, making use of program `ubcgif2vtr` as required.
- Perform a visual comparison of the positioning of the POI mesh and the COI mesh. Check the positioning is consistent with your expectations.
- Perform a visual comparison of the positioning of the POI mesh with the topography and potential field data. Check the positioning is consistent with your expectations.

22.4 Generating 3D unstructured meshes

◷ Estimated time: 90 minutes.

- **You will need the meshing programs Triangle (`http://www.cs.cmu.edu/~quake/triangle.html`) and TetGen (`http://wias-berlin.de/software/tetgen/`) installed and compiled on your computer.**
- **Read about the concepts of Planar Straight Line Graphs (PSLGs, `http://www.cs.cmu.edu/~quake/triangle.defs.html`) and Piecewise Linear Complexes (PLCs, `http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual002.html#sec7`) before you continue.**
- **Read about 2D (`http://www.cs.cmu.edu/~quake/triangle.poly.html`) and 3D (`http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual006.html#ff_poly`) `.poly` file formats before you continue.**
- **Read about the different options for running Triangle (`http://www.cs.cmu.edu/~quake/triangle.switch.html`) and TetGen (`http://wias-berlin.de/software/tetgen/switches.html`) before you continue.**

The procedure for unstructured meshes is more involved than for rectilinear meshes but there are utility programs to make this fairly simple. You begin with a topography data file in the `.node` format. You need to generate a triangulated topography surface to become the top of your mesh. Generally, topography data is very densely sampled and generating a triangulated topography surface with the raw topography data would lead to a mesh with far more cells than necessary. Hence, the suggested procedure is as follows:

1. Triangulate the topography data, for interpolation purposes, using the Triangle meshing program (http://www.cs.cmu.edu/~quake/triangle.html).
2. Generate a surface of tessellated triangles to become the top of the mesh, again using the Triangle meshing program.
3. Interpolate the elevations for that tessellated surface.
4. Sew that tessellated surface onto the top of a block representing the volume of interest (VOI).
5. Use the TetGen meshing program (http://wias-berlin.de/software/tetgen/) to generate a 3D unstructured mesh of space-filling tetrahedra.

Those five steps are discussed in more detail in the subsections below.

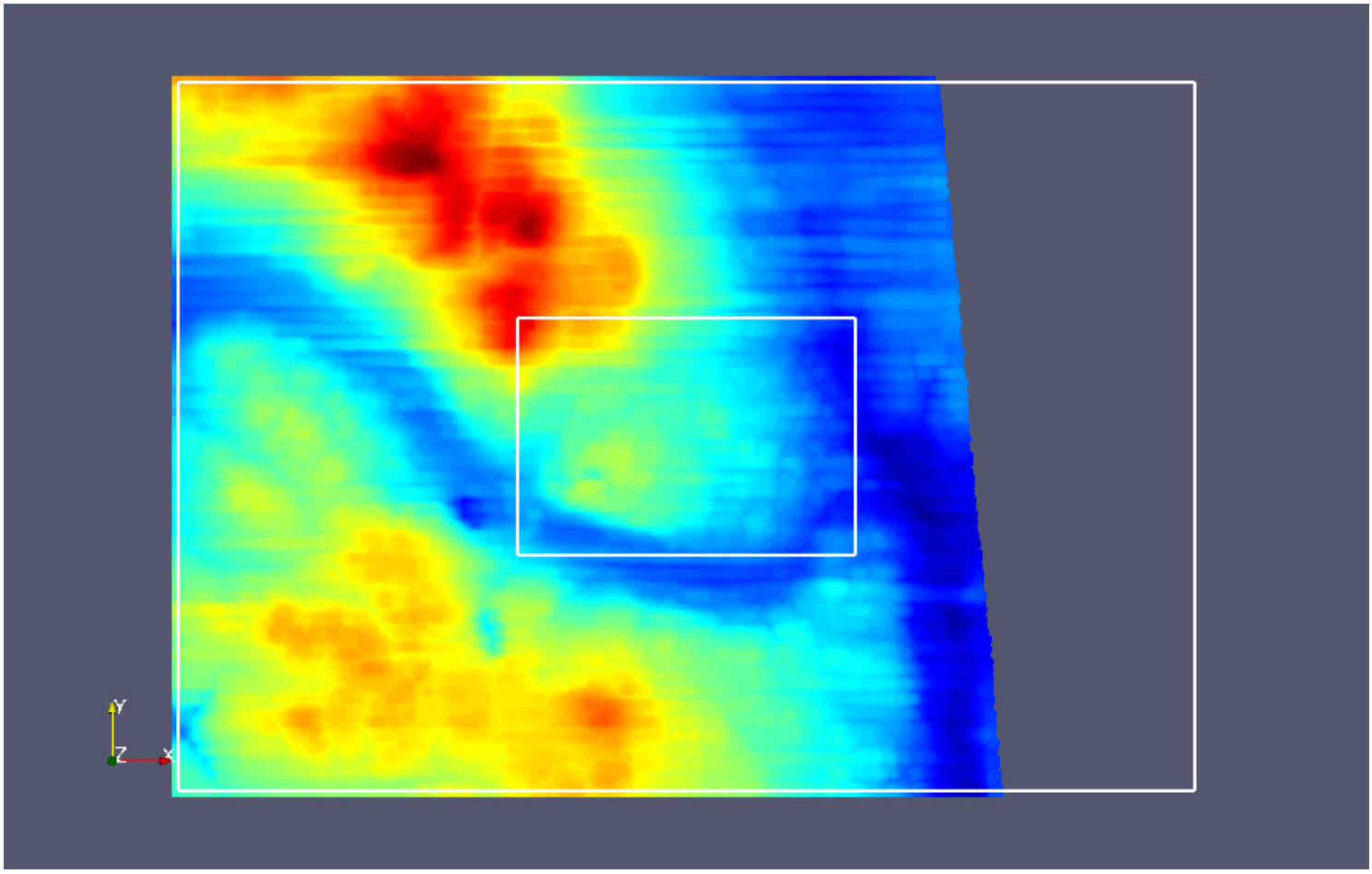
22.4.1 Triangulate the topography data for interpolation purposes

While this step is optional, I suggest you use it whenever your topography data covers the entire VOI. The interpolation in section Section 22.4.3, using program `interpolate_date`, can be performed in two ways:

1. Using a triangulated version of the topography. The interpolation is linear across each triangular surface patch.
2. Using a non-triangulated version of the topography (points only). The interpolation is performed using inverse distance weighting (IDW).

The triangulated approach is much faster and no less accurate than the IDW approach. However, if your topography data does not cover the entire POI then it is easier to skip this section and use the IDW approach instead.

The screenshot below shows some gridded topography data after performing the processing in Section 21.7. The white lines indicate the lateral extents of the POI and COI.



To triangulate the topography data for interpolation purposes, you first have to change the 3D topography `.node` file to a 2D file for passing to the Triangle meshing program. This can be achieved in a text editor by changing the header of the `.node` file. Alternatively, you can use program `node2d`.

Now run the Triangle meshing program to triangulate the topography. You want to make sure not to add any additional nodes. Generally, the following command achieves what is required when the topography data are on a grid:

```
> triangle topo.node
```

where `topo.node` should be replaced by the name of your topography `.node` file. That generates two new files named `topo.1.node` and `topo.1.ele`. The output file `topo.1.ele` contains the required information: it lists the triangles in the triangulated topography surface.

Don't forget to change the header in your original topography `.node` file back to 3D. Again, this can be achieved in a text editor or you can use program `node3d`.

QC tasks:

- Check that the input `.node` file passed to Triangle (e.g. `topo.node`) contains identical coordinate information to the output `.node` file generated by Triangle (e.g. `topo.1.node`). It is generally enough to compare the number of nodes in the two files. If the the coordinate information in the two `.node` files is not identical then you will have to give up on the topography tessellation in this section and continue with IDW interpolation in Section 22.4.3.
- Visualize the triangulated topography in ParaView, making use of program `mesh2vtu` as required, and check the meshing performed as expected.

You should now delete the redundant output file, e.g. `topo.1.node`.

> Triangle works in 2D. By changing the header of your topography `.node` file before running Triangle you are essentially telling Triangle to ignore the elevations: you are tricking Triangle by making it think that the elevations are in an attribute column rather than a coordinate column. If you run Triangle and the output `.node` file has different nodes than the input `.node` file then Triangle has added new nodes. The problem is then how to assign elevations to those new nodes. You could use program `interpolate_data` to interpolate topography elevations at those new nodes using IDW interpolation but that is circular reasoning: you may as well give up on the topography tessellation in this section and continue with IDW interpolation in Section 22.4.3.

22.4.2 Generate a surface of tessellated triangles to become the top of the mesh

You will again use Triangle for this task. You need a 2D `.poly` file to specify the lateral extent of the VOI. This file can simply specify a single rectangle for the COI, or it could specify two rectangles, one for the COI and another for the POI, within which you want different cell sizes. Here I show you that more complicated case as it is usually the desired mesh scenario. You can generate the `.poly` file by hand in a text editor, or using FacetModeller, or you can make use of program `blocks2poly`.

