Volsung User Manual
Version 1.15.0

March, 2021

Dr Peter Franz, Dr Jonathon Clearwater

FLOW STATE SOLUTIONS

www.flowstatesolutions.co.nz

Contents

Introduction

"Ill it is to sit lamenting for what cannot be had."
Volsunga Saga

Geothermal reservoirs can be complex beasts to work with. A thorough scientific approach is required to understand them and the right tools and processes are needed to manage them. A key tool in enabling the sustainable, cost effective development of geothermal resources is the reservoir model. Unfortunately, many of the modelling tools used by geothermal operators are not up to standard. In order to use models to make better decisions, operators need modelling tools that are easy to use, fast to run and include the functionality to link reservoir conditions through to fuel supply forecasts at surface. Because existing tools didn't provide these key features we decided it was time for a new one-stop-shop software package for geothermal reservoir modelling.

When developing a new software package one is often in need for finding names for new modules. We started by writing the reservoir simulator and wanted to name it after a fiery, powerful beast that lives underneath the ground - a dragon came to mind. So *Fafnir*- the name of the dragon from the *Volsunga Saga* - was chosen. And - since this saga is rich in diverse characters - other modules and main applications were named after them. In the saga *Fafnir* has two brothers, *Regin* and *Otter*. Associating steel with a well pipe it was a good association to call the wellbore simulator module after the smith, *Regin*. And an *Otter* swimming through channels was a good metaphor for a surface flow network.

Initially only thought as internal names we really liked the characters and so decided to keep them as official release names and name the whole package after the *Volsung* saga. Most of them work quietly behind the scenes, but among the more public characters you will find are the mighty *Brynhild* which adds strength and beauty to our main piece, the reservoir simulation GUI. Similar in a feature role *Gudrun* is our GUI for wellbore simulation. *Sigurd* is working hard behind the scenes fighting dragons aka running coupled reservoir (*Fafnir*), wellbore (*Regin*) and surface network (*Otter*) simulations. Our pipeline simulator *Sigrun* is a little

spin-off from *Gudrun*; *Odin* keeps the oversight and under his other name *Harbard* is responsible for ferrying data from and to remote places.

We hope you enjoy these characters and don't mind us using them. After all we are geeks, and even geeks need to have some fun.

Installation

2.1 Version Numbers

Volsung is released using a major-minor-patch version numbering system. For example, version "1.2.3" indicates major version 1, minor version 2 and patch 3.

Patches are the most frequent updates of the software package; they usually consist of bug fixes, updates in the documentation etc. *Volsung* input files may change slightly but usually without concern to the user. For example a newer patch version may contain a new parameter but existing files will simply read in the new parameter as a new standard value. Output files should not change; however if a bug fix happened in a calculation routine then the newer version may not generate the same output values any more.

Minor version updates occur when a significantly large new features has been added to *Volsung*. For example if a new surface network object has been created then this would be released in a *Volsung* version with incremented minor number. *Volsung* applications should always be able to read model input files with smaller minor numbers. However trying to open a model file with a *Volsung* application which has a lower minor number might cause the application to crash, in which case you would need to update the *Volsung* package before opening this file.

Minor version updates occur when backwards compatibility has been broken. For example, a *Volsung* application with major number 2 may not work correctly or crash when trying to load a model input file with major number 1, or features which had been support in major number 1 might be discarded in major version 2.

We strive to make the *Volsung* package backwards compatible, i.e. you should always be able to read in model files generated by an older version of *Volsung* unless the major release number has changed. However note that while you sometimes can open a model generated by a newer version of *Volsung* in an older version of the application; however if you save such a model in the older application version it may lose important features or information, or it may simply be corrupted. We hence advise against doing so and recommend you always use the latest version

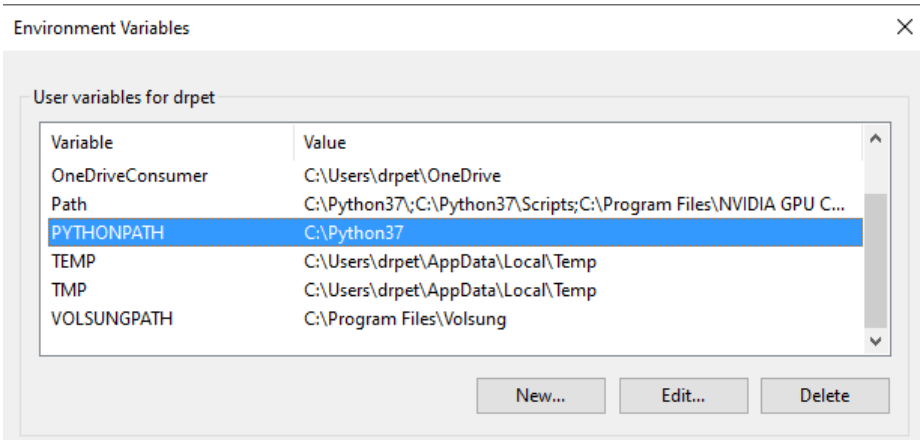

Figure 2.1: Setting the `PYTHONPATH` environmental variable prior to install *Volsung* on *Windows*

of *Volsung*.

2.2 Microsoft Windows Installation

If your system has a *CUDA* capable GPU then start by installing the latest *NVIDIA* driver for your GPU. You can download the driver from `https://www.nvidia.com/Download/index.aspx`. You need at least driver version 460 for the *CUDA* routines in *Volsung* to work correctly.

You can install *Volsung* under *Windows* (64bit) by running the application installer. You will need to run this as an administrator, i.e. by right-clicking the installer and selecting the administrator option.

While running the installer you have the option to install the *Python 3* modules for *Volsung*. For the successful installation of these modules it is important that the `PYTHONPATH` environment variable is set correctly (see figure 2.1); if you have not done so yet exit the installer and set the variable to point to your active *Python 3* installation.

After you have run the installer you will require a license file to run the *Volsung* applications; see section 2.4 for more details.

To uninstall *Volsung* you can use the `unins000.exe` executable which is found inside the installation folder.

2.3 Linux Installation

The installation of *Volsung* on *Linux* systems (64bit AMD) is facilitated via *snap*. This makes the installation straightforward; *snap* applications can run on many different *Linux* distributions without problems.

If you plan on using an *NVIDIA* GPU as an accelerator please install the necessary video card drivers first. For example in *Ubuntu* you would do this via

```
$ sudo apt install nvidia-driver-460
```

Note you should opt for the latest driver; version 460 is the minimum requirement for *Volsung*. Also, if you install the *NVIDIA* driver you will need to reboot the machine before you can use the GPU accelerators in *Volsung*.

Before starting the installation you need to ensure that you have *snapd* installed on your *Linux* distribution. Please go to `https://snapcraft.io/volsung` and consult their documents on how to achieve this on your system if *snapd* is not installed on your system by default. In the simplest case for *Ubuntu* you will only need to execute

```
$ sudo snap install volsung
```

which will automatically download and install *Volsung* from the *snap* store.

Currently *snap* can not register new file associations (MIME types). In order to correctly create these you can run the script

```
$ sudo /snap/volsung/current/register-mime-types
```

which will associate files ending in `.brynhild` with the *Brynhild* application etc.

Due to the fact that *Volsung* uses *snap* containers it is currently not possible to automatically install the *Python 3* modules automatically. Hence you need to install them manually using

```
$ sudo /snap/volsung/current/install-python-modules
```

After the successful installation you will need to obtain and set a valid license file, see 2.4.

You can uninstall *Volsung* via the command

```
$ sudo snap remove volsung
```

Applications installed via *snap* will automatically update themselves from the *snap* store whenever a new version of *Volsung* is available; *snap* by default will check four times per day if newer versions are present in the store. On very rare occasions when using a remote you may have issues when your local machines has already

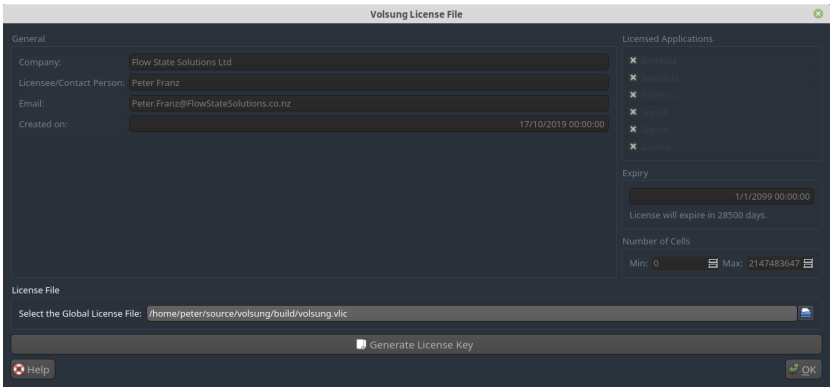

Figure 2.2: The license information dialog.

updated but the remote has not. In this case you may need to run the following on the remote to force an update:

```
user@host $ sudo snap refresh
```

2.4 License Files

The *Volsung* license is bound to a single PC via unique internal identifiers. All applications can only be started if a valid license file can be found. The only exception is running of *Sigurd* on a remote linux machine; here the process on the remote can be started from a licensed PC without having a license file present on the remote.

The license file information will come up automatically on starting a GUI if the license is invalid. Alternatively it can be accessed through the help menu. To obtain a valid license from us please push the *Generate License Key* button and send us the garbled key information via email - note it will be automatically copied to the clipboard so you can paste it easily into an email.

After obtaining a valid license file from us you will need to copy it to a local folder in your machine and point the license dialog to its location, as shown in figure 2.2.

After accepting the dialog you should be able to start the applications licensed to your machine.

2.4.1 Headless Machines

If you intend to use a licensed version of *Volsung* on a headless machine (i.e. one without a display) you will need to contact us for more instructions on how to

obtain a valid license file.

After obtaining a license file you will need to set it from the command line by using

```
$ volsung.set-license-location path/to/my/license-file.vlic
```

2.5 External Licenses

The *Volsung* package is built using a variety of external open-source libraries. The licenses associated with these libraries can be found in the application folder. They are located in the `licenses` subfolder.

The Brynhild GUI for Reservoir Modelling

The *Brynhild* graphical user interface is the heart of the *Volsung* software package. It allows you to create new models or make changes to existing ones.

In this chapter we will focus on how to set up a model and work with the features in *Brynhild*; the mathematical/physical aspect of the simulation software will be treated in separate chapters.

We will begin by introducing general settings and features; after that we step through the typical process of generating a model.

3.1 General

The general layout of the *Brynhild* GUI is shown in figure 3.1. We will use this figure to introduce the different parts of the GUI:

- Just below the menu bar of the GUI you can see the *time tool bar*. In general this toolbar can be used to display simulation results at the print times.
- On the left hand side below the time tool bar you can see the *navigator* panel. You can use the *items* in this panel to quickly navigate to the settings you want to change. In general you interact with these items by either double clicking, right clicking or expanding them by clicking on the small arrow symbol to their left.
- On the right hand side below the time tool bar is the main *viewer window*. Its content can change between different viewers; for example by double clicking on the "Reservoir" item in the navigator you can bring up the reservoir viewer which is currently shown in figure 3.1. Other choices can be the flow network editor or the simulation log.

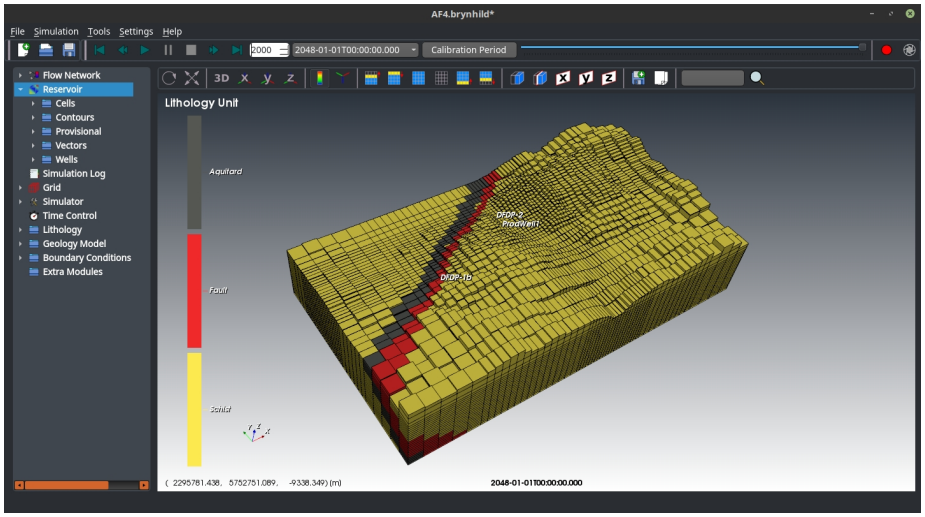

Figure 3.1: The general layout of the *Brynhild* GUI

3.2 Menus

The submenus of the *File* menu are self-explanatory.

The *Simulation* menu contain actions for running the simulation or to change the equation of state.

The *Tools* menu contains useful tools like a diff-tool which can be used to find differences between different versions of a model.

3.2.1 The Settings Menu

The *Settings* menu has actions for changing important settings. Some of these settings are applied throughout the whole *Volsung* package. For example changing physical units will affect the settings in all applications of the package which make use of them.

Setting Units for Physical Quantities

You can select the units for physical quantities in the *Settings/Units* menu. Choices you make in here affect the GUI immediately, i.e. you will see the units in tables and graphical displays to change when you alter a unit setting.

Note that altering the units temporarily can be a very useful feature when you want to import data via copy & paste. Simply change the units to match the ones

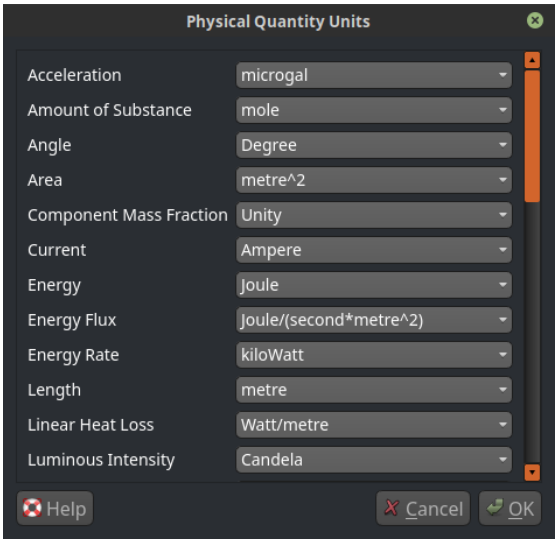

Figure 3.2: Dialog for changing the units for physical quantities

which you want to import, paste your data into the appropriate table and then change the units back to the ones you prefer.