Using a text editor is usually simple enough. A 2D `.poly` file could look like this:

```
8 2 0 0
1 xc1 yc1
2 xc2 yc1
3 xc1 yc2
4 xc2 yc2
5 xp1 yp1
6 xp2 yp1
7 xp1 yp2
8 xp2 yp2

8 0
1 1 2
2 2 4
3 4 3
4 3 1
5 5 6
6 6 8
7 8 7
8 7 5

0

2
```

```
1 xcr ycr 0.0 ac
2 xpr ypr 0.0 ap
```

where

- `xc1`, `xc2`, `yc1` and `yc2` define the easting and northing limits of the COI.
- `xp1`, `xp2`, `yp1` and `yp2` define the easting and northing limits of the POI.
- `xcr` and `ycr` define a coordinate point lying anywhere inside the COI.
- `xpr` and `ypr` define a coordinate point lying inside the POI but outside the COI.
- `ac` and `ap` define the desired cell areas inside and outside the COI: `ac` should be set based on the desired resolution; `ap` should generally be set to a larger value to reduce the number of cells outside the COI.

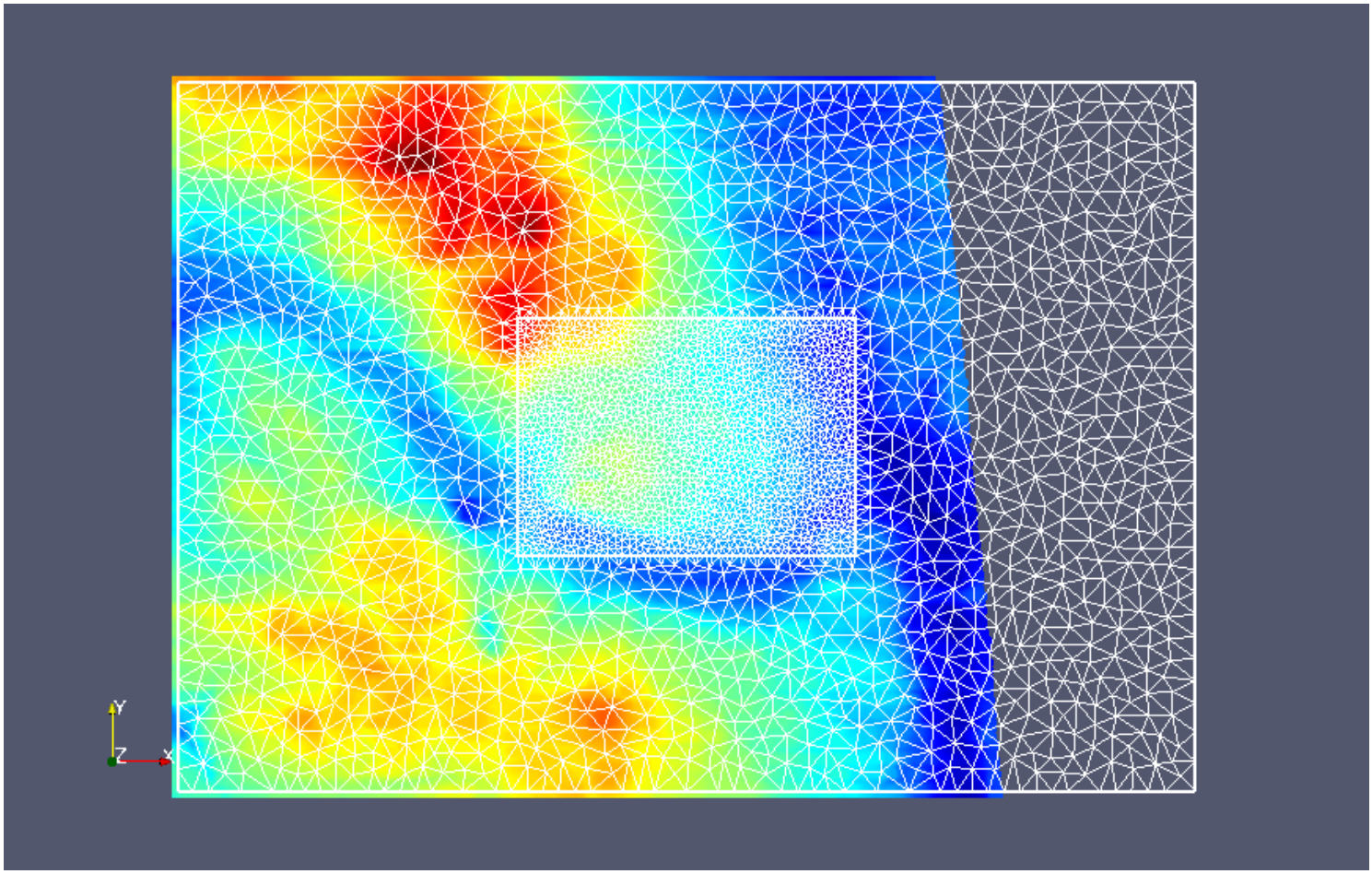
QC tasks:

- Visualize the information in the `.poly` file in ParaView, making use of program `poly2vtu` as required.
- To convert to a 3D object for viewing in ParaView, use programs `poly2mesh`, `node3d`, `mesh2poly` and `poly2vtu` in that order, making sure not to overwrite the original `.poly` file.
- You could also try the shorter procedure using `poly2mesh`, `node3d` and `mesh2vtu`, although this may lead to a slightly different result in ParaView (the information shown will be the same, but the style of visualization may change, depending on the ParaView version and computing platform).
- Program `node3d` is used because you want to visualize the `.poly` file as a 3D object, because it represents the top of your mesh. However, the `.poly` file is defined as a 2D object because the intention is to pass it to the Triangle meshing program. If you were to convert the 2D poly file to a `.vtu` file directly, then load it into ParaView, unfortunately ParaView treats the 2D coordinates as (x, z), and sets $y = 0$, when really the coordinates in the `.poly` file are (x, y). Hence, all of this is trickery is just to get ParaView to display it correctly in 3D.
- Check for consistency between that `.poly` file object and the locations of the topography and potential field data.

Your `.poly` file will likely specify different maximum triangle area constraints, `ac` and `ap`, inside and outside the core rectangle (smaller cells inside the core, larger in the padding region). Running the Triangle meshing program with the `-a` switch specifies that you want to prescribe those constraints. You should also use the `-q` and `-D` switches to improve the quality of the mesh: this will make it easier for TetGen to create the final 3D mesh. The following command generally achieves what is required:

```
> triangle -pqaDP outlines.poly
```

where `outlines.poly` should be replaced by the name of your `.poly` file. This generates two new files named `outlines.1.node` and `outlines.1.ele` which will be used later. Returning to the running example, the screenshot below now additionally shows the tessellated surface that will become the top of the 3D mesh; the edges of the triangles are painted with thin white lines.



QC tasks:

- Visualize the information in the output files from Triangle. To convert to a 3D object for viewing in ParaView, use programs `node3d` and `mesh2vtu` in that order.
- Check for consistency between that mesh object (the output of Triangle) and the `.poly` file object (the input to Triangle).

22.4.3 Interpolate the elevations for that tessellated surface

You have to make sure you are working in 3D, so change the header of the `.node` file generated by Triangle in the previous step (e.g. `outlines.1.node`) using a text editor or program `node3d`. The elevations in that file are currently meaningless: Triangle created a column of boundary markers (values of 0 or 1) that are now treated as elevations. You now want to use program `interpolate_data` to replace those meaningless elevations with values interpolated from the topography information. If in Section 22.4.1 you triangulated your topography surface for interpolation purposes, then the command

```
> interpolate_data topo topo.1 outlines.1 outlines_interpolated 1 0 f
```

takes the topography information in files `topo.node` and `topo.1.ele` and interpolates at the locations in file `outlines.1.node`. If you did not triangulate your topography surface in Section 22.4.1 then you will have to use IDW interpolation, for example using the command

```
> interpolate_data topo null outlines.1 outlines_interpolated 1 0 f [pow]
```

where you may want to play with the `pow` parameter to improve the interpolation (the default value is the number of dimensions, 3).

The output file from the call to `interpolate_data` above is named `outlines_interpolated.node` and there may possibly be a second named `outlines_interpolated_2d.node`. I like to rename the former file to `outlines.1.node`, which overwrites the previous file of the same name generated in Section 22.4.2. The latter output file, `outlines_interpolated_2d.node`, is not required and can be safely deleted. Alternatively, to save a step, you could use the command

```
> interpolate_data topo null outlines.1 outlines.1 1 0 f [pow]
```