If the unit you want to work in is not available please contact us so we can add it to the collection.

Pipe Data Base

A simple data base is available for storing default pipe configurations. These can be used when creating a flow path configuration for wellbore and pipeline models.

You can edit your own configurations into the database. Restoring the defaults will erase all your edits and reload the pre-configured data base.

Remotes

This menu can be used to manage remote hosts to which you can off-load time intensive simulation runs. Using these and how to run on remotes is explained in appendix B.

Preferences

The preferences dialog contains diverse settings:

- The Diff Tool section allows you to select an external tool for diffing text files, for example *meld*.
- The Zoom tab allows you to change the magnification of the fonts and objects in the application. You will need to restart the application for changes to take effect.
- The Locale settings allow you to modify the language and country settings. Note the language in the application will remain English; the setting is only used in combination with the country setting to determine the locale. You can use custom date input format to specify how you want to input dates.

3.2.2 The Help Menu

The *help* menu has simple actions to display contact information (*about*) or to bring up this manual in its online form.

The *license* menu displays license information; see section 2.4 on how to configure the licenses to run applications in the *Volsung* package.

3.2.3 The Tools Menu

3.2.4 Thermodynamic Table Tool

You can use this tool to calculate the thermo-physical properties of a thermodynamic state. See 12.2.1 for more details.

3.2.5 The Model Version Control Menus

Geothermal models often have a long development process in which dozens if not hundreds of models are run when trying to optimize the parameters used. It is quite easy to lose oversight over which parameters have been changed in different model versions. Also sometimes models are quickly discarded and it is later on discovered that they were useful. This provides the modeller with the dilemma on keeping track on the model versions developed.

There are two principal work flows which can be used to keep track of model versions. The first one is to use a full version control system, for example *git*[1]. These version control systems can be really powerful tools; however they are usually geared towards software code developers and can be intimidating to use for the novice. Further they require strict discipline.

[1] `https://git-scm.com`

Volsung offers a second, simpler solution which makes it easier to store model versions and determine the key differences in parameters to the current model version. This in-built version control system is not as powerful or complete as *git* but it addresses the key requirements for geothermal modellers.

The key concept in this system is that you can easily *stash* a version of a model. This stashed version is copied into a subfolder of the current project folder. You can then browse through stashed model versions and quickly see their version information, i.e. title and description. You can quickly run a diff tool (see the subsection below and section 3.2.1 on how to set one up) between the current model and a stashed version; this diff tool can show you all changes in the raw model input files. You also have the options to save a stashed version as a new file or restore it as your current model.

Further you have the option to diff the current model to other files, i.e. if you have stored another version of your model in another folder. Models developed in the *Brynhild* GUI also have the possibility to diff versus the model version stored in a results file.

Setting a Diff Tool

The diff tool is a useful feature to track differences in model input files, for example if you have different versions of a model on your drive. Using the diff tool you can establish which parameters have been changed between these models.

To use the diff tool you first need to install an external diff tool. We recommend *kompare* or *meld*[2]; if you are using *Linux* then *kompare* should already be installed on your local machine and set up to work with *snap*[3].

After you have installed the diff tool you need to set its command line arguments in the preferences dialog.

3.3 Generating a new Model

Select the *File/New* menu. A dialog box will pop up asking you what kind of model you want to generate. At this stage you have the following choices:

- You can select a *Fafnir* model. This will create a model with all features available in *Brynhild* and should be the standard choice when creating new models.

[2] `http://meldmerge.org/`

[3] *Linux snap* installations work in a confined environment; you may need to alter the confinement level of the *Volsung* package if you want to use a different difftool to the one set up as default.

- If you have a pre-existing *TOUGH2* model you can select a *T2Fafnir* model. In this model you will use your *TOUGH2* input file and an associated grid file to run your model. The new model will take all its necessary information from the *TOUGH2* input file and the grid, hence you can not edit parameters for the rock types, time settings etc. However you can still integrate new wellbore/surface models into the existing *TOUGH2* model.

For the rest of this chapter we will focus on *Fafnir* models only; the available features for *T2Fafnir* models are the same as for *Fafnir* models. Appendix C gives an overview over how to run *TOUGH2* models.

After you have generated a new model you should save it in a new folder, i.e. keep it separated from other models you may have created already.

3.4 Generating a Grid

Fafnir requires a grid to run. This grid is loaded by using a grid file which can be either a compatible file generated by another program or a file you can generate yourself from within the GUI. You can select the file to use by expanding the *Grid* item in the navigator and selecting the *Grid File*.

Compatible external file formats are:

- *MULGraph* files like those generated by *PyTOUGH*. Only the grid geometry will be taken from these files, i.e. other objects described in the file are ignored.
- *VTK* files, for example those of type *vtkUnstructuredGrid*. Care must be taken during their construction to ensure that cells which share a face make use of exactly the same points (by index, not location) for the shared face. Optionally these files can contain a cell array called `Name` which is either of type *vtkStringArray* or *vtkCharArray*. This array is used for naming the cells and is a strict requirement when running *T2Fafnir* (see appendix C). Another optional array is an integer array called `Cell Layer` containing the layer index, starting with zero at the bottom. This is used only for visualization/navigation purposes in the viewer. The example *Python* script `examples/createVTKGrid.py` demonstrates how a customized grid can be generated.

Alternatively you can generate a grid from within *Brynhild*. Simply right-click on the *Grid* item in the navigator, select *New* and then choose the grid type you want to generate. A dialog will appear in which you can set the grid parameters; when you accept the dialog it will ask you for a file name for the grid file. You can later on edit the grid parameters again by double-clicking on the *Grid* item.

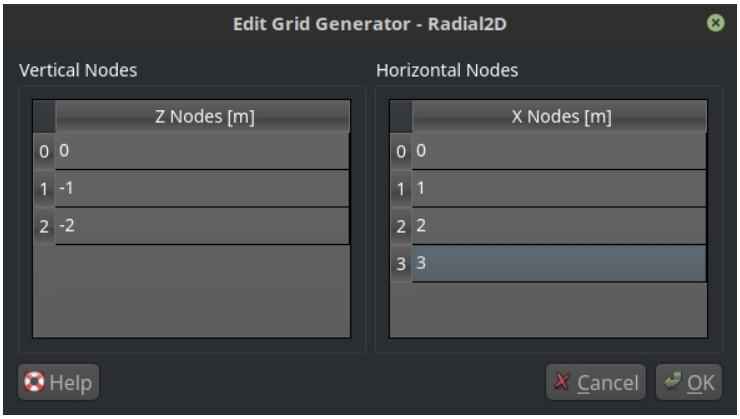

Figure 3.3: The Radial2D grid generator dialog

You also have the option to rebuild the grid during simulation runtime. This is usually not required; however it can be useful when using *PEST*, for example when changing the dimensionality parameter in a grid used for numerical pressure transient analysis.

If you are building a model from scratch it is at this stage nice to visualize the grid by enabling the *Reservoir/Provisional/Grid* item. You can deselect this switch later on when you have simulation results to display.

3.4.1 Radial2D

Figure 3.3 shows the grid generator dialog for generating grids with radial symmetry. You need to provide a list of vertical and horizontal nodes. The resulting grid will visually appear like a pie-segment but internally uses the correct block volumes and connection areas. But note that for looking up geometric locations - for example if you want to insert a well - the grid geometry as visualized is used.

3.4.2 Radial PTA

This logarithmically spaced radial grid (see figure 3.4) is tailored towards numerical pressure transient analysis (PTA). It is based on the numerical PTA framework [Zarrouk and McLean, 2019, McLean, 2020].

The grid contains a single layer described by the layer thickness. Three zones are set up - the well, skin and reservoir. The well is made out of a single block whose internal volume and radius can be controlled.

The skin zone is the transition zone between the well block and the outer reservoir.

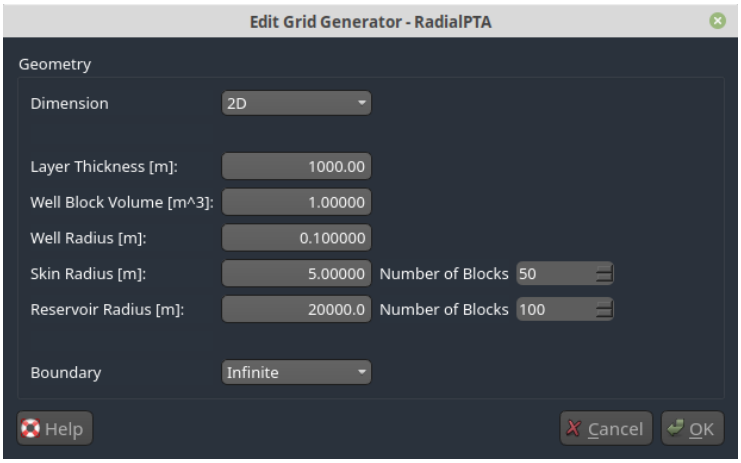

Figure 3.4: The RadialPTA grid generator dialog

Its number of blocks can be controlled. The reservoir zone is the zone outwards from the skin zone. Both skin and reservoir zone have logarithmically spaced grid dimensions.

The dimension setting describes the flow pattern from/towards the well block and can be fractional. The boundary can be infinite or can describe a linear or channel boundary. We refer to Zarrouk and McLean [2019] for the details on this.

3.4.3 Tartan

The tartan or recti-linear grid is one of the most commonly used grids in reservoir simulations.

Brynhild allows you to use local or global coordinates (for example NZMG) when setting up a tartan grid. Local coordinates, for example $x = [0, 10, 20, 30]$ are easier to deal with when defining spacings. Figure 3.5 shows the transformation options. You can use the azimuth angle to rotate the local grid around its origin, then translate it to its global position using the base point parameters.

You can use an undulated surface for the grid by supplying a list of surface points as shown in figure 3.6; this list can be either in local or global coordinates. When building the columns of the tartan grid the elevation at their cell centre is interpolated using this point list; blocks in the column above the interpolated elevation are cut out of the grid and the block through which the surface protrudes will have its height set accordingly. You can use the minimum cell height setting to ensure that cells in the surface layer are not too thin.

Finally you will need to provide lists of vertical (see figure 3.7) and horizontal (see figure 3.8) nodes.

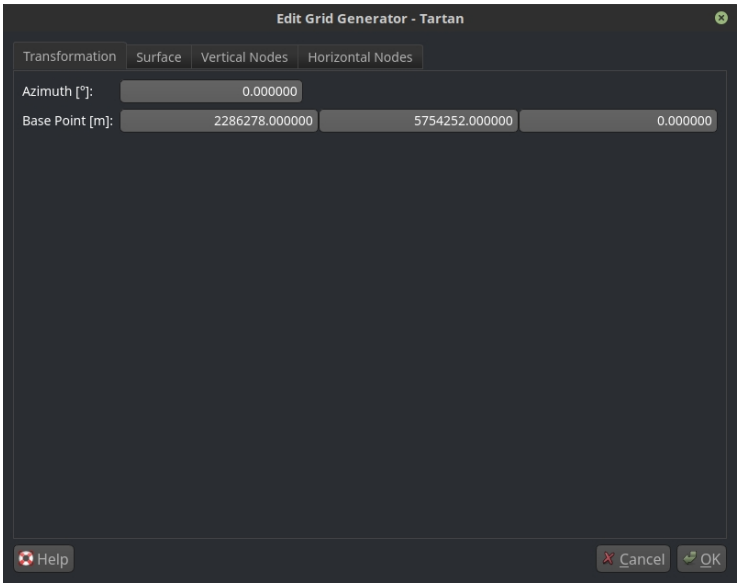

Figure 3.5: Tartan Grid Base Point

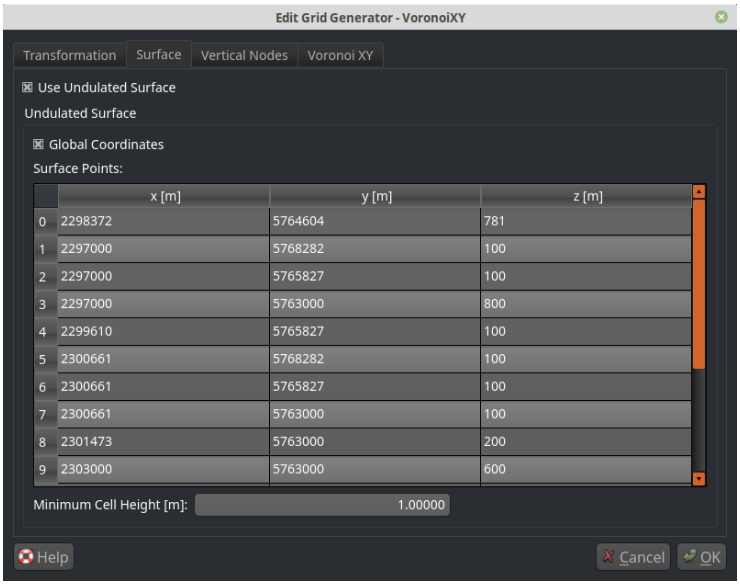

Figure 3.6: Undulated Surface option by using a list of surface points.

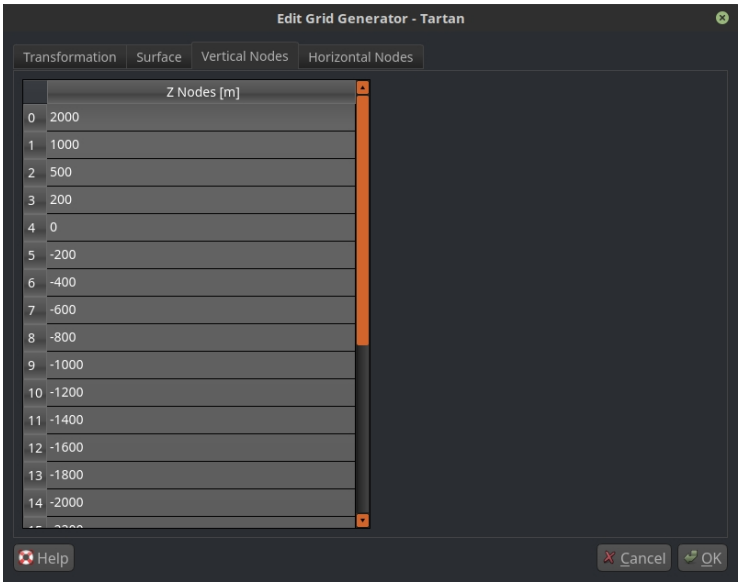

Figure 3.7: Nodes to specify grid vertical layers.

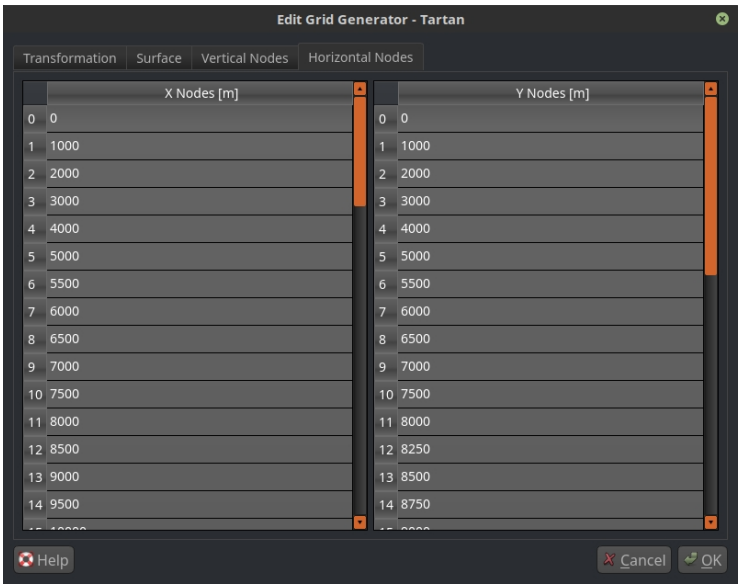

Figure 3.8: Nodes to specify horizontal grid structure.

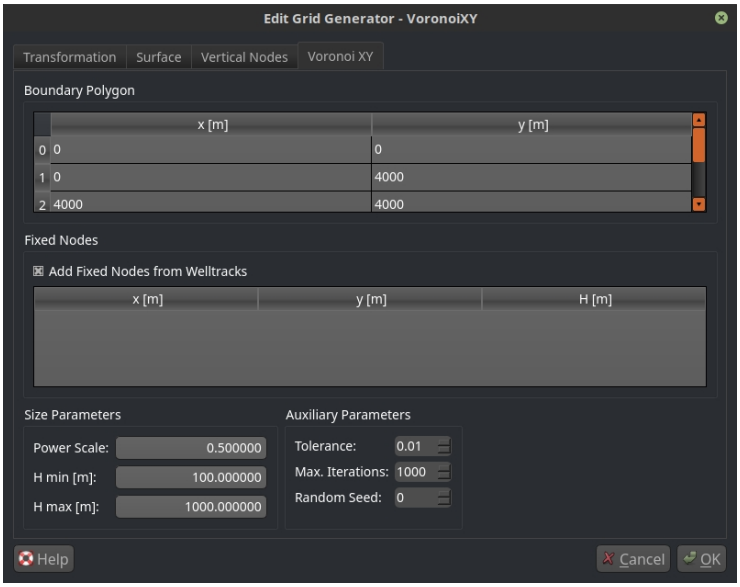

Figure 3.9: The Dialog options for setting up a Voronoi grid in the (x, y) plane

3.4.4 VoronoiXY

You can set up a Voronoi grid in (x, y) which is then extruded in z direction; sometimes this is referred to as a $2.5D$ Voronoi grid.

The first three tabs of the dialog for this grid generator are exactly the same as for the tartan grid (see **??**). The third tab as shown in figure 3.9 deals with the creation of the Voronoi grid.

The boundary polygon is used to cut the raw Voronoi grid - which is larger and has jagged edges - to its final shape. The example in 3.9 shows how to setup a rectangular boundary polygon. Note that you need to use local coordinates here but of course you can make use of the transformation to global coordinates by setting a base point and azimuth in the transformation tab.

Fixed nodes with local refinement can be introduced as a list in the (x, y) plane. You can also opt to automatically use welltrack coordinates as refined areas; however note that the welltracks must already be present at the time of the grid generation, i.e. the grid will not be changed if you later on add or remove wells to the model.

The size parameters are used to determine the typical cell dimensions. H_{min} and H_{max} describe the minimum and maximum approximated cell diameter, respectively. The power scale determines how quickly cells get larger when moving away from the refined areas around the fixed nodes. Larger values for the power scale will result in a more rapid transition and less cells. Note you should be careful when

using smaller values than 0.5 for the power scale since you can end up generating too many cells and the grid generator might appear stuck for a long time.

The grid generator uses a relaxation method to transform the cells into approximate centroidal Voronoi cells. You can alter the maximum number of iterations; larger values generate better approximations of centroidal Voronoi cells. The iteration is stopped if the Voronoi sites move by smaller distances than given by the relative tolerance.

If you do not like Voronoi grid instance generated you can generate another instance using the same parameters by changing the random seed.

Note that the underlying Delaunay triangulation algorithm can be very sensitive. If you notice "weird" looking cells or cell boundaries criss-crossing the grid then repeat the grid generation with a different seed. Sometimes it is also possible to slightly change the number of iterations to get rid of these artefacts.

3.4.5 VoronoiXY (Manual)

You can set up a Voronoi grid in (x, y) which is then extruded in z direction. You will need to specify the (x, y) coordinates of the Voronoi centres manually. No optimization or automatic refinement is performed.

3.4.6 MULGraph

While MULGraph files can be natively read by *Volsung* you can also use a grid generator to perform this job. This gives you the option to transform the grid via an azimuth rotation angle and a basepoint translation; this is a useful option when the MULGraph file contains local vertex positions. Figure 3.10 shows the dialog options.

3.4.7 TOUGH2-Regular

This grid generator attempts to extract the grid geometry from a *TOUGH2* input file. The basic underlying assumption is that the grid is a recti-linear, regularly spaced grid with parallel elevation layers for `ISOT=3`, has no manually specified MINC elements and all its geometric properties like volumes, areas and connection distances are consistent with the geometry. The geometry is established using the grid centres specified in the `ELEME` section and the connection distances from the `CONNE` section. Missing connections, i.e. for the boundary blocks, are completed if suitable mirror connections to the other side of the boundary block can be found. Elements with no grid centres specified will not be included in the final grid, i.e.

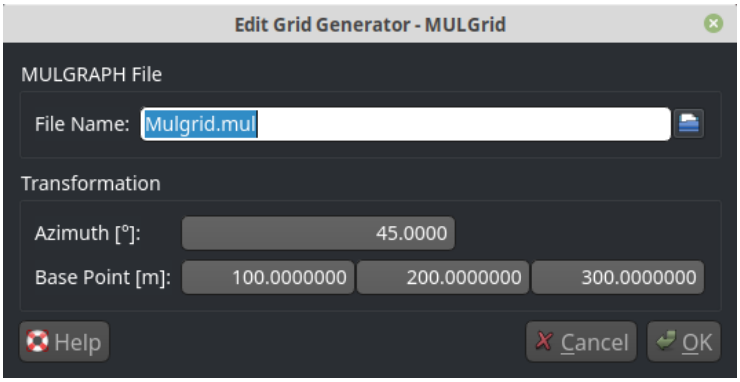

Figure 3.10: The Dialog options for setting up a MULGraph grid using azimuth rotation and basepoint translation.

if the *TOUGH2* input file contains boundary conditions like an atmosphere or hot plate ensure that they use blank characters as grid centres.

Establishing a grid from *TOUGH2* input files is difficult since it has incomplete data, can be illogical and has limited precision, so it is possible that the grid generated with this generator is not suitable for numerical simulations. In particular if the grid centre positions or connection distances are given in floating point notation, e.g. $1.2345E+08$ the precision may be too poor to correctly build the grid.

3.5 Changing the Equation of State

If you require a different Equation of State you can select the *Simulation/Change Equation of State* menu. You will be given a choice of equations of state available.

If you select a different EOS your current model will be saved; after this you will be asked for another filename for the model with the changed EOS.

When changing the EOS *Brynhild* will aim to preserve your current mass fractions for the components; this of course works only if the new EOS contains some of the original mass components. Pressure and enthalpies in the model are conserved. However you need to consider that using a different set of mass components may result in different thermodynamic states when using the same pressure and enthalpy, so you need to manually check your initial state settings to ensure that the new initial states are acceptable to you.

3.6 Running a Model

At this stage you will have an empty model with no boundary conditions attached. There will be some default material functions, lithological units and initial conditions. The following chapters will go into further details on how to set up the reservoir and surface network simulations.

To run your model you can select the *Simulation/Run (F5)* menu. However a useful feature is to make a short *test* run first by using *Simulation/Run Test (F6)*. A test run will go through all sets of running the model but will not step through time. It is useful for error checking and generating some basic output which will display the initial state and the lithology of the model.

Another run option is to run your model on a remote host; how to do this is explained in detail in appendix B.

Once you have run the simulation at least once you can also run a surface flow network simulation on its own by using *Simulation/Run Flow Network (F9)*. The flow network simulation will be run at the current time as selected in the time tool bar. Its results are transient, i.e. as soon as you select another time the simulation results will be lost. The main purpose of this simulation mode is to provide a simple means to find errors in the flow network or to test different parameters before running them in a time-consuming coupled simulation.

While running the simulation the view will be automatically changed to display the simulation log. This log view gives you important details while running the model, for example the time and step progress towards the end of the simulation in form of progress bars. The log also contains important information like errors/warning which appeared over the simulation time.

If you want to prematurely quit a simulation you have two option buttons in the simulation log viewer:

- You can *kill* a simulation. This means the simulation quits instantly. The simulation result files will be corrupted and you won't be able to see the results up to the current point of simulation.

- Most of the times it is hence advisable to softly *terminate* a simulation instead. When receiving a terminate signal the simulation will finish its current time step and properly write out the result files before quitting. Unfortunately the terminate option is not available on the *Windows* platform, so terminating a simulation under *Windows* will simply kill it.

3.7 The Reservoir Viewer

The reservoir viewer is the main tool for displaying results from the reservoir simulation. Other parts of the simulation, like the flow network simulation, have their own viewers and are explained elsewhere.

Most of the user interaction with the viewer is quite intuitive using the viewer's tool bar. We encourage you to play around with the tools in there until you have familiarized yourself with their use.

You can rotate/zoom/move the displayed reservoir model by using the mouse buttons in combination with moving the mouse (for example for rotation/translation).

You can use the left mouse button to select individual cells of the model. If you press the `Ctrl` key you can extend the selection. Right-clicking on selected cells will give you more options, like selecting cells in the columns below or above the current selection and showing only the selected cells. Selecting the rubber band in the tool bar toggles rubber-band mode on or off. In rubber band mode you can select more cells by left-clicking the mouse and dragging it over the cells you want to select.

Note some geology models offer more choices as well for selected cells.

If you want to change the displayed quantity for the model you can right-click on the colour scalar bar on the right hand side of the viewer and select the property to display from the list which pops up. The range of the scalar bar is usually set to automatically reflect the maximum/minimum values found in the current property under display but you can manually select a range and colour options by double clicking on it.

Other properties of the viewer are controlled using items in the navigator panel:

- You can change the cell properties by changing the MINC layer you want to view; you can also select the property array to view just like you would have selected it in the colour scalar bar.
- You can display contours, i.e. iso surfaces, of the cell properties. Note that the range and colour options for the contours are taken from the colour scalar bar, so if you want to generate contours in a certain value range you need to set this range for the property in the scalar bar first.
- It can be useful to provisionally display the grid and wellbore tracks. This is an option most useful when generating a simulation from scratch so you can see where to place geological elements and boundary conditions while no valid simulation results are available. Later on you may want to switch these display options off.
- You can visualize flow vectors.

- You can also set a radius factor for displaying the welltracks; this is useful to make them visually thicker.

- The wells can be coloured by interpolated field data. To do so you must specify the well's field data in the flow network editor. For colouring the field data closest to the currently displayed time will be selected and an interpolation method is used to determine the colour at points along the welltrack of geometric interest. This feature is very useful when calibrating a reservoir model to temperature data collected along welltracks.

3.7.1 Making a Movie

You can make an animated movie using the actors displayed in the reservoir viewer. To do so click on the *record reservoir movie* item in the time tool bar. You will be asked for a filename to save the movie to and to enter a frame rate; the higher frame rate you select the faster the movie will change frames when playing.

The simplest way of recording frames for a movie is by clicking the *start* item in time tool bar. Whenever a new time is loaded into the reservoir view a frame will be recorded. You can pause this process in order to change the view or select different properties to display.

Alternatively you can manually record frames by clicking on the *shutter* item next to the record button.

Once you have finished the movie you need to click on the record item again to finalize it.

Note the only file format presently supported is the open source *Ogg Theora* format.

3.7.2 Props

Props are additional items which can be displayed in the reservoir viewer. They are purely visual, i.e. they do not affect any calculations.

Slice Images

A slice image allows you to place an image from a file into the $3D$ scene of the reservoir. You will need to provide a suitable image file and you need to specify the top left, top right and bottom left coordinates for the image.

Surface Images

A surface image allows you to project an image from a file onto the top of the reservoir. This can be useful for using maps featuring roads, infrastructure etc. You will need to provide a suitable image file and you need to specify the top left, top right and bottom left (x, y) coordinates for the image.

Micro Earthquakes

You can import a set of micro earthquake (MEQ) events and display them in the reservoir. You have the choice of displaying all of the events or only those prior to the current display time (cumulative). As colour options you can select a base colour and a colour for active events. Active events are those events which occurred between the previous and the current display time.

The events will be displayed as spheres with the volume proportional to the event magnitude.

3.8 Extra Modules

Extra Modules contain optional calculations during simulation runtime. These calculations do not influence the core simulation but usually provide additional output like micro gravity calculations. Right-click on the *Extra Modules* item in the navigator to create a new module.

Note that renaming extra modules can lead to loss of simulation data, i.e. you will need to re-run the simulation again.

3.8.1 CSV Element Data Writer

This module creates a simple comma-separated-value output file for the elements located by the positions you specify. In case the model is MINC'd then all internal layers of the specified cell are printed. Output is generated at the normal print times and is written to the file `ElementData.csv`.