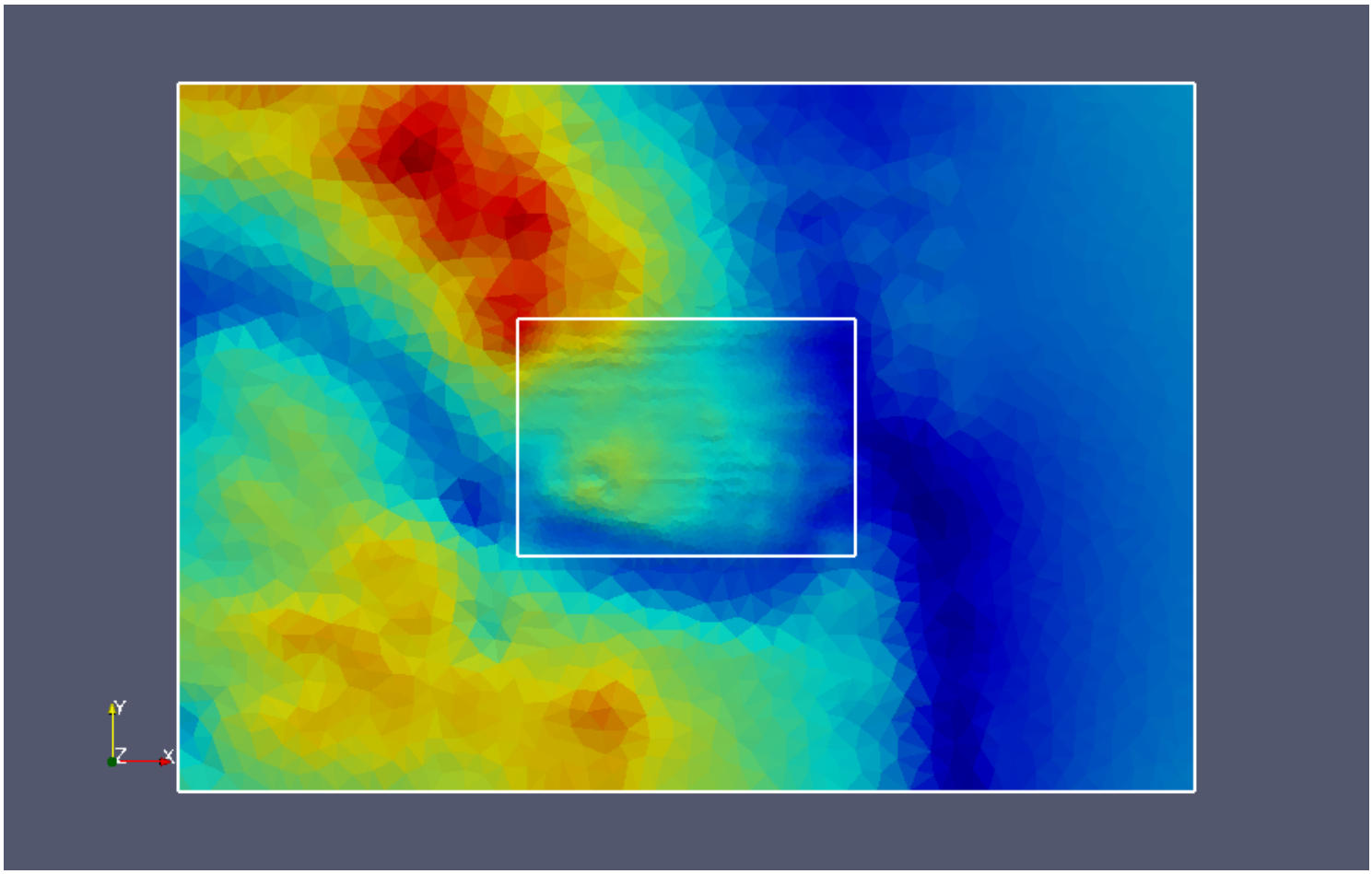
which immediately overwrites the file `outlines.1.node`, but if a mistake is made then you will then need to run the meshing again that generated file `outlines.1.node`.

If the interpolation takes too long, you may want to use program `decimate` to decimate the topography and return to Section 22.4.1.

QC tasks:

- Visualize the interpolated topography surface in ParaView, making use of program `mesh2vtu` as required.
- Check for consistency between the interpolated topography surface and the original tomography data.

The following screenshot shows the new tessellated mesh surface, coloured by the interpolated elevations, which will become the top of our inversion mesh.



22.4.4 Sew the tessellated mesh surface onto the top of a block representing the VOI

This is a multiple-stage process. If there were only a single block representing the VOI then it would be simpler but here we have two blocks: one for the COI and another for the POI. The basic idea is to generate two objects and then combine them together. You'll use program `conform_topography` twice and therefore need two blocks-format files (see Section 3.9).

First, create two blocks-format files for both the COI and POI. They will both look like this

```
1 1 3 0 0
xc yc zt dx dy dz 0.0 90.0 0.0
```

where

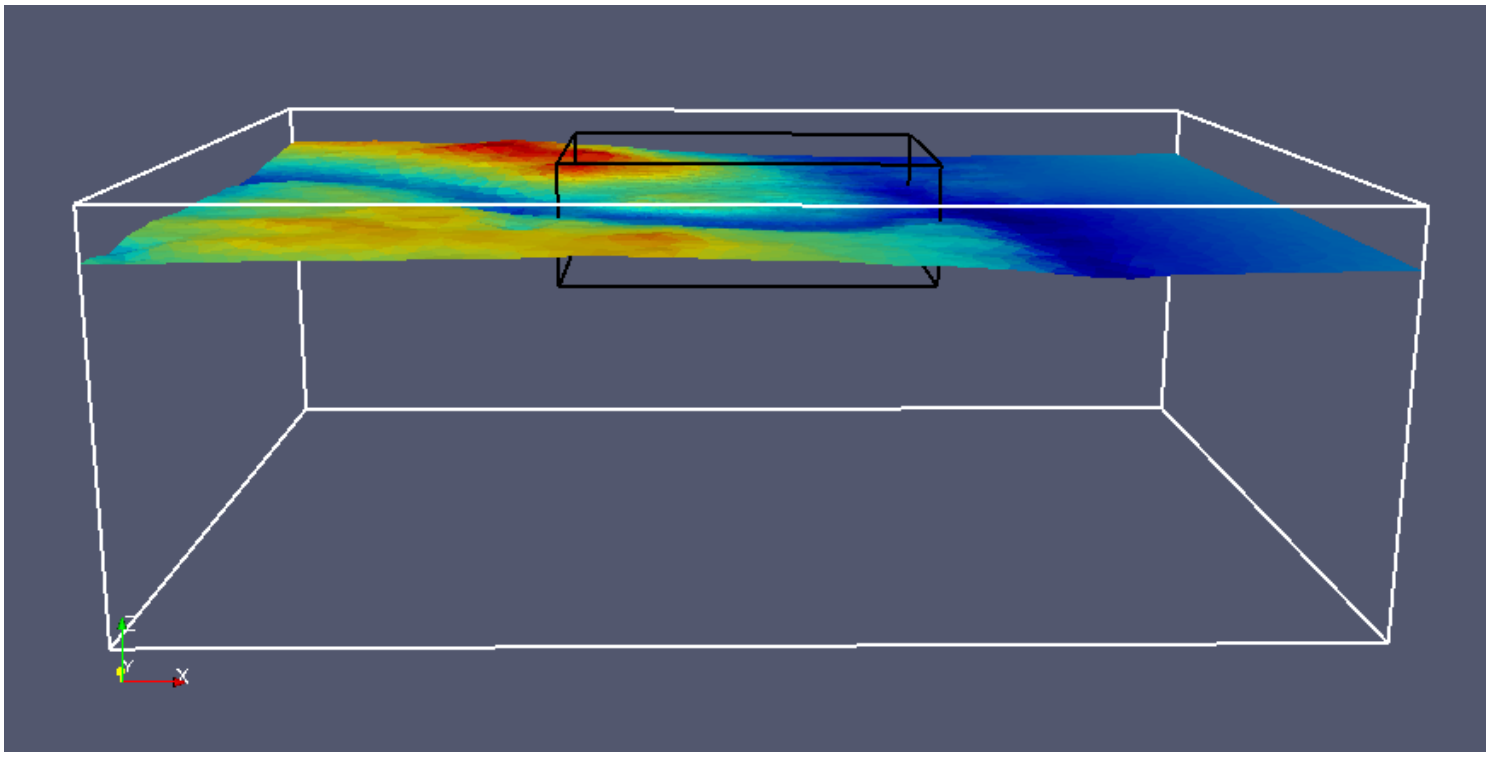
- `xc` and `yc` define the lateral centre of the volume (COI or POI depending on the file).
- `zt` defines the elevation of the top of the volume. This must be above the largest elevation in the topography surface, z_{topo2}. The value of `zt` used for the COI and POI should be identical.
- `dx`, `dy` and `dz` define the size of the volume. **The value of `dz` used for the COI should be less than that used for the POI, so that the COI is completely contained within the POI. TetGen will fail if any facets in the input PLC are coplanar, for example if the base of your COI is at the same depth as the base of your POI.**

In such a scenario, you would have to use FacetModeller to define facets on the bottom of your mesh to follow the rules of a valid PLC (refer to http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual002.html#sec7).

QC tasks:

- Visualize the blocks-format files in ParaView, making use of program `blocks2vtu` as required.
- Perform a visual comparison of the positioning of the blocks with the topography and potential field data. Check the positioning is consistent with your expectations.

The screenshot below shows a view of the tessellated mesh surface (coloured by elevation) and the two blocks (black and white).