3.8.2 Flux Integrator

Flux integrators can be used to calculate the flow over surfaces in the simulation. Different types are available depending on how you want to define the surface.

You have two options for integrating the flux. The *discreet* method determines which connections between two cell centres cut through the surface. If such an intersection is found then the mass/energy flow given by that connection contributes to the total flow. The area of the surface represents the sum of all connection areas which contribute.

The *flux approximation* method determines the flow rate by numerically calculating the integral

$$w = \int \vec{F} \cdot d\vec{A} \tag{3.1}$$

where the flux vector $\vec{F}$ is *approximated* using the methods shown in section 4.2.4. The area integral is performed by subdividing the surface into refined triangles which are cut along the grid faces. Parts of the surface which lie outside of the grid do not contribute to the integral. Note that while the interpolation method for the flux is usually very reliable we have encountered some special cases where it misbehaves. If in doubt use the discreet method on a similar shape to check if the numbers are sensible.

You an invert the sign of the flow rate if the normal vectors as determined by the algorithm point in the wrong direction. Note when editing a shape the direction of the normal vectors can change and you may need to adjust the sign inverting selection.

3.8.3 Micro Gravity

Volsung can calculate the microgravity at defined station (benchmark) locations. The results can be displayed as differences in microgravity using a base time. You need to specify an elevation in the navigator tree at which the $2D$ plot is to be displayed; this elevation does not influence any calculations. If you have survey data you can also display it as contours on the $2D$ plot for comparison purposes.

Figure 3.11 shows the available options.

First you need to select the calculation method for the gravitational factors, i.e. the dependency of a station's gravity on a particular cell's geometry. The default is an *exact* method which should work for all kinds of polyhedron cells and is based on Singh and Guptasarma [2001]. For comparison purposes you can also use the *Point Mass* method which treats the blocks as point masses, i.e. like all of its mass would be concentrated at a block's centre.

The *recti-linear* method can only work on tartan type grids and is based on Chappell et al. [2012]. Its results should be the same as using the exact method but it can calculate the gravity factors about four times faster which may be of interest when dealing with large models.

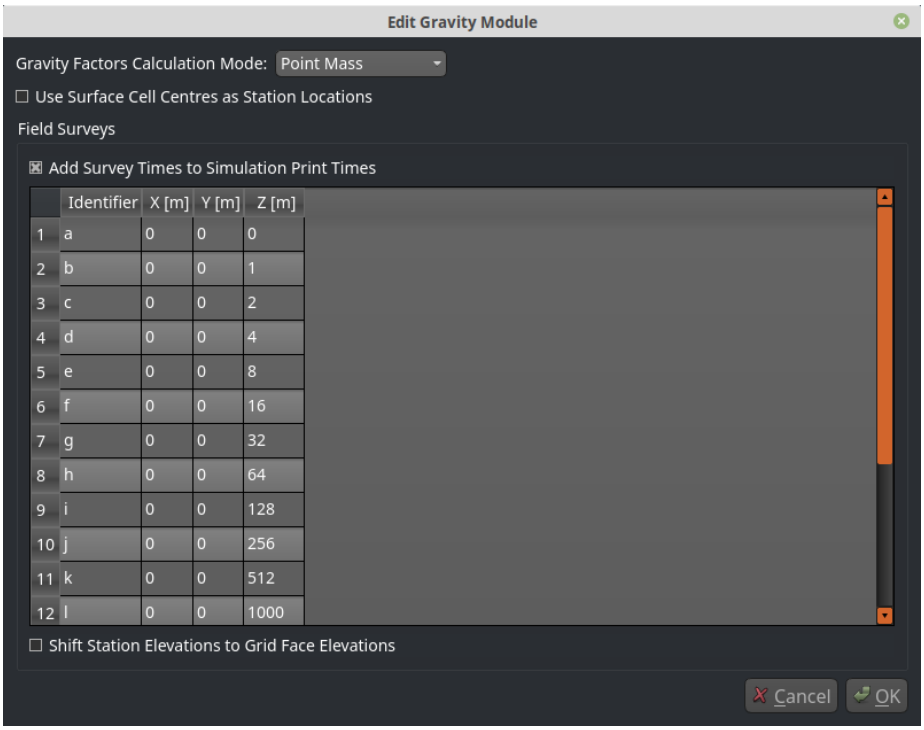

Figure 3.11: The dialog for editing micro gravity calculation options.

Next you will need to define station locations. You have the option to automatically generate stations on the surface face centres of the model grid. Additional stations and survey data can be added in a table; you have the option to shift the elevation of these stations to the face elevations of the grid. It is also a good idea to generate additional printout at exactly the same times for which survey data is present.

3.9 Reports

In *Brynhild* you have the option to generate some standard reports. These are generated off-line, i.e. not during the simulation. These reports can be used to calculate additional data and to print or export presentation style documents.

To generate a report right-click on the *Reports* item in the navigator. This will give you the option to generate a new report or to print/export a master document. This master document contains all reports in the order they are given in the navigator. You can right-click on individual reports to edit or print them or to change their order.

If you would like to generate a standard report which is currently not present in *Brynhild* please contact support so we can create this report for you.

3.9.1 Text Document

This report type can be used to generate title or section pages for your master report. It allows you to edit a rich-text document which can also contain images.

3.9.2 Number of Makeup Wells

This report shows the number of makeup wells as a chronological chart.

3.9.3 Reservoir Slice

This report allows to display a slice showing reservoir properties; an example is shown in figure 3.12. The reservoir is cut along an infinite plane; you need to specify an origin point and normal vector which define the plane. Note you can use the slice widget in the reservoir viewer to help you obtain the values for these; when you move the slice plane widget the viewer will show the origin and normal in the bottom left corner. You can also right-click onto the slice widget; it will give you a context menu which you can use to create new slices or update existing ones.

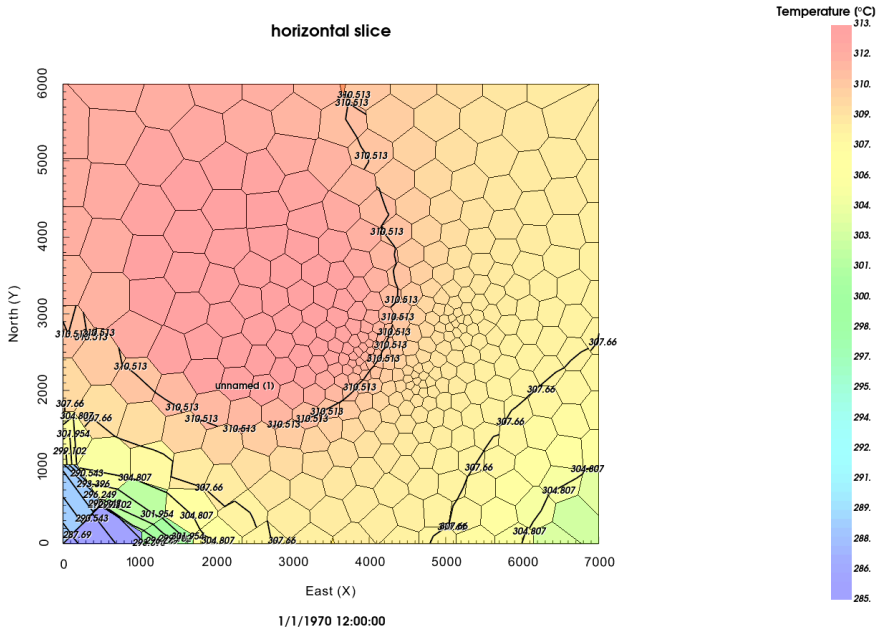

Figure 3.12: Example reservoir slice report showing cell colouring and contouring.

You will also need to specify a time at which the slice is performed. If no exact print time matching the desired time is found then the closest earlier print time will be used for the report.

You can display the cell properties on the slice and/or define contour lines. Use the opacity settings to put emphasis on one or the other.

Well tracks can be projected onto the slice. To prevent wells far away from the slice plane showing in the report you can use the near/far side threshold to block wells which are too far away from the plane. Note that you may need to lower the opacity for the cells to see wells which are on the far side of the slice plane.

Another option is to display intersection data with wells. If a well intersects the slice plane then field data is interpolated for the intersection point. To use this functionality enter the field data in the results tab of the well in the reservoir field data section. The temporal data closest to the print time for the report is gathered and interpolated along the welltrack to find the approximated value to plot. Note that for the interpolation along the welltrack the data end members are selected if the spatial data does not span over the intersection point.

Plotting well intersection data is in particular very useful for directly comparing temperatures in horizontal reservoir slices. Visually matching the well field data to the modelled reservoir temperature enables you to quickly see which parts of the model are colder/hotter than the real world data.

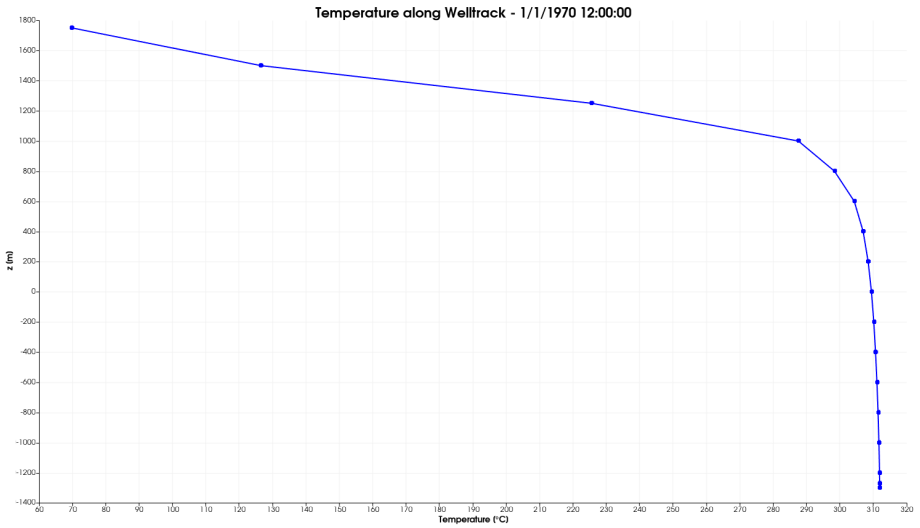

Figure 3.13: Example of a reservoir properties along welltrack; in this case the linearly interpolated temperature along a welltrack is plotted.

3.9.4 Reservoir along Welltrack

A common task in reservoir modelling is to compare reservoir data from a survey along a welltrack; the most commonly used type of this report are natural state temperatures.

This report has a list of wells and times, where the wells are objects defined in the surface network. For each well in the list the report will query the closest print time and query reservoir data along the welltrack. Field data (entered in the well's reservoir field data section on the results tab) and time filtered field data can be plotted for comparison. The time filtered data is useful when multiple field surveys are available and you only want to plot a survey which falls into the given filter interval.

The simulation data can be probed (i.e. using a cell lookup) or interpolated using either linear or Shepard kernels. For interpolation purposes you need to specify the number of neighbouring cell vertex points which the kernel will take into account. Figure 3.13 shows an example where the interpolated temperature along a welltrack has been plotted.

3.10 Running Brynhild from the Command Line

You can run *Brynhild* from the command line in order to perform automated tasks. You can use this feature to run models on either your local computer or on

a remote host. A typical example would look like:

```
$ brynhild model.brynhild --hide --run --verbose --remote Tex
```

The above example will start *Brynhild* without displaying any windows, run the model on the remote called "Tex" and will generate verbose output, i.e. display the log and other information while running. The application will block until the simulation has finished and all data has been transferred. If something goes wrong you may want to omit the `--hide` option to check where the problem occurs.

You can get a list of all command line options by using

```
$ brynhild myModel.brynhild --help
```

The Reservoir Simulator

The reservoir simulator *Fafnir* is the computational engine of the *Volsung* software package. It simulates multi-phase, multi-component flow through porous and fractured media using an integral finite difference method modelled on the very successful and popular *TOUGH2* code built by Pruess et al. [1999].

While *TOUGH2* is a de-facto industry standard in the geothermal industry it is also known for not being very user friendly and lacking key features which are important for geothermal operators. The focus in the development of *Volsung* has hence been to implement what works from *TOUGH2* and add in new features as required for making better use of reservoir models in geothermal operations. Development will continue and will be driven by user demand for particular features.

We will first start with a brief description of the equations of state used to calculate all thermophysical quantities required from a set of primary variables. Thereafter we will look at how *Fafnir* solves the basic differential equations to provide the solution for transient flow of mass and heat.

4.1 Equation of State

All thermophysical properties of the fluid required for the calculation of fluid flow are calculated within *Equation of State* (EOS) modules. Using a set of primary variables ($\vec{X}$) all secondary variables are calculated. The choice of $\vec{X}$ depends on the components of the system, however we will always need $N_p = N_c + 1$ equations per computational element. Here N_p is the number of primary variables per element and N_c is the number of fluid components, e.g. $N_c = 1$ for pure water or $N_c = 2$ if a second component like CO_2 is present.

Using a particular set of primary variables is beneficial for the computational speed of the simulator. However it can be hard for a user to compute them since quite often the choice of primary variables does not reflect common usage of variables. For example a system containing non-condensable gases uses the partial pressure

of the NCG as a primary variable. However the user usually wants to give NCG quantities using mass fractions and would need to go through tedious calculations to manually determine the partial pressure. *Volsung* avoids this issue by requiring the user to enter the fluid composition using the set of persistent variables of pressure, specific enthalpy and fluid mass fraction components, i.e. $(p, h, \bar{X})$. *Thermodynamic Tables* are used to calculate the required initial primary variables from the user input. You as a user hence will never need to do these calculations yourself.

While the thermodynamic tables always use $(p, h, \bar{X})$ as set of persistent variables in some cases you may want to enter values differently, say using $(p, T, \bar{X})$. State setter tools allow you to do this conversion and view all thermodynamic variables of the state you selected so you can be assured of setting the correct state. Note however that the setter tools make use of numeric routines and may reproduce your state with some small - usually insignificant - deviations. For example you may have set your state using a temperature of $T = 500K$ but the simulation results might show $T = 499.98K$.

Another advantage of using $(p, h, \bar{X})$ is that you can sometimes preserve some fluid components when changing the EOS. For example, if you are using an EOS with CO_2 as NCG and now want to switch to an EOS with CO_2 and $NaCl$ then your mass fractions of CO_2 will be preserved during the model conversion. However you will still need to recheck your states to ensure the enthalpy is correct since it is EOS specific.

4.1.1 Pure Water

The pure water EOS is based on the thermophysical properties of water as described in IAPWS-IF97 [2007] and related auxiliary and backward equation releases. Viscosity is calculated using IAPWS [2008] and the surface tension uses IAPWS-2014 [2014].

The thermodynamic table allows setting a state using (p, h), (T, x) or (p, x) where x is gas mass fraction for two-phase states.

4.1.2 Water + Carbon Dioxide

This EOS corrects the pure water thermophysical properties for the presence of CO_2. The total pressure is the sum of the water and CO_2 vapour pressures which are primary variables.

If liquid phase is present then the CO_2 liquid mass fraction is calculated using the Henry's constant given by Battistelli et al. [1997] as a function of temperature for $T \leq 659.7815787K$. Above $T = 659.7815787K$ the Henry constant is assumed to be constant. This is a numerical trick since the Battistelli et al. [1997] formulation

would turn negative above this value; using this trick we can continue to use this EOS for high temperatures though it needs to be said that there is little literature data around to indicated how the system is best described.

The heat of solution is calculated using differentiation of the Henry's constant using the equations given by Battistelli et al. [1997]. Viscosity and density of the liquid phase is assumed to be not affected by the presence of CO_2.

The gas mass fraction of CO_2 is calculated by assuming the mixture to be an ideal gas and using the gas specific density of CO_2 from Pritchett et al. [1981]. Viscosity of CO_2 is taken from the same publication; the gas phase uses mass-weighted viscosity. The enthalpy of CO_2 gas is given by Sutton and McNabb [1977].

4.1.3 Water + Air

This EOS uses the same partial pressure approach as described in the section for water and CO_2 (4.1.2). Specific density and enthalpy of air are calculated using an ideal gas approximation. The gas viscosity is taken from Lemmon and Jacobsen [2004]. Henry's constant and heat of solution from its differentiation are the same as in the *TOUGH2* manual [Pruess et al., 1999].

4.1.4 Water + Sodium Chloride

This EOS can handle up to three phases, with halite forming the solid phase. Its thermodynamic formulations are built on the Battistelli et al. [1997] and Battistelli [2012] publications. However some errors found in the thermodynamic functions they cite have been fixed. Also the pure water properties from which the brine properties are calculated employ the IAPWS-IF97 [2007] formulations.

Restrictions on this EOS are that temperatures are below $350°C$ and elements can not attain 1.0 solid saturation. Also note that while Battistelli et al. [1997] claim the equations to be valid up to halite saturation we have found that the brine compressibility of Andersen et al. [1992] on which the brine density and enthalpy are founded have a pole at around $X_{NaCl,brine} = 0.37$. In fact Andersen et al. [1992] claim their formulation only to be valid to about $X_{NaCl,brine} = 0.3$. Hence we don't recommend the use of this EOS for $X_{NaCl,brine} > 0.3$.

The corresponding thermodynamic table can use $(p, T, \bar{x})$ only if gas and liquid phase are not present at the same time; you can use $(T, x, \bar{x})$ in this case instead, with x being the gas mass fraction. The $(p, h, \bar{x})$ state setter can handle all phase states.

4.1.5 Water + Sodium Chloride + Carbon Dioxide

This EOS can handle three phases and has as components sodium chloride and carbon dioxide. It is built on the logic described by Battistelli et al. [1997] and Battistelli [2012] and uses the basic thermodynamic property functions given in sections 4.1.2 and 4.1.4.

4.1.6 Water + Sodium Chloride + Air

This EOS can handle three phases and has as components sodium chloride and air. It is built on the logic described by Battistelli et al. [1997] and Battistelli [2012] and uses the basic thermodynamic property functions given in sections 4.1.3 and 4.1.4.

4.2 Mass and Energy Balances

Conservation of mass and energy is the fundamental principle on which *Fafnir* is built. Pruess et al. [1999] show the derivation of the basic conservation equations in more detail; we will give a brief recap here and focus on their discretized form.

4.2.1 Accumulation

The general form of the mass accumulation term is

$$M^k = \phi \cdot \sum_\beta S_\beta \cdot \rho_\beta \cdot X^k_\beta \tag{4.1}$$

where k denotes the component (e.g. H_2O, CO_2) and β denotes the phase (e.g. liquid, gas). The phase parameters S, ρ and X^k_β denote the phase saturation, phase density and phase component mass fraction of k. M^k describes the amount of mass per unit volume, considering that the mass is only present within pore space ϕ of the volume under consideration. Note that k starts with index 0 within *Fafnir* since the *Volsung* package is built on C++ which prefers zero-based indexing instead of one-based indexing used in Fortran on which *TOUGH2* was built. So if N_c mass components are present then the index will run from 0 to $N_c - 1$.

We can define a similar heat accumulation term as

$$M^{N_c} = (1 - \phi) \cdot C_R \cdot \rho_R \cdot T + \phi \cdot \sum_\beta S_\beta \cdot \rho_\beta \cdot u_\beta \tag{4.2}$$

The first part of the right hand side of the equation describes the amount of heat stored in the rock portion of the computational element. It makes use of the rock thermal heat capacity C_R, the rock density ρ_R and the absolute temperature T.

The second part of the equation describes the heat stored in the fluid within the pore space using the specific internal heat u_β of the corresponding phase.

We can now combine the mass and heat accumulation terms into a single vector M:

$$M = (M^0, M^1, ..., M^{N_c-1}, M^{N_c}) \tag{4.3}$$

So far we have only looked at a single computational element. However our system containts N elements. Hence we introduce the subscript index i and the total accumulation vector $\vec{M}$:

$$\begin{aligned} \vec{M} &= (M_0, M_1, ..., M_{N-1}) && (4.4) \\ &= (M_i) && (4.5) \end{aligned}$$

Conservation of mass and energy dictates that mass and heat within the system under consideration must be conserved. Hence the change of mass/heat within one computational element must be balanced by either sources or sinks attached to the element or must be due to exchange of mass/heat with its neighbouring elements. We can write this as the differential equation

$$\frac{dM_i^k}{dt} = V_i^{-1} \cdot \left(\sum_j A_{ij} F_{ij}^k + Q_i^k \right) \tag{4.6}$$

with V_i being the volume of element i. A_{ij} is the interface area between element i and element j; the summation goes over all neighbours j to the element i. F_{ij}^k is the component flux between the elements and is symmetric, i.e. $F_{ij}^k = -F_{ji}^k$.

Q_i^k is the component source or sink rate.

4.2.2 Sources and Sinks

Sources and sinks appear in many forms within a reservoir simulation. Quite often they are specified by the user, either in a table form or from more complex computations like wellbore simulations.

The common form for the user to specify a flow rate is in the form $(w, q, h, \bar{X})$, i.e. using mass flow rate w, dry heat rate q, specific enthalpy h and component

mass fractions $\bar{X}$. They relate back to the component source/sink rate; for the mass rates this is

$$Q_i^k = \pm w \cdot X^k \tag{4.7}$$

and for the heat rate it is

$$Q_i^{N_c} = \pm w \cdot h \pm q \tag{4.8}$$

Note that the $\pm$ here indicates if the mass/heat is injected into the element $(+ve)$ or extracted from the element $(-ve)$.

Sources to an element are easy to deal with; you will need to fully specify all quantities in $(w, q, h, \bar{X})$.

However sinks are trickier since the flowing enthalpy and component mass fractions depend on the current composition of the fluid. You will therefore only need to specify w and q; the remaining quantities h and $\bar{X}$ are calculated internally using the concept of phase mobilities:

$$\Upsilon_\beta = \frac{k_{r\beta} \cdot \rho_\beta}{\mu_\beta} \tag{4.9}$$

where $k_{r\beta}$ is the phase relative permeability and μ_β the phase viscosity. The total mobility is

$$\Upsilon = \sum_\beta \Upsilon_\beta \tag{4.10}$$

Phase and total mobilities are then used to calculate the flowing quantities:

$$h = \Upsilon^{-1} \cdot \sum_\beta \Upsilon_\beta \cdot h_\beta \tag{4.11}$$

$$X^k = \Upsilon^{-1} \cdot \sum_\beta \Upsilon_\beta \cdot X_\beta^k \tag{4.12}$$

4.2.3 Fluxes

Fafnir currently only includes linear Darcy flow for the calculation of fluxes between two compuational elements. Molecular diffusion and hydrodynamic dispersion are currently not part of the code but may be added later on.

In its discretized form the phase Darcy flux between two elements is given by

$$F_{ij,\beta} = -k_{ij} \cdot \Upsilon_{ij,\beta} \cdot \left(\frac{P_{i,\beta} - P_{j,\beta}}{D_{ij}} - \rho_{ij,\beta} \cdot g_{ij} \right) \tag{4.13}$$

D_{ij} is the nodal distance between the two element centres.

$P_{i,\beta}$ is the phase pressure of element i. It differs from the system pressure p in i by taking the capillary pressure into account, i.e.

$$P_{i,\beta} = p + P_{cap,\beta} \tag{4.14}$$

g_{ij} is the gravity acting between the two elements, i.e. $g_{ij} = \cos\alpha \cdot g$ where g is the gravitational acceleration ($9.80665 m/s^s$) and α the angle to the vertical between the two element centres.

k_{ij}, $\Upsilon_{ij,\beta}$ and $\rho_{ij,\beta}$ represent the total permeability, phase mobility (see equation 4.9) and phase density, respectively. However one encounters the difficulty that these quantities are different in the two elements i and j. We hence need to use their *weighted* quantities between the two elements. We can use the following weighting schemes:

- *Average* weighting - both elements bear the same weight.
- *Upstream* weighting - quantities are taken from the upstream element only.
- *Harmonic* weighting - elements are weighted using their distance between their cell centre and the interface location.

You can select the weighting to be used from within the *Brynhild* GUI.

Note that upstream weighting can not be used for determining the phase density term since the pressure gradient (the term in the paranthesis in equation 4.13) itself depends on its value. An additional weighting scheme we can use for determining the phase density in the the pressure gradient is *Saturation* based weighting. For single phase cases this equals the same as average weighting; however it has very nice smoothing properties when phases appear/disappear and can aid numerical stability significantly. We recommend it as the default weighting scheme for this use case.

In case where a phase is absent in one element the density and viscosity will be taken from the other element. If the phase is absent from both elements then the phase flux will be zero.

After calculating the phase Darcy flux between two elements one can use it to calculate the phase mass component flux

$$F^k_{ij,\beta} = F_{ij,\beta} \cdot X^k \tag{4.15}$$

the total component flux

$$F^k_{ij} = \sum_\beta F^k_{ij,\beta} \tag{4.16}$$

and finally the total flux

$$F_{ij} = \sum_k F^k_{ij} \tag{4.17}$$

The heat flux between two elements in its discretized form is

$$F^{N_c}_{ij} = \sum_\beta h_\beta F_{ij,\beta} - \lambda \frac{T_i - T_j}{D_{ij}} - \epsilon \cdot \sigma_{SB}(T_i^4 - T_j^4) \tag{4.18}$$

The first term describes heat convection due to mass flow between the two elements.

The second term describes heat conduction; λ is the heat conductivity and always uses the harmonic weight between the two elements. However you can use dry and wet heat conductivity and decide how the wet conductivity is to be calculated from the saturation.

For the linear heat conductivity function we have

$$\lambda_i = \lambda_{i,dry} + S_{liq} \cdot (\lambda_{i,wet} - \lambda_{i,dry}) \tag{4.19}$$

and for a square-root function we have

$$\lambda_i = \lambda_{i,dry} + \sqrt{S_{liq}} \cdot (\lambda_{i,wet} - \lambda_{i,dry}) \tag{4.20}$$

The third term in equation 4.18 refers to radiative transfer, with radiative emittance ϵ and the Stefan-Boltzmann constant σ_{SB}. For the reservoir itself this term is always automatically set to zero by using $\epsilon = 0$. However when coupling to an atmosphere one can chose a different value for ϵ; a value of $\epsilon = 0.612$ should be a typical value for radiative transfer between ground and the free atmosphere[1].

[1] see `https://en.wikipedia.org/wiki/Climate_model`

Permeability Anisotropy

In order to accomodate anisotropy the permeability is defined as a tensor [Liakopoulos, 1965]. We can write the flux vector across a block interface as

$$\vec{f} = \Upsilon \cdot K \cdot \vec{\nabla} p \tag{4.21}$$

where K is a tensor of second rank and can be expressed as a symmetric, positive definite 3x3 matrix. In the GUI it can be entered via specifying the three principal components of the tensor and providing up to three rotation angles:

$$K = R_{z-} \cdot R_{x+} \cdot R_{z-} \cdot \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \cdot R^t_{z-} \cdot R^t_{x+} \cdot R^t_{z-} \tag{4.22}$$

Here the rotation matrices R have a subscript which denotes the rotation axis and rotation direction using the basic right hand rule. Usually only the innermost R_{z-} matrix is necessary and the azimuth angle is used to orientate the permeability in the $x - y$ plane.

The gradient vector $\vec{\nabla} p$ can be expressed as the sum of two vectors; one in the direction of the surface normal $\vec{n}$ and one perpendiular to it:

$$\vec{\nabla} p = \vec{\nabla} p_{\parallel} + \vec{\nabla} p_{\perp} \tag{4.23}$$

Since *Fafnir* makes a two-point approximation for determining the pressure gradient we can not evaluate the perpendicular term and hence neglect it. This means that some tensor flow effects can not be modelled in *Fafnir* but in practical terms this simplification is usually justified. Chen et al. [2008] give a good overview over this problem.