You now need to run program `conform_topography` twice to generate two objects:

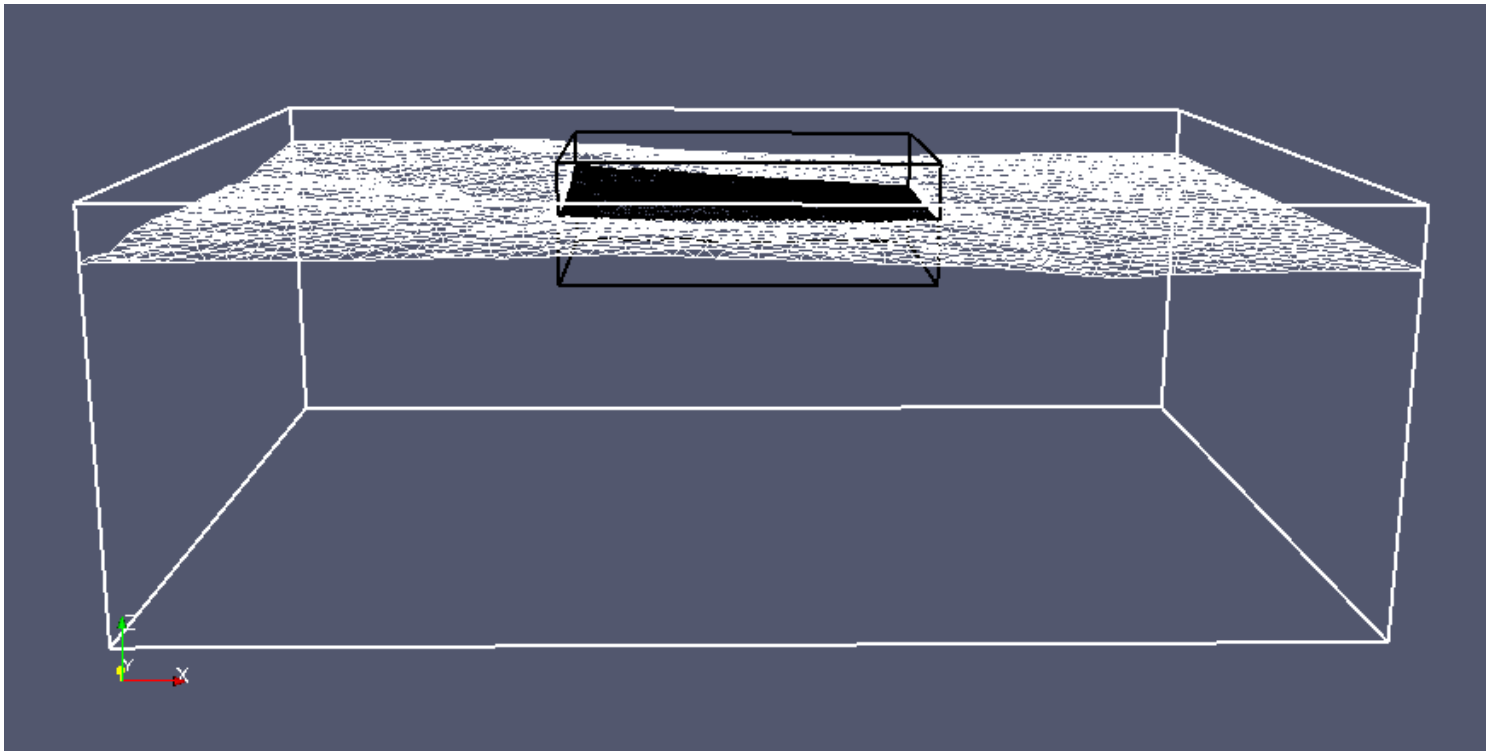
```
> conform_topography blocks_COI.txt outlines.1 conf_COI f
> conform_topography blocks_POI.txt outlines.1 conf_POI f
```

where

- File `blocks_COI.txt` should be replaced with the name of your blocks-format file for your COI.
- File `blocks_POI.txt` should be replaced with the name of your blocks-format file for your POI.
- File root `outlines.1` should be replaced with the root name (no extension) of the file generated after running program `interpolate_data` in Section 22.4.3.

This generates 8 files named `conf_COI/POI_topo/boundary.node/.ele`. There are 4 objects stored in those 8 files because the PLC's for the COI and POI are each divided into the topography facets and the facets around the boundary sides.

The following screenshot shows the two objects for the running example: the black wireframe contains all the facets for the COI; the white wireframe contains all the facets for the POI. Those two objects contain identical topography facets above the core volume (those duplicate facets are removed in the processing that follows).



To combine the objects together you use program combine_files, for example:

```
> combine_files V 4 mesh conf_COI_topo conf_COI_boundary conf_POI_topo conf_POI_boundary combined
```

The result for that command are files named `combined.node` and `combined.ele`. There are some additional facets in that combined mesh above the topography surface that can be removed using program remove_range:

```
> remove_range combined combined combined_range t -1E9 1E9 -1E9 1E9 -1E9 z2
```

(where the output file is named `combined_range.node`) or

```
> remove_range combined combined combined t -1E9 1E9 -1E9 1E9 -1E9 z2
```

(where the input file `combined.node` is overwritten). There you need to supply huge coordinate values for all the input parameters except the last, `z2`, which should be set to an elevation lying just below the top of the original COI and POI blocks. The "huge" coordinate values used above may not be appropriate for your dataset: run program print_coordinates on the `combined.node` file to assess what coordinate values you should use with program remove_range.

QC tasks:

- Visualize the combined object in ParaView, making use of program mesh2vtu as required.
- Perform a visual comparison of the positioning of the combined object with the block objects visualized previously. Check the positioning is consistent with your expectations.

Now use program mesh2poly to convert to a `.poly` file (for use with TetGen), e.g.

```
> mesh2poly combined
```

where you need to specify conversion of the output `.node` file written by program remove_range above.

22.4.5 Use the TetGen meshing program to generate a 3D unstructured mesh of space-filling tetrahedra

You first need to manually add regions to the `.poly` file generated in the previous step, using a text editor. Add the minimum volume constraints for use when running TetGen.

At this stage I typically run TetGen as follows:

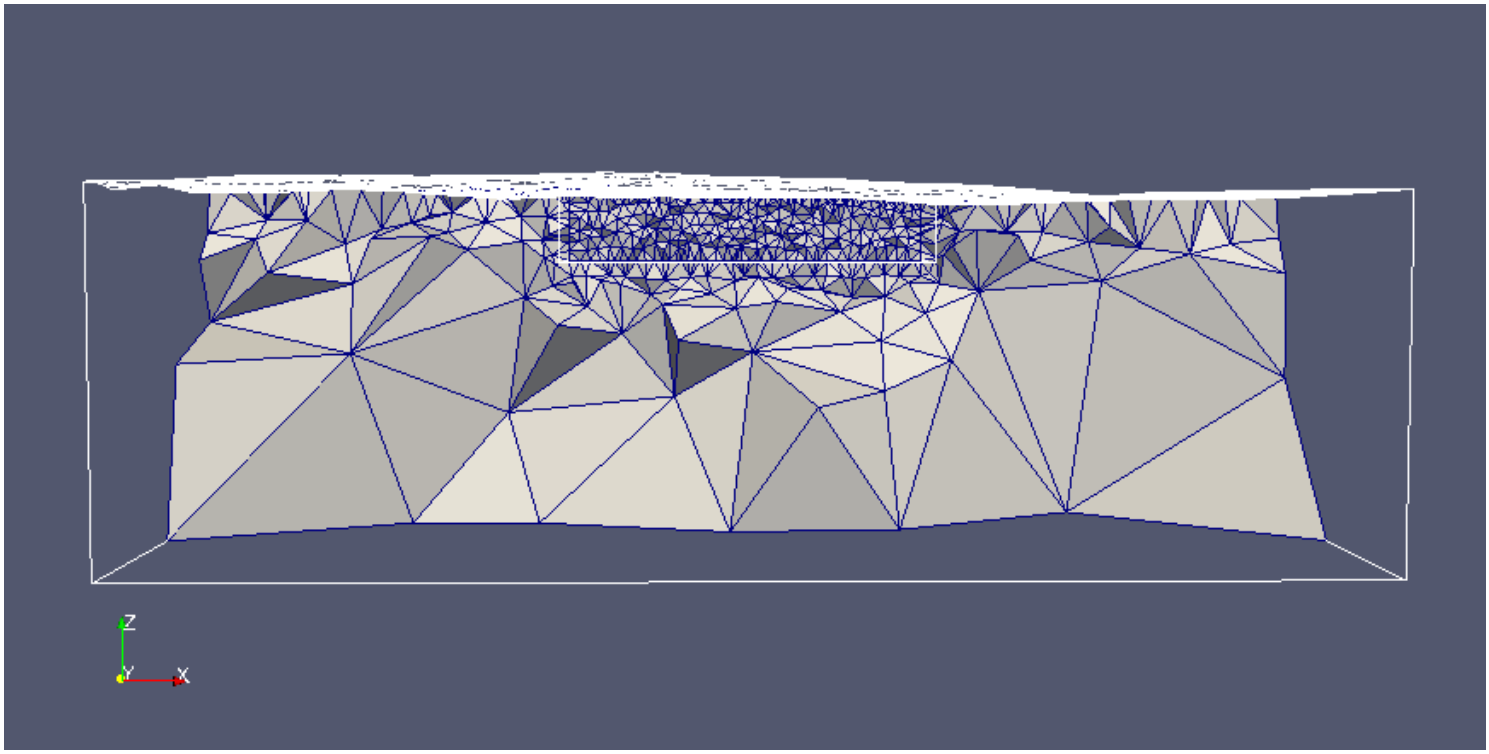
```
> tetgen -pqAanF combined.poly
```

This generates files `combined.1.node/.ele/.neigh` that define a volumetric unstructured mesh of space-filling tetrahedra.

QC tasks:

- Visualize the tetrahedral unstructured mesh in ParaView, making use of program `mesh2vtu` as required.
- Perform a visual comparison of the positioning of the mesh with the combined PLC object visualized previously. Check the positioning is consistent with your expectations.

The screenshot below shows the mesh for the running example: the input PLC (in file `combined.poly`) is drawn white; the mesh is grey with the edges of the tetrahedra drawn blue; I have sliced off cells on the southern half of the mesh to show the different cell sizes inside the volume.



The number of tetrahedral cells in the POI unstructured mesh will depend on two main factors: the maximum area constraints passed to Triangle in Section 22.4.2 and the maximum volume constraints passed to TetGen in this section. The area and volume constraints in the COI are determined by the desired resolution so can't be altered much. In the padding region, I prefer to use a finer meshing for the mesh surface, using Triangle in Section 22.4.2, and a coarser meshing for the mesh subsurface, using TetGen in this section. This generates a mesh with finer cells at the top of the padding region and much larger cells at depth. This strategy is still able to dramatically reduce the number of cells in the POI mesh compared to a rectilinear mesh equivalent, but by building a mesh with smaller cells at shallower depths it makes it easier for subsequent regional-scale inversions to fit the data.