The parallel part can be expressed by mutliplying a scalar $\tilde{p}$ on to the normal vector; $\tilde{p}$ is given in equation 4.13 as the term in the parenthesis.

$$\vec{\nabla} p_{\parallel} = \tilde{p} \cdot \vec{n} \tag{4.24}$$

We now evaluate the total mass flow w across the block interface:

$$\begin{aligned} w &= A \cdot \langle \vec{n}, \vec{f} \rangle && (4.25) \\ &= A \cdot \langle \vec{n}, K \vec{\nabla} p \rangle && (4.26) \\ &\approx A \cdot \langle \vec{n}, K \vec{\nabla} p_{\parallel} \rangle && (4.27) \\ &\approx A \cdot \langle \vec{n}, K \vec{n} \rangle \cdot \tilde{p} && (4.28) \end{aligned}$$

Hence the scalar permeability k used in equation 4.13 is evaluated for each individual interface area between blocks as

$$k = \langle \vec{n}, K\vec{n} \rangle \tag{4.29}$$

4.2.4 Cell Connectivity

We distinguish between two basic kinds of connections: *Internal* connections are made between the internal MINC layers of a cell (see section 5.2 for more information on MINC). *External* or inter-cell connections connect the same internal layers of two neighbouring cells with each other, i.e. fracture to fracture, matrix layer 1 to matrix layer 1 and so on. External connections are always made between the same layer indices, i.e. the fracture (layer 0) is never connected to another cell's matrix 1 layer etc.

The *dual porosity* model only connects the fractures (layer 0) of two neighbouring cells and uses the full area in equation 4.28. Similarly the *dual permeability* model connects both the fracture (layer 0) and first matrix layer (layer 1) between two neighbouring cells, again using the full interface area.

In order to allow full flexibility for cell connectivity you can set the following parameters:

The *area weight* determines which area should be taken for calculating the mass flow between two elements. Full uses a weight of 1.0, volume fraction uses the average volume fraction of the two elements.

The *cell connectivity* can be given individually for layers 0, 1 and all other remaining layers. You can select to connect all internal layers, none, or only elements for which the centers are horizontal or vertical to each other. The distinction of horizontal versus vertical is made by using a 45° criterion between the connection vector and the vertical axis.

3D Fluxes and Velocities

The fluxes above are defined at the face centre locations and are only given as a flux normal to the face. For visualization purposes as well as more detailed streamline analysis we are interested in the $3D$ vector field for the mass flux and the phase velocities. We use a linear approximation for the fracture flux vector inside a cell [2]:

[2] method developed by Adrian Croucher, personal communication

$$\vec{f}(\vec{x}) = \vec{f^0} + \cdot \begin{bmatrix} \delta_x & 0 & 0 \\ 0 & \delta_y & 0 \\ 0 & 0 & \delta_z \end{bmatrix} \cdot (\vec{x} - \vec{c}) \tag{4.30}$$

where $\vec{f^0}$ is a constant and $\vec{c}$ is the cell centre. The flow through the $i-th$ face of the cell can be approximated by

$$w^i \approx A^i \cdot (\vec{f^0} + D \cdot (\vec{x^i} - \vec{c})) \cdot \vec{n^i} \tag{4.31}$$

where A^i is the face area, $\vec{x^i}$ the face centre location, $\vec{n^i}$ the face normal pointing in outward direction and D is the matrix from equation 4.30. We can use equation 4.31 to set up N linear equations with six unknown for each of the cell's N faces. Note that internal fracture/matrix flows are ignored for the estimation of the fracture flux vector since they are interpreted as internal sources/sinks to the flux field.

We now set up a linear system

$$M \cdot (f_x^0, f_y^0, f_z^0, \delta_x^0, \delta_y^0, \delta_z^0)^t = (w^0, w^1, ..., w^{N-1})^t \tag{4.32}$$

where the matrix coefficients for M are

$$m_{ij} = A^i \cdot n_j^i \qquad \forall j < 3 \tag{4.33}$$

$$m_{ij} = A^i \cdot (x_j^i - c_j) \cdot n_j^i \qquad \forall j \geq 3 \tag{4.34}$$

Depending on the number of faces and face orientations this system is potentially under- or over-determined. The over-determined case can be solved using a least-square fit while the under-determined case can be solved by using the minimum norm solution of the solution subspace. The Moore-Penrose pseudo-inverse $M^\dagger$ can be used for all these cases and we calculate the unknowns as

$$(f_x^0, f_y^0, f_z^0, \delta_x^0, \delta_y^0, \delta_z^0)^t \approx M^\dagger \cdot (w^0, w^1, ..., w^{N-1})^t \tag{4.35}$$

and hence the fracture flux vector for the cell using equation 4.30.

After the calculation of the fracture flux vector we can use it to calculate the fracture phase velocities:

$$\vec{v_\beta}(\vec{x}) = \frac{\Upsilon_\beta}{\Upsilon \cdot \rho_\beta \cdot S_\beta \cdot \phi \cdot V/V_{cell}} \cdot \vec{f}(\vec{x}) \tag{4.36}$$

Note that the V/V_{cell} term is used to correct the fracture phase for the fact that the fracture occupies only a fraction of the total cell volume.

4.2.5 Numerical Solution

Starting from an initial state $M_i(t)$ (see 4.2.7) we are interested to find the numerical solution for system at $t = t + \Delta t$. For each computational element i we introduce its residual vector R_i as

$$R_i^k(t+\Delta t) = M_i(t+\Delta t) - M_i(t) - \frac{\Delta t}{V_i}\left(\sum_j A_{ij}F_{ij}^k + Q_i^k\right) \tag{4.37}$$

Conservation of mass and energy demands that we find a solution so that

$$R_i^k(t+\Delta t) \quad = \quad 0 \quad \forall i,k \tag{4.38}$$

Analogous to Equations 4.4 and 4.4 we use vector notation $\vec{R}$ to indicate the set of all residual for all elements. We can write $\vec{R}$ as a function of a set of unknown primary variables $\vec{X}$. Then we expand the residuals into a Taylor series around a point $\vec{X_p}$ and keep only the first linear term:

$$\vec{R}(X_{p+1}) \approx \vec{R}(\vec{X_p}) + J \cdot (\vec{X}_{p+1} - \vec{X_p}) \tag{4.39}$$

where J is the Jacobian matrix

$$J_{rc} = \frac{\delta R_r}{\delta X_c} \tag{4.40}$$

and is evaluated at X_p.

The Jacobian is built inside *Fafnir* using numerical differentiation. We then use Newton's method to find an iterative solution by solving the linear system

$$J \cdot (-\delta\vec{X}) \quad = \quad \vec{R}(\vec{X_p}) \tag{4.41}$$

$$\vec{X}_{p+1} \quad = \quad \vec{X_p} + \delta\vec{X} \tag{4.42}$$

The iterative solution will be accepted once the residual becomes sufficiently small for each element and component. The basic acceptance criterion is

$$\frac{|\tilde{M}^k_{i,p+1}(t+\Delta t)|}{|M^k_i(t)|} \leq \epsilon_1 \tag{4.43}$$

where ϵ_1 is the *relative* convergence criterion (typically 10^{-5}). We use

$$\tilde{M}^k_{i,p+1}(t+\Delta t) = max(|R^k_{i,p+1}(t+\Delta t)|, |\grave{M}^k_{i,p+1}(t+\Delta t)|) \tag{4.44}$$

$$\grave{M}^k_{i,p+1}(t+\Delta t)) = max(\epsilon_2, \epsilon_3 \cdot max(|M^k_{p+1}(t+\Delta t)|)) \tag{4.45}$$

In here ϵ_2 is the *absolute* convergence criterion (typically 1.0); *TOUGH2* makes use of a similar convergence criterion. A third convergence criterion has been introduced here; ϵ_3 links each component with to the maximum accumulation found for this component throughout the entire system and is called the *absolute maximum* criterion. In effect this determines what a "large" amount for this component would be and sets the absolute convergence criterion accordingly. A typical value for ϵ_3 is 10^{-4}; however one can choose to ignore this criterion altogether by setting it to zero in which case only ϵ_2 is used to determine the absolute criterion.

4.2.6 Solving the Linear System

The linear system described in equation 4.41 is solved using a sparse iterative solver. Currently *Fafnir* supports three different *solver types*, BiCGStab [van der Vorst, 1992], GMRES(m) [Saad and Schultz, 1986] and AAR[3] [Suryanarayana et al., 2019]. ILU0 is used as the internal preconditioner for all solver types.

The system is solved iteratively until it is considered to be converged using the *relative* and *absolute* convergence criteria or until the *maximum number of internal iterations* has been exceeded. The relative *Ginsburg* factor is used in combination with the maximum number of iteration parameter to determine when the residual in the iterative solver is to be replaced by the actual residual using the current solution vector.

The maximum number of iterations for the linear solver is given by the *relative maximum iteration* parameter which finds an integer number by multiplying the number of unknowns in the system by the relative factor. Some solvers will adjust this parameter to suit their internal algorithms. Good values for the relative maximum iterations parameter are usually 0.05 to 0.1. If this parameter is chosen too high the solver may end up using a lot of internal iterations but can't improve the solution any further, so sometimes choosing a smaller value can improve performance. But there is no good general rule except for watching the simulation log and checking if the solver struggles to find a solution and is running out of iterations, in which case one may want to increase the parameter value.

[3] currently only working correctly under *Linux*

We have found that the basic BiCGStab solver is usually the solver of choice - it is fast and consumes very little memory. GMRES(m) can be more robust but has a quadratic term which grows with m and hence tends to be slower than BiCGStab.

Sometimes - in particular at large time steps - the linear solver can struggle due to limited numerical precision. You can then opt to select a backup solver; if the primary solver fails to reduce the residual to the desired criteria then the backup solver can start using the best guess solution from the primary solver and aim to reduce the residual further. A good combination may be to use BiCGStab as primary solver and GMRES(m) as a backup solver.

The linear solver can run on different backends to perform the actual arithmetic calculations. The basic choices are:

- `CPU` uses the basic CPU backend. It is well suited for very small models or if you don't have a computer with a GPU.
- The `GPU` solver uses an *NVIDIA* graphics card if your computer is equipped with one of these. GPU solvers are much faster than CPU solvers so you should check if you can employ one of these for *large* models - note that on *small* models the CPU solver may be faster. See appendix A for more information on GPUs.
- `CPULC` is a backend for using CPUs with large cache, like the *AMD Ryzen* or *AMD Threadripper* series.

When creating a new model you should try out the different backends to see which one works best for your model. You can do this by running the model for a set number of time steps and then compare the linear solve time in the profiling information at the end of the simulation log.

4.2.7 Initial State

Fafnir requires an initial state for the reservoir in order to start the simulation. By default the initial state is determined by the thermodynamic states defined in the lithological units and the boundary conditions.

Fafnir will periodically write a *SAVE* file using the *interval* specified by the user. This file contains the primary variables and the porosity at the end of a time step.

You can opt to reload a SAVE file at the beginning of the simulation, for example you may want to reload a natural state before running through a calibration or scenario - note you should rename this file first, say to `INCON.fafnir`. When reloading a SAVE file it is a requirement that the grid has not changed. However it is possible to change the MINC layers of individual blocks; in this case *Fafnir* will try to look up the initial state from the next outer layer if an exact layer match is not found.

If you have the python tools installed (see appendix **??**) you can also use the `sigurdcmd.py` script to generate an `INCON.fafnir` file from any print times in the `Results.sigurd` output file.

Also note that boundary condition initial states from the model are applied *after* the initial state file has been loaded. This means you can alter boundary conditions like fixed states between model runs. However you can also switch this behaviour off for individual fixed state boundary conditions within the model so that whatever state they have currently loaded will be preserved.

4.2.8 Steady State Detection

Fafnir has the option to detect a system state close to steady state when running through the natural state period. This feature allows you to set stability criteria; if all of these criteria are fulfilled then the state is considered close to steady-state and the time is skipped towards calibration or scenario period.

The criteria are entered on a per-time basis:

- the pressure criterion; the default value is $3 \cdot 10^{-6}$ which is about $0.1 bar$ over 100 years
- the temperature criterion; the default value is $3 \cdot 10^{-11}$ which is about $0.1 K$ over 100 years
- the saturation criterion; the default value is $3 \cdot 10^{-14}$ which is about 0.01% over 100 years
- the mass fraction criterion; the default value is $3 \cdot 10^{-14}$ which is about 0.01% over 100 years

The state is considered stable when all above criteria have been met over three consecutive time steps.

Note that the steady state detection is meant to be an auxiliary feature to help you when your simulation has problems reaching a natural state due to numerical precision problems. You should first try to reach a stable solution without using this feature since it can leave artefacts in your simulation, like regions which have not fully warmed up yet.

4.2.9 Simulator Settings

Use of a GPU as an Accelerator

If your computer is equipped with a *NVIDIA* GPU you can opt to use it as an accelerator for the linear solve (see 4.2.6) which - particularly in natural state runs - is often the most computationally expensive part of a simulation.

It will depend on your model size, the number of components used in the model and on your GPU and CPU specifications if it makes sense to employ a GPU for the task. Due to the numerical overhead when parallelizing the linear solve and transferring the data to the GPU's memory you will find that for small models it will be faster to perform the solve on the traditional CPU instead. It is easy to check which one you should use:

- Run your model using the GPU for about 20 time steps and note down the "Solving Linear System" time which is printed out in the profiling information at the end of the simulation.
- Re-run the model with the GPU disabled for the same number of time steps. Compare the time for the linear solve to the time used when employing the GPU and hence decide if it makes sense to use the GPU.

For more information what type of GPU to select see appendix A.

Logging

The amount of text output during a simulation is control via the log level setting. The higher this setting the more output will be generated; this can be useful when trying to figure out why a model has problems with convergence etc.

Fafnir allows the simulation log to be accessible through a TCP/IP connection. This is particularly useful when working with remotes; however you can also select this option when running on your local machine. You will need to remember to open your firewall to allow outside access via the port you have specified.

Backtracking

A simple backtracking algorithm can be used in *Fafnir* to avoid excessive flip-flopping behaviour in the Newton-method. When solving the linear system 4.41 it will be tested that going the full Newton-step (i.e. the solution of the system) will actually improve the overall residual. If the residual is getting larger when going the full Newton step then smaller steps are tried out; however the algorithm will abort when the step size is getting too small. Backtracking may result in better convergence in some problems but can lead to longer runtimes in others, so it may need to be decided on a case-by-case basis if it is worthwhile using it.

Switching MINC off in Natural State Period

You can opt to switch the internal MINC layers off during the natural state period. This can result in much faster natural state run times since state changes in the individual cells are no longer buffered. It is highly recommended to use this switching method; quite often a simulation fails to significantly progress during the natural state period due to the matrix buffering.

You have the following methods for handling matrix layers during natural state runs:

- **Matrix On:** In this mode the matrix layers are fully active.
- **Fracture Only:** In this mode all matrix layers are disabled, only the fracture system is modelled. At the end of the natural state period the matrix layers are switched back on an are set to the same state as the fracture. This method does not work correctly if the elements of a block have different capillary pressure functions.
- **Matrix Shell:** This is the recommended method if the elements in a block have different capillary pressures. In this mode the the natural state simulation time is split into equal parts so that each MINC layer has time to equilibrate individually. In the first stage the fracture system is modelled, while all matrix layers are disabled. In the successive stages one matrix layer is enabled while the next outer layer is set to fixed state. This results in a stable system since each layer has time to fully equilibrate with the next outer layer. Note if you use this mode you need to enlarge the natural state period by a factor equal the number of MINC layers. For example, if you want to reach a steady state over 1 million years and you have 1 fracture and 4 matrix layers you need to set the simulation time to cover 5 million years.

When using the more advanced modes you should also ensure that the model has constant conditions over the natural state period, i.e. no changing flow rates or element states.

You should also not use the advanced modes if you have inter-block connections of matrix layers since it is assumed that all transport of fluid and heat is internal to a block only.

Translator

Fafnir has a built-in option where it can try to output a model in another simulator's format. This is a very rudimentary feature and will only work on a limited number of very simple models.

Currently a translation to *TOUGH2* models is implemented. You can use this translator when you have:

- Only simple wells with fixed-fraction feedzones which only use tables (i.e. not using the surface network feedback).
- Your permeability tensors are aligned with the grid orientation.

There can be no guarantee that the translated model truly reflects the *Fafnir* model so use this feature carefully.

TOUGH2 Input/Output Files

When running in *TOUGH2* compatibility mode you will have the option to specify the *TOUGH2* input file. You will also have the option to select a listing file to which output from *Fafnir* is written similar to *TOUGH2* output; however note that the listing file will use the units you have selected in *Brynhild* instead of the default *TOUGH2* units. The listing file can be read with *PyTOUGH*.

4.3 Time Step Control

The time step control allows you to specify time related parameters for the simulation.

Volsung uses the ISO 8601 extended format, i.e. times are represented as "yyyy-MM-ddTHH:mm:ss.zzz" where "zzz" stands for milliseconds.

Internally all times are stored as seconds and the offset is 01/01/1970 (UTC). You can enter times either directly as seconds, e.g. "-1e+15" is a valid entry. You can also enter dates in the normal "dd/MM/yyyy" fashion or using the ISO 8601 format.

The *start* and *end* times denote the total time period over which the simulation is supposed to run. The simulation will stop if the *maximum number of steps* is exceeded.

The *start time step* is the initial step in the simulation. When the number of Newton iterations in a time step is less than or equal to *fast convergence* then the time step will be expanded using the *expansion factor*. If a Newton iteration fails then the time step will be adapted using the *reduction factor*. The *maximum* and *minimum time step* provide the limits for the time steps during the natural state period. Note that a lower time step limit of $1ms$ is imposed since this is the smallest difference that the ISO 8601 time format can handle.

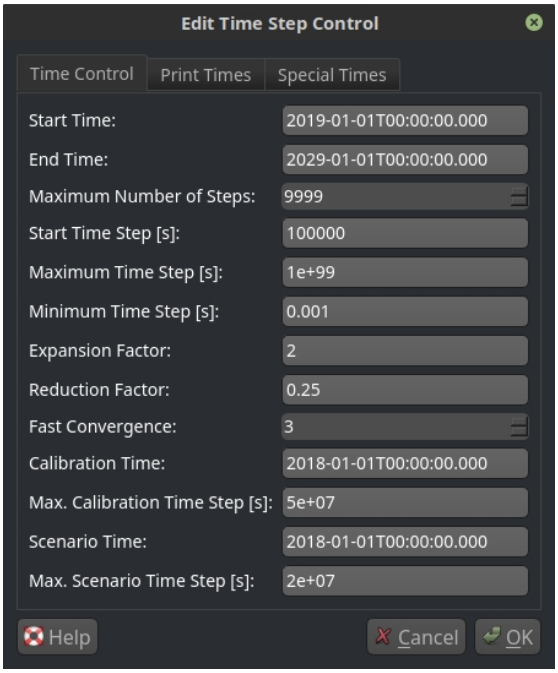

Figure 4.1: The Time Step Control Dialog

You can specify the beginning of the *calibration time* and the *maximum calibration time step* during this period. In the same fashion you can specify the *scenario time* and *maximum scenario time step*.

Figure 4.2 shows the output control settings. You can opt to *print the initial state*. The *print interval* can be used to periodically generate output. However since time steps usually vary it is often better to specify a list of *print times* directly.

It can be very useful to generate an additional output state when a *warning* or *error* is present from the surface network, e.g. when a well can't produce the amount of fluid targeted. Always remember that output is only indicative for the current time and that the system can be in quite a different flow state between print times. Hence if a warning or error is encountered you may want to rerun the simulation with additional output times specified around this time.

Finally you can enter *special times* as shown in figure 4.3. These are times you want to force the simulation adhere to because something special is happening but you don't need printout at this time. Internally *Fafnir* will add a lot of times automatically to this list, like surface network object commission dates etc.

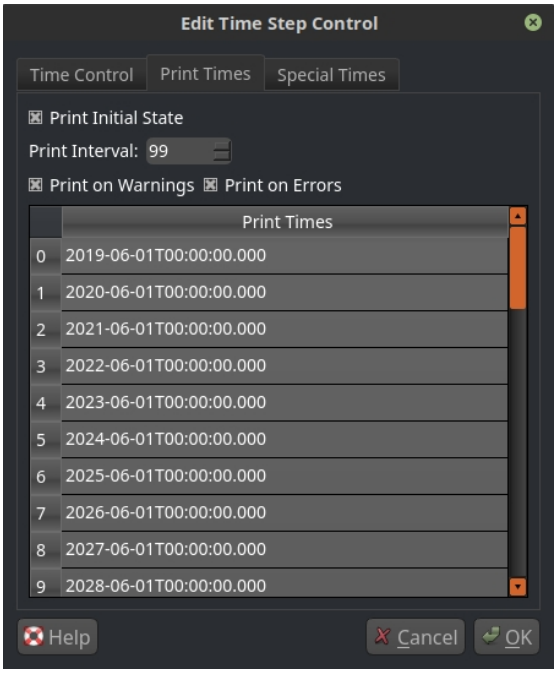

Figure 4.2: Print Times in the Time Step Control Dialog

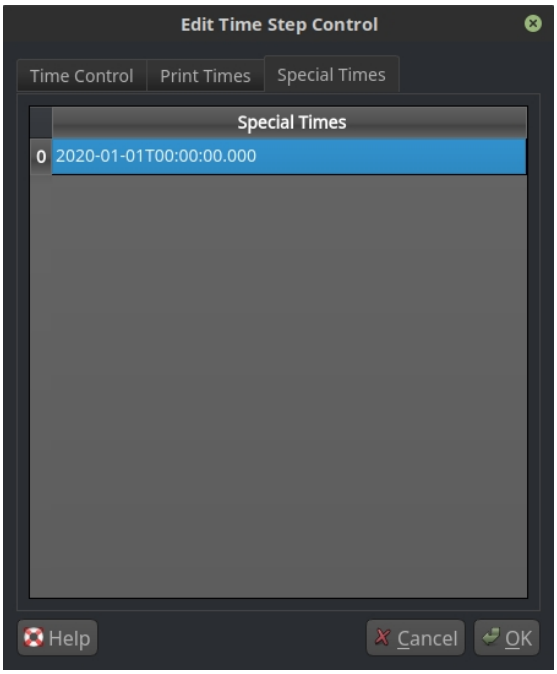

Figure 4.3: Special Times in the Time Step Control Dialog

4.4 Boundary Conditions

Boundary conditions can be set in *Fafnir* independently from the grid; the software will automatically determine which cells are affected by the boundary condition and if necessary allocate source/sink flow rates according to the interaction between the specified boundary condition and a cell.

Fafnir currently supports three boundary condition types:

- *Extra Blocks* (see 4.4.1) are boundary conditions where one ore more extra computational element is added to the simulation. This element is located outside of the visible grid but attaches to certain elements.
- *Fixed State* (see 4.4.2) boundary conditions maintain elements selected by the interaction at fixed thermodynamic conditions. Note that fixed state boundary conditions you create will override the base element state of the grid blocks as they were defined by the lithology unit (see 5.3 for details).
- *Sources/Sinks* (see 4.4.3) are boundary conditions which either inject fluid of known composition or extract fluid using the flowing composition (see 4.2.2 for details).

4.4.1 Extra Blocks

Atmosphere

The atmosphere boundary condition will set up a single computational element using the fixed state conditions which you can select in the atmosphere dialog (see figure 4.4).

The atmosphere element will be connected to all cells with exposed top surfaces in the grid. The connection distances are the distance between the cell centre and the top cell surface one one side and zero on the other side. The permeability for each individual connection is taken from the fracture layer of the cell.

You can set an emissivity for radiative heat transfer; see equation 4.18 for details.

Hot Plate

The hotplate boundary condition will set up a single computational element using the fixed state conditions which you can select in the hot plate dialog (see figure 4.5).

The hot plate attaches to the bottom surfaces of the cells in the grid. The connection distance is the distance from the cell centre to the bottom surface

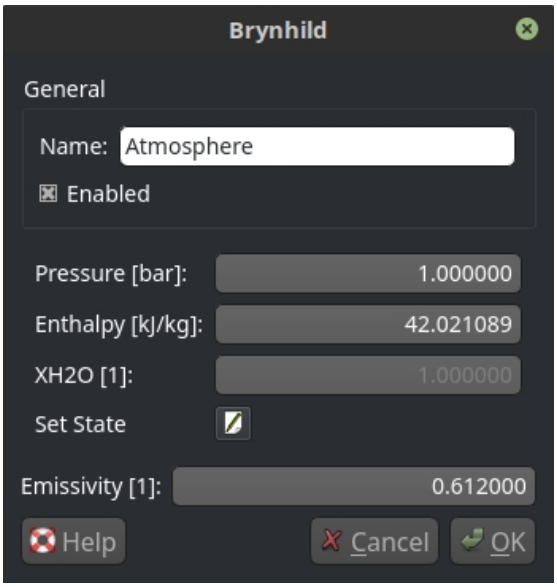

Figure 4.4: The Atmosphere Parameters

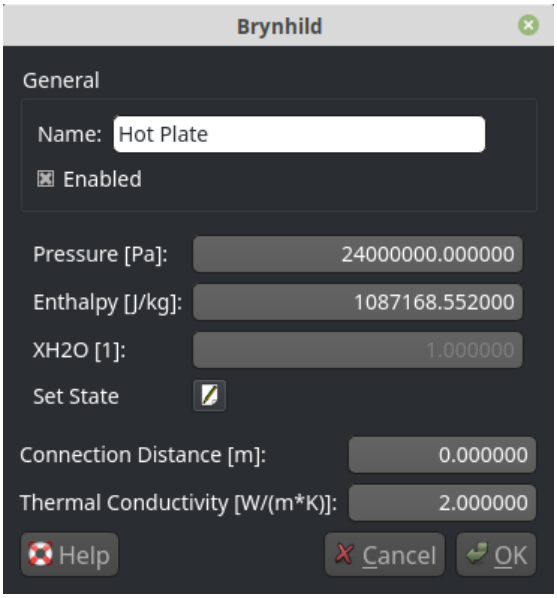

Figure 4.5: The Hot Plate Parameters

on one side. The second connection distance can be set by the user; the default is $0m$. You will also need to specify the thermal conductivity. This boundary condition is set to impermeable, i.e. it can not exchange fluid with the reservoir.

Extra Block List

The extra block list boundary condition lets you define an extra block outside the cell grid. You need to manually edit all connection properties and cell locations to connect to, i.e. you can not edit this boundary condition interactively in the viewer.

Properties labelled with "1" will belong to the cell specified by location and MINC layer id. Properties labelled with "2" will belong to the extra block. The gravitational acceleration needs to be negative if the extra block is located above the the cell.

A word of caution when using this boundary condition in combination with a GPU accelerator: If this boundary condition connects to a large number of cells then it is advisable to use it as a fixed state only. If the element uses the enabled state then the GPU accelerator can struggle or even run out of memory, resulting in a crash of the simulation.

Manual Extra Blocks

The following extra block boundary conditions set up a fixed state extra block and allow you to manually specify all connection parameters. Note that parameters labelled with a "1" refer to parameters inside the cell this boundary condition interacts with, while "2" refers to parameters inside the extra block. Figure 4.6 shows an example.

Note for 2D interaction manifolds you can opt to automatically calculate the interface area instead of setting it manually.

Load from File

You can load the manifold for the extra block boundary condition from a file and perform a rotation/translation. See section D.1.1 and above for details.

Poly Vertex

You can use a list of points to create a Poly Vertex manifold. See D.1.2 and above for details.

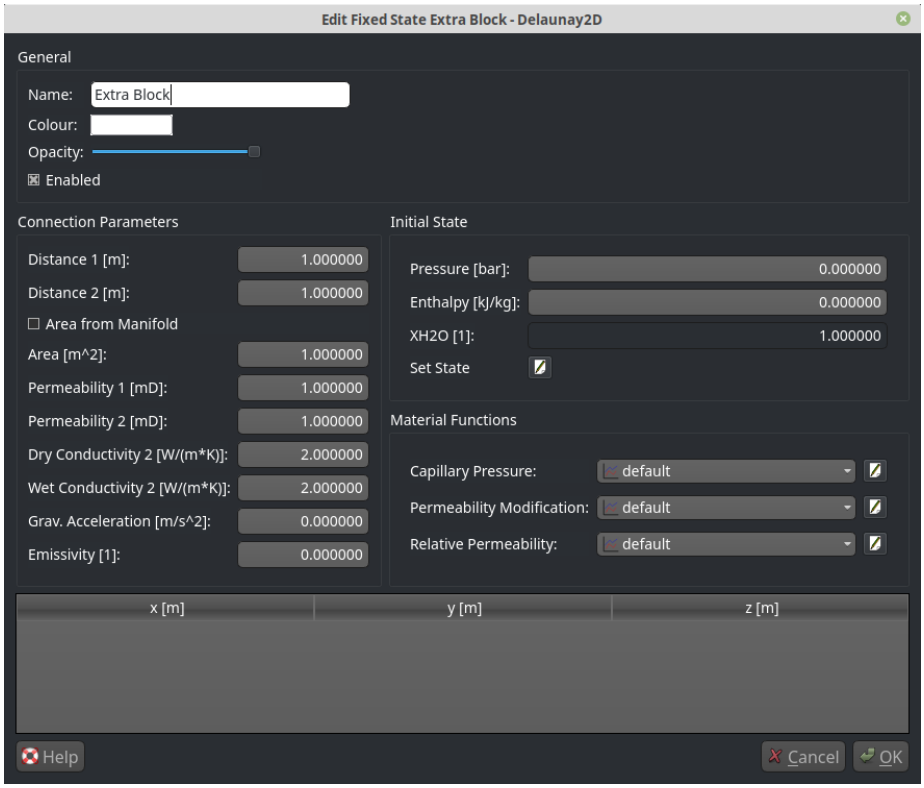

Figure 4.6: An Extra Block Boundary Condition.