22.4.6 Extract the COI mesh from the POI mesh

You can extract the COI mesh from the POI mesh using program `remove_range`. Make sure you use exactly the extents of the COI.

QC tasks:

- Visualize the COI unstructured mesh in ParaView, making use of program `mesh2vtu` as required.
- Perform a visual comparison of the positioning of the COI mesh with the combined PLC object visualized previously. Check the positioning is consistent with your expectations.

22.4.7 Repeat steps as necessary

You will probably want to generate several meshes with different cell sizes so that you can run test inversions later that take less time. You will want to repeat some of the steps above, at least those in Section 22.4.5 and Section 22.4.6, changing the minimum volume constraints in the `.poly` file that gets passed to TetGen.

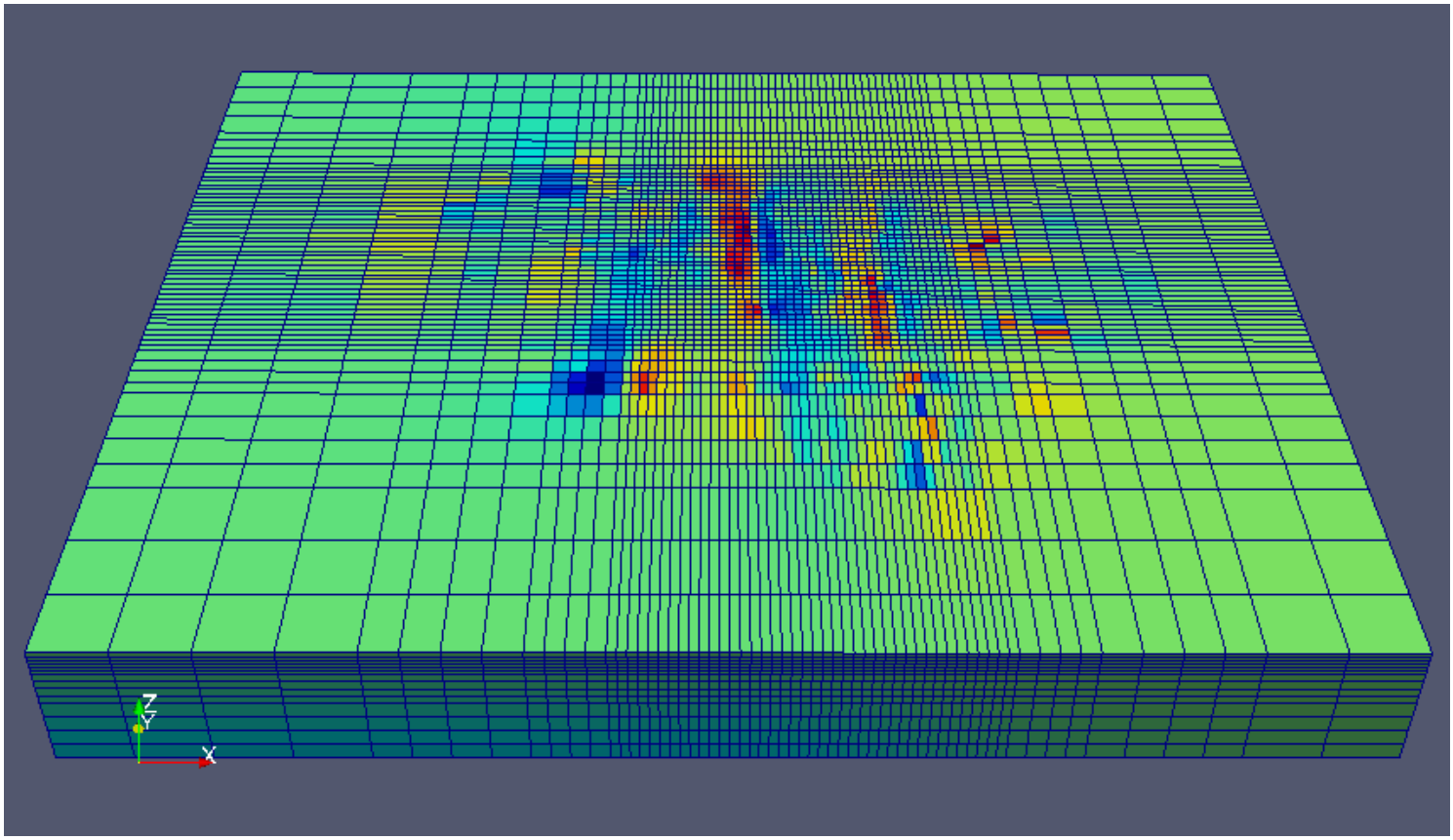
Chapter 23

Regional removal via the scooping procedure

This chapter provides instructions for performing a regional removal for potential field data (gravity or magnetics) using the scooping procedure of Li & Oldenburg (1998b). The procedure requires that you have first performed an inversion on a padded mesh, like those in the workflow of Chapter 22. Section 23.1 explains the scooping procedure on rectilinear meshes and Section 23.2 on unstructured meshes. The aside below further explains the procedure and some nomenclature.

> There is a "regional scale" mesh that contains a "local scale" mesh volume inside it. In the documentation below I refer to these as "padded" and "core" meshes. The "local scale" mesh is the volume on which you are interested in performing finer resolution inversions, on a more finely discretized mesh. The "regional scale" mesh is required for the removal of the regional data component following the approach of Li & Oldenburg (1998b). To perform the scooping, whether you are working on rectilinear or unstructured meshes, you have to make sure that the exact boundaries of the local scale mesh exist inside the regional scale mesh. Usually the process is to build the regional scale mesh by adding padding cells to a coarse version of the local scale mesh, e.g. see Chapter 22, which is why I used the terms "padded" and "core" below.

23.1 Scooping 3D rectilinear meshes



The ParaView screenshot above shows a padded rectilinear mesh containing a model recovered from a regional-scale inversion. The mesh file used, named `mesh.txt`, is as follows:

```
62 59 14
-76000 -68000 0
4*8000 4*4000 4*2000 4*1000 32*1000 4*1000 4*2000 4*4000 2*8000
3*8000 4*4000 4*2000 4*1000 32*1000 4*1000 4*2000 4*4000
6*500 4*1000 4*2000
```

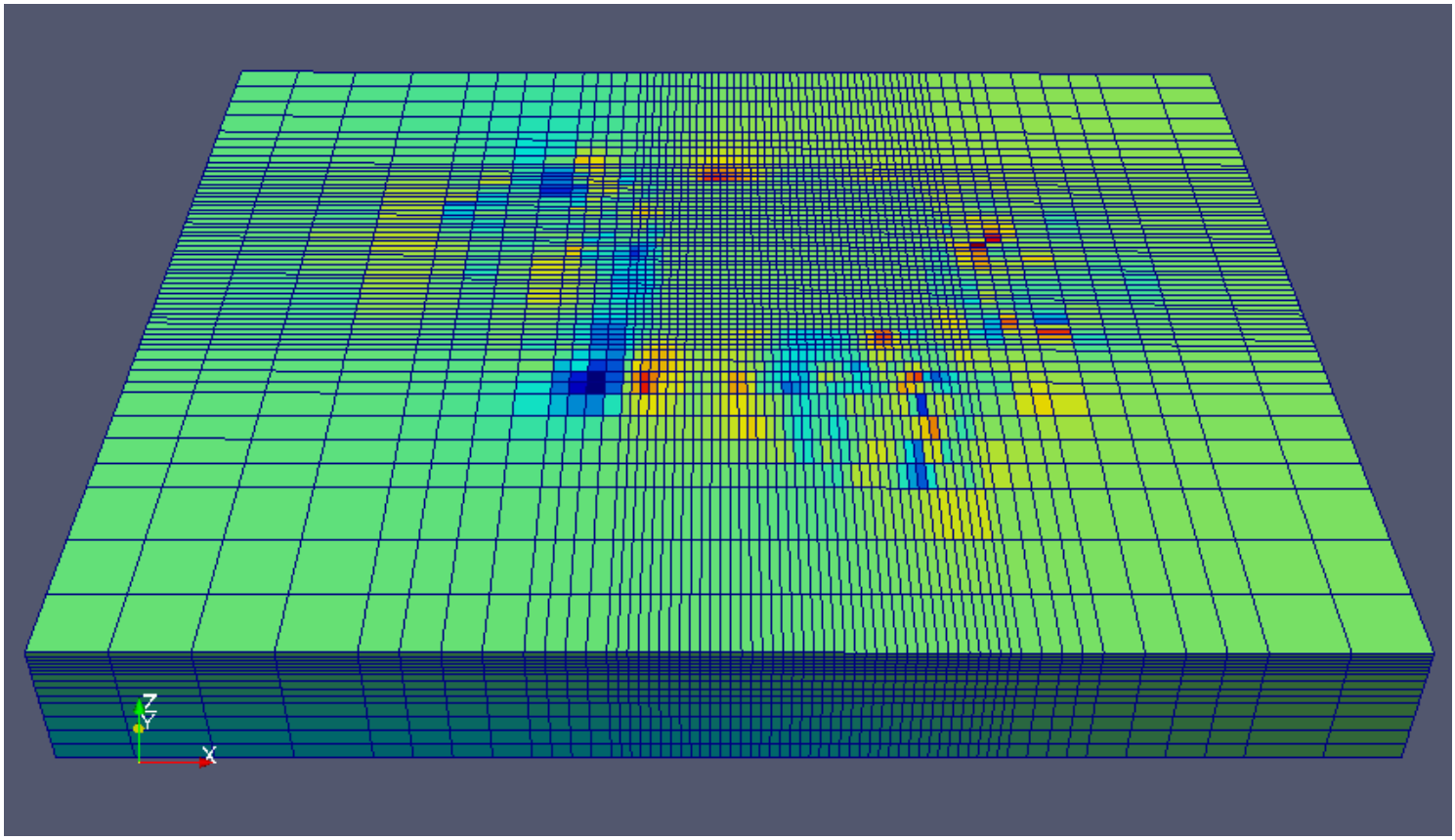
There is a core volume of 32-by-32-by-10 cells and larger padding cells on the outside of that core. The goal is to remove a regional component from the data such that the data only contains a signal from the core volume. To perform the regional removal procedure of Li & Oldenburg (1998b) you must scoop out the material in the core volume. This is achieved using program `add_blocks`. For this example I used the following command:

```
add_blocks mesh.txt model.txt scoop.txt
```

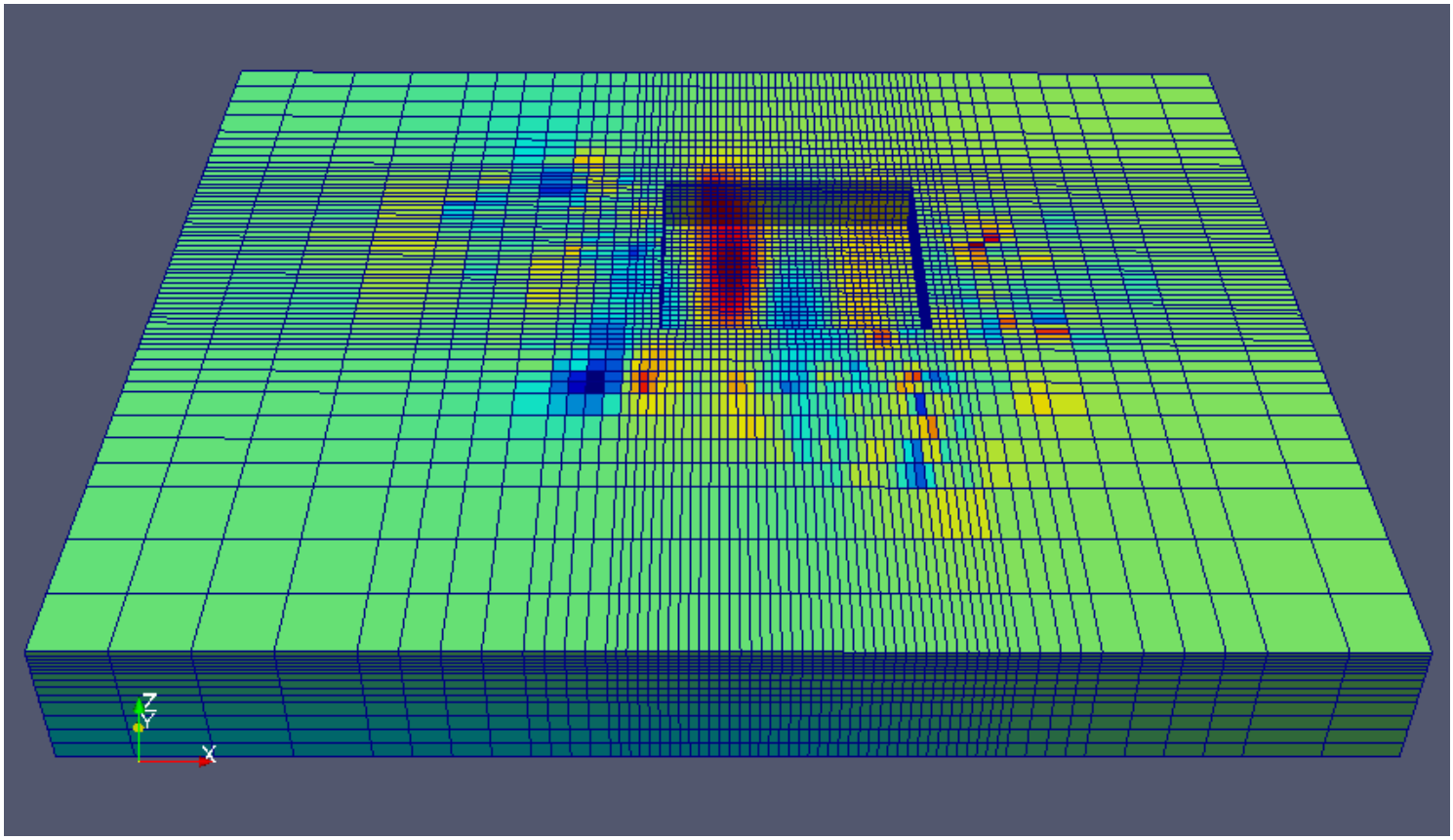
where file `model.txt` contains the model recovered in the regional-scale inversion and file `scoop.txt` is as follows:

```
1 3 0.0
17 48 16 47 1 10 0.0
```

You should read the documentation for program `add_blocks` to understand the contents of that file. **This directly alters file `model.txt` by adding zeros in the core region so you should make a copy of the file if you care about the original being overwritten.** The screenshot below shows the new model with zeros in the core volume.



The following screenshot shows a threshold of that model with any zero values removed.

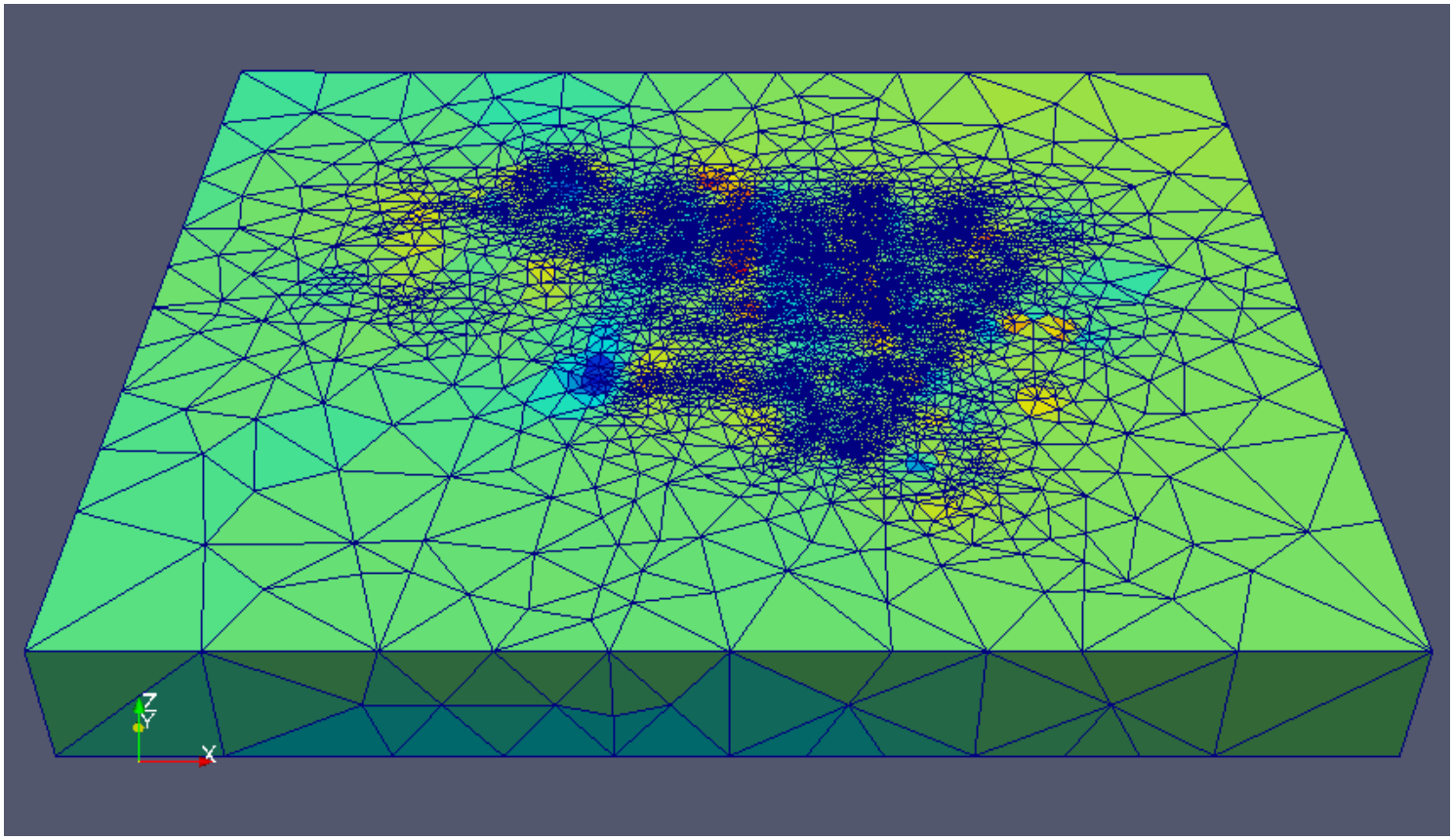


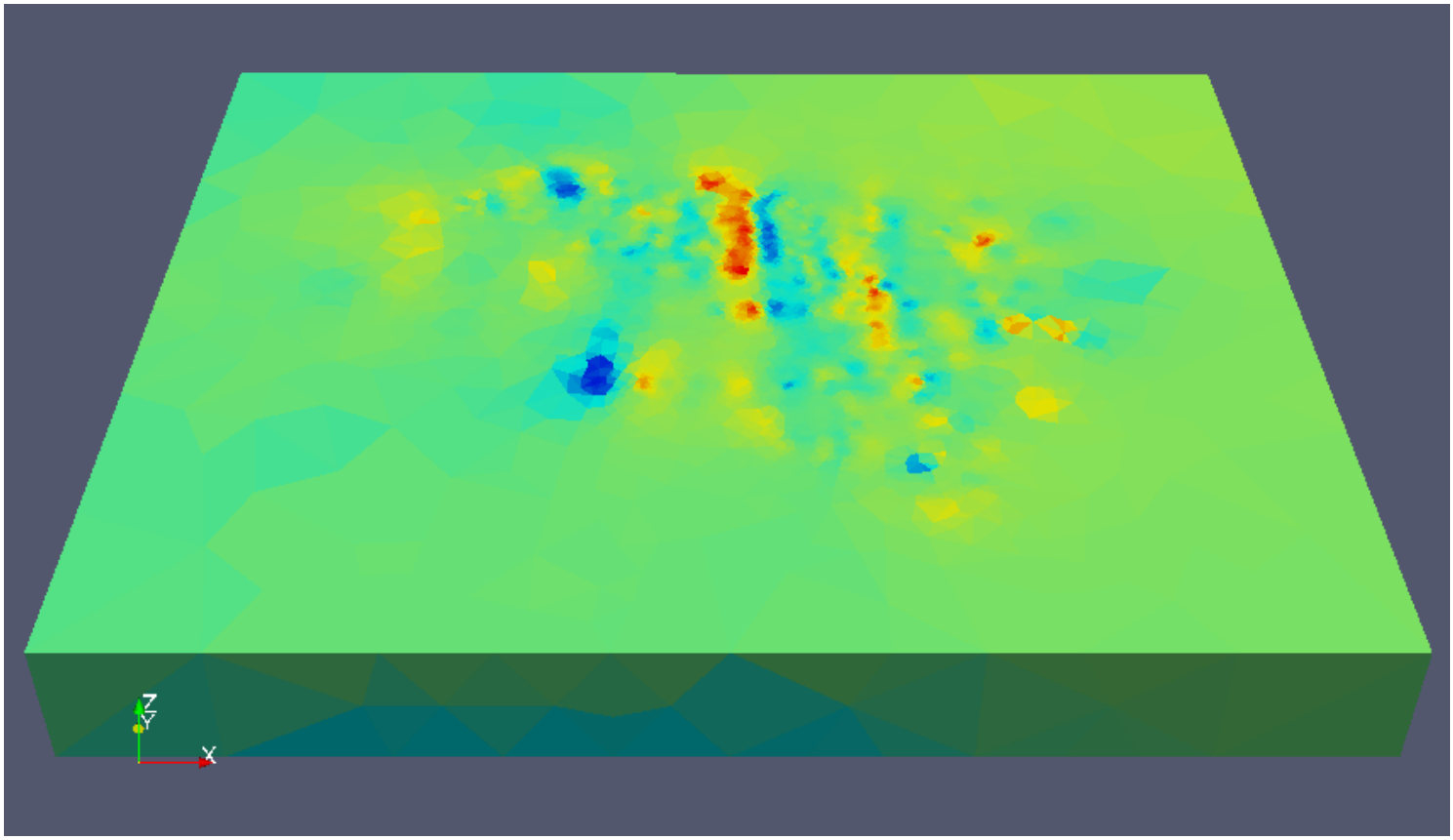
You can now continue with the forward modelling and regional removal steps that are explained further in the Li & Oldenburg (1998b) reference, which I outline here for your convenience but you should read Li & Oldenburg (1998b) carefully:

1. Use program `FOGO` to calculate the data response of the scooped model: the result is the "regional component".
2. Use program `data_difference` to subtract the regional component from the original data: the result is the "regional-removed data".
3. Use program `remove_range` to extract the regional-removed data that lies above the core volume (regional-removed data above the regional padding cells will be close to zero at this stage and should not be used in subsequent inversion on the core mesh).

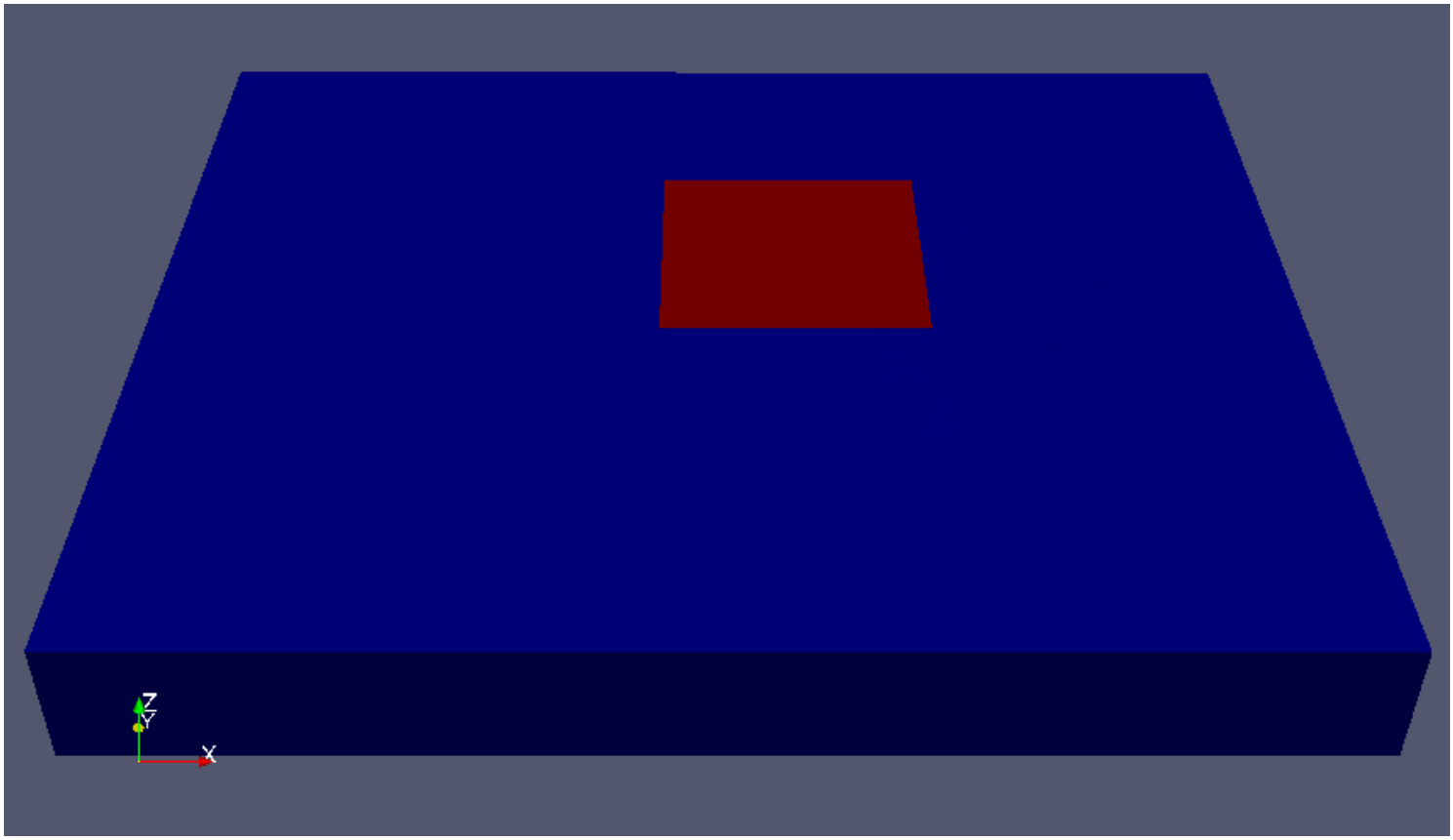
23.2 Scooping 3D unstructured meshes

For unstructured meshes, the scooping procedure is slightly more complicated. You use program `scoop_model`. The screenshots below show a padded unstructured mesh containing a model recovered from a regional-scale inversion, with and without the mesh edges displayed.





That mesh also contains core and padding volumes, although it is more difficult to see the core in the images above. The screenshot below shows the same mesh with the core and padding volumes coloured red and blue respectively.

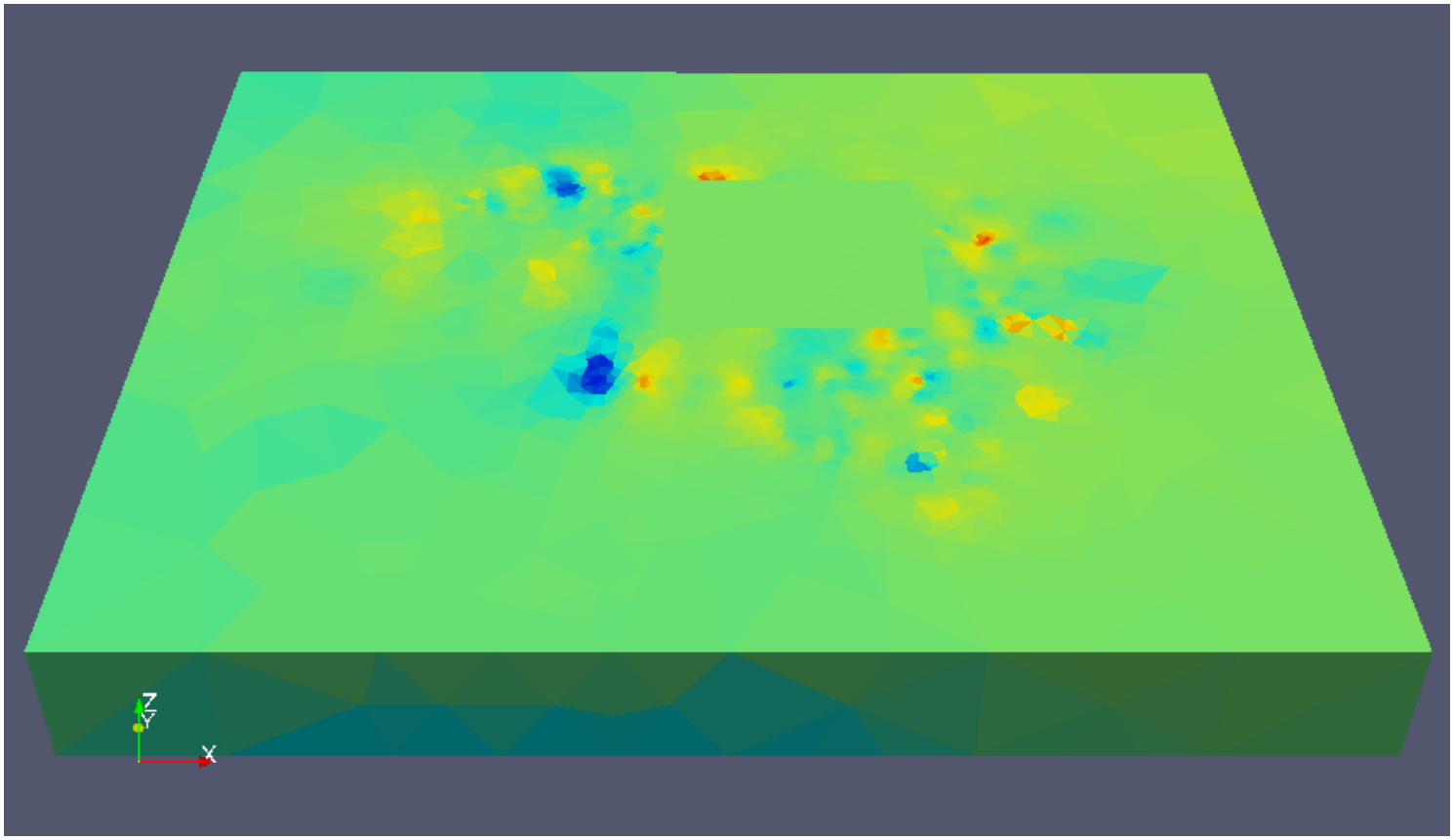


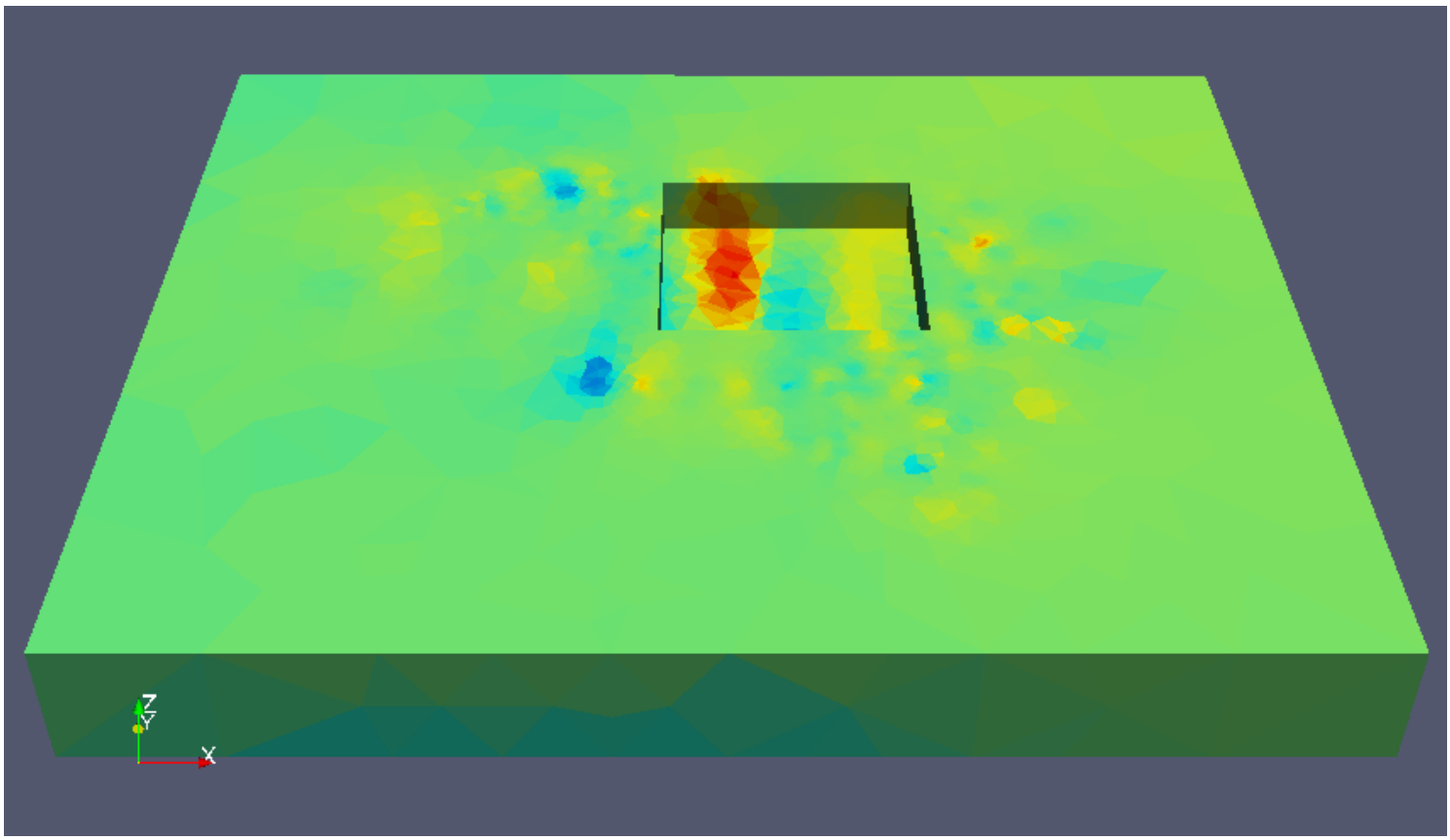
For this example I ran program scoop_model as follows:

```
scoop_model model.1.ele 1 2 2 model2.ele 1 0.0
```

File `model2.ele` is the file containing the model recovered from inversion, i.e. the first two screenshots in this section. File `model.1.ele` is another model **on the same mesh** that would have been part of the inputs to the inversion (see parameter `meshinp` for program `VIDI`). For example, in the workflow of Chapter 22, Section 22.4, I showed you how to generate an inversion mesh with two volumes: when I used TetGen to generate that mesh I used the `-A` switch to assign attributes that identified tetrahedra in the two different regions, and it is those attribute values that are utilize in the call to program scoop_model above. The screenshot directly above is a visualization of that model, with the red cells corresponding to the value of 2 that is used in the command above for the parameters `v1a` and `v1b` (see the documentation for program scoop_model).

The scooped model is shown in the following screenshots. You can now continue with the forward modelling and regional removal steps that are explained further in the Li & Oldenburg (1998b) reference.


Z
Y
X


Z
Y
X

Part VI

For Developers

Chapter 24

Incorporating code for new geophysical data types into MAGNUM

If you wish to incorporate code into the MAGNUM programs (FOGO, VIDI or , DYNO), to perform forward or inverse modelling with a new geophysical data type, then you will need to use the geop_cls interface. In object oriented programming, an "interface" defines the "methods" (in Fortran: subroutines, functions) that can act on an "object" (in Fortran: a variable declared as some derived data type).

The module geop_cls.f95 defines the geop_type data type and associated procedures for general geophysical data. It is the module that provides "polymorphism", another object oriented programming idea: a variable declared as a geop_type can be any general geophysical data type; FOGO and VIDI don't need to know exactly what type of data it is, just how to interact with it via pre-defined methods.

Developers should look at the detailed comments in file geop_cls.f95 for further information and instructions.

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.

K. Deb, et al. (2002). 'A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II'. IEEE Transactions on Evolutionary Computation **6**(2):182–197, doi:10.1109/4235.996017.

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 & C. G. Farquharson (2013). 'Gradient and smoothness regularization operators for geophysical inversion on unstructured meshes'. Geophysical Journal International **195**(1):330–341, doi:10.1093/gji/ggt255.

P. G. Lelièvre, et al. (2011). 'Computing first-arrival seismic traveltimes on unstructured 3D tetrahedral grids using the Fast Marching Method'. Geophysical Journal International **184**(2):885–896.

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, et al. (2014). 'Using an equivalent source with positivity for low-latitude reduction to the pole without striation'. Geophysics **79**(6):J81–J90, doi:10.1190/GEO2014–0134.1.

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.

M. Okabe (1979). 'Analytic expressions for gravity anomalies due to homogeneous polyhedral bodies and translations into magnetic anomalies'. Geophysics **44**(4):730–741.

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.

N. C. Williams (2008). Geologically-constrained UBC-GIF gravity and magnetic inversions with examples from the Agnew-Wiluna greenstone belt, Western Australia. Ph.D. thesis, Dep. of Earth and Ocean Sciences, The University of British Columbia, Vancouver, British Columbia, Canada.