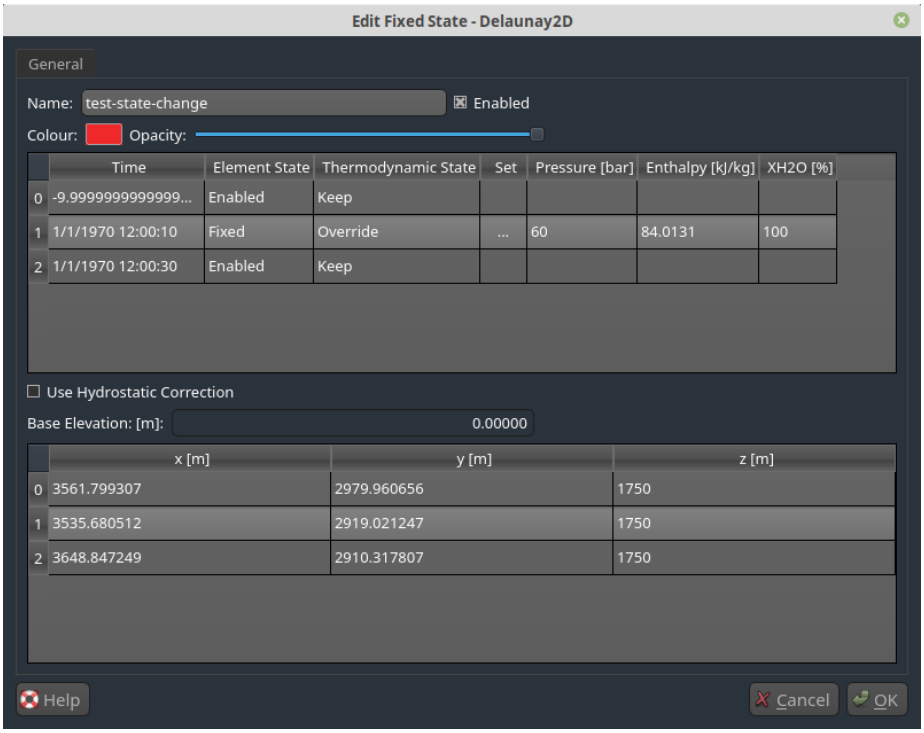

Figure 4.7: The Fixed State Boundary Condition Dialog

Poly Line

You can use a list of points to create a Poly Line manifold. See D.1.4 and above for details.

Polygon

You can use a list of points to create a Polygon manifold. See D.1.5 and above for details.

Delaunay 2D

You can use a list of points to create a Delauny 2D manifold. See D.1.7 and above for details.

4.4.2 Fixed States

Figure 4.7 shows the basic fixed state boundary condition dialog. The upper part contains the standard name, colour and opacity fields.

The middle section contains a table where you can define state changes over time. You can opt to change the element state to be either enabled, fixed or disabled. Also you can opt to keep the thermodynamic state which the elements have at the time of the state change or to override it with a new thermodynamic state.

If the table is left blank then the default behaviour is make affected cells fixed state but not change their thermodynamic state.

The bottom part of the dialog is specific to the manifold type of the boundary condition.

Load from File

You can load the manifold for the fixed state boundary condition from a file and perform a rotation/translation. See section D.1.1 for details.

Poly Vertex

You can use a list of points to create a Poly Vertex manifold. See D.1.2 for details.

Poly Line

You can use a list of points to create a Poly Line manifold. See D.1.4 for details.

Polygon

You can use a list of points to create a Polygon manifold. See D.1.5 for details.

Delaunay 2D

You can use a list of points to create a Delauny 2D manifold. See D.1.7 for details.

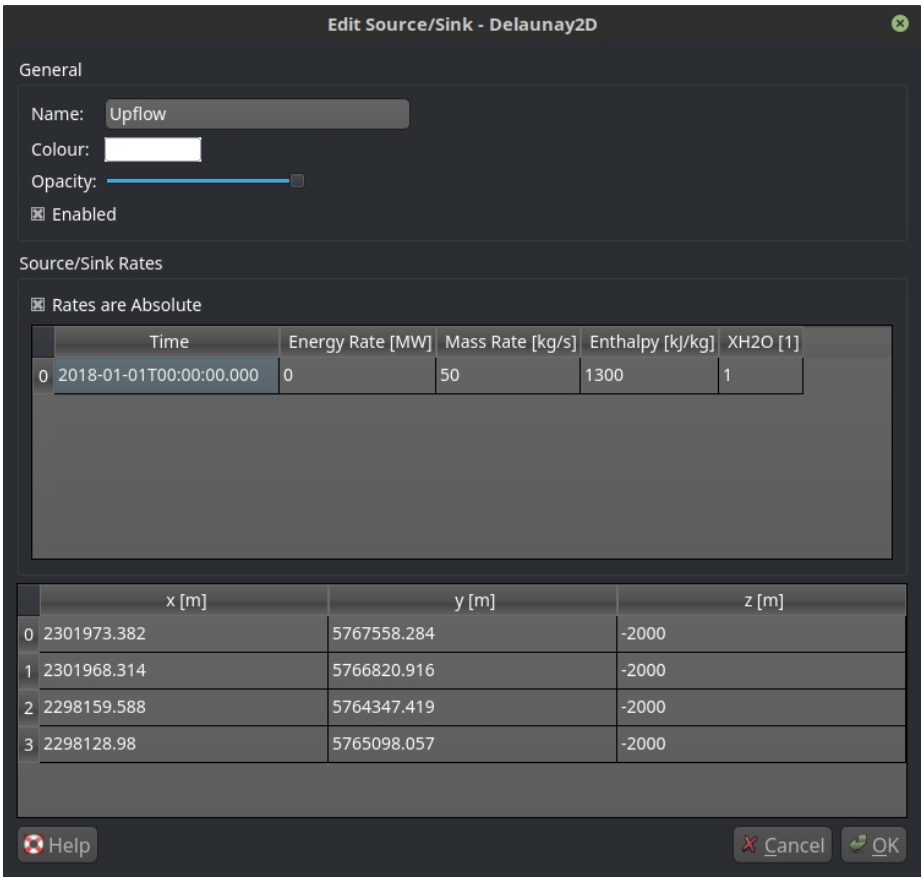

Figure 4.8: The Source/Sink Boundary Condition Dialog

4.4.3 Sources/Sinks

Figure 4.8 shows the basic source/sink boundary condition dialog. The upper part contains the standard name, colour and opacity fields.

The middle section contains a rate table for entering the flow rates. Note you will need to provide full detail for the fluid for injection but only the mass rate for production (see 4.2.2 for details). Note that $+ve$ values for the mass/energy rate corresponds to injection and $-ve$ values for production.

Source/sink rates can be *absolute* or *relative*. Absolute rates are interpreted as they are entered in the rate table and distributed into cells using the manifold interaction divided by the total manifold interaction (see section D).

When using relative rates then the value entered for the rates in the table is interpreted as *manifold dimensional unit*. For example, if you enter $w = 1kg/s$ and the manifold has $2D$ dimension then it will be interpreted as $w = 1kg/(s \cdot m^2)$. The manifold dimensional unit will always be in SI units, i.e. 1 for vertices, m for lines, m^2 for surfaces.

The bottom part of the dialog is specific to the manifold type of the boundary condition.

Load from File

You can load the manifold for the source/sink boundary condition from a file and perform a rotation/translation. See section D.1.1 for details.

Poly Vertex

You can use a list of points to create a Poly Vertex manifold. See D.1.2 for details.

Poly Line

You can use a list of points to create a Poly Line manifold. See D.1.4 for details.

Polygon

You can use a list of points to create a Polygon manifold. See D.1.5 for details.

Delaunay 2D

You can use a list of points to create a Delauny 2D manifold. See D.1.7 for details.

Grid Top

For this source/sink boundary condition the grid top is used as interaction manifold, i.e. the source/sink will attach to the top grid layer. This is an invisible manifold and hence you can not enter colour or opacity information.

This boundary condition is useful for describing surface precipitation.

Grid Bottom

For this source/sink boundary condition the grid bottom is used as interaction manifold, i.e. the source/sink will attach to the bottom grid layer. This is an invisible manifold and hence you can not enter colour or opacity information.

This boundary condition is useful for describing a heat flux into the bottom layer.

Grid Sides

For this source/sink boundary condition the grid sides are used as interaction manifold, i.e. the source/sink will attach to the grid side cells. This is an invisible manifold and hence you can not enter colour or opacity information.

Single Point

This source/sink boundary condition is a point source/sink, i.e. it is defined by a single point (the base point) in space.

Pressure Transient Analysis (PTA)

This is a specialized source/sink for numerical pressure transient analysis.

Lithology

The lithology describes the material properties of the lithological units which we use in the reservoir simulation. We need to distinguish two different sets of parameters which determine the properties of the unit:

- *Rock Properties* describe the basic properties of rocks. Each element is internally assigned one rock type.
- *MINC Properties* describe the internal layer structure of a lithological unit.

We will start by introducing the material functions; these describe rock properties which can change with the thermodynamic state of the fluid. After that we will introduce the MINC layer model for describing fracture/matrix interaction. At the end of this chapter we will introduce the remaining properties and how the *Brynhild* lithological unit data base can describe the units which fill the grid blocks.

5.1 Material Functions

5.1.1 Relative Permeability Functions

The relative permeability is part of the Darcy-Flux equation 4.13. Different formulations for the calculating the relative permeability of a material exist in literature. If not stated otherwise then the mathematical formulations for these functions in *Volsung* are taken from the *TOUGH2* manual [Pruess et al., 1999].

Note that when a solid phase is present the relative permeability functions renormalize their gas/liquid phase saturations, i.e. the solid phase does not have an effect on the relative permeability.

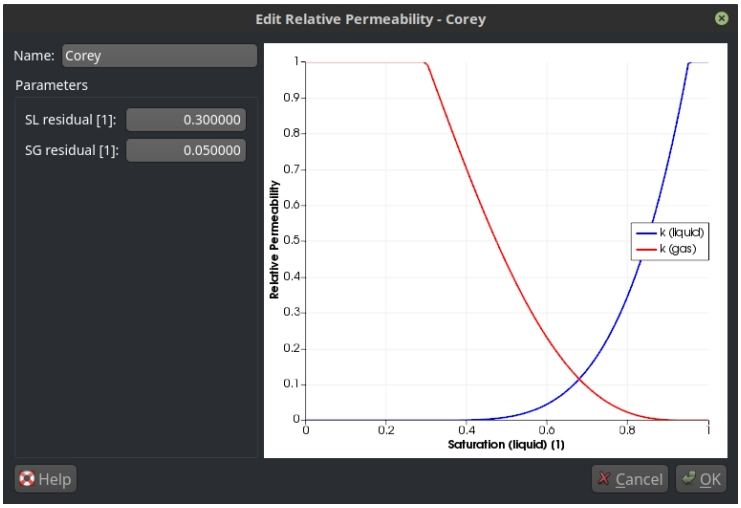

Figure 5.1: Corey Relative Permeability Function

Corey

The default formulation is

$$\hat{S} = \frac{S_{liq} - S_{r,liq}}{1 - S_{r,liq} - S_{r,gas}} \tag{5.1}$$

$$k_{r,liq} = \hat{S}^4 \tag{5.2}$$

$$k_{r,gas} = (1 - \hat{S})^2 \cdot (1 - \hat{S}^2) \tag{5.3}$$

with restriction $S_{r,liq} + S_{r,gas} < 1$.

For $S_{liq} < S_{r,liq}$ we use $k_{r,liq} = 0, k_{r,gas} = 1$.

For $S_{gas} < S_{r,gas}$ we use $k_{r,liq} = 1, k_{r,gas} = 0$.

Fatt and Klikoff

The default formulation is

$$\hat{S} = \frac{S_{liq} - S_{r,liq}}{1 - S_{r,liq}} \tag{5.4}$$

$$k_{r,liq} = \hat{S}^3 \tag{5.5}$$

$$k_{r,gas} = (1 - \hat{S})^3 \tag{5.6}$$

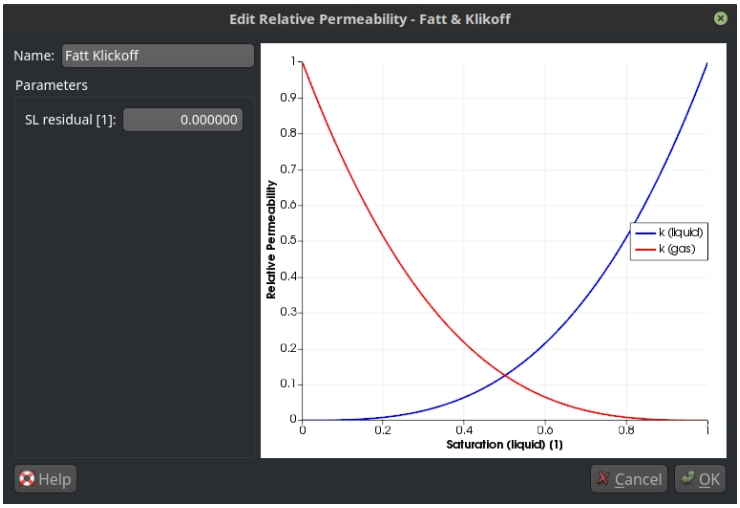

Figure 5.2: Fatt-Klikoff Relative Permeability Function

with restriction $S_{r,liq} < 1$.

For $S_{liq} < S_{r,liq}$ we use $k_{r,liq} = 0, k_{r,gas} = 1$.

Grant

The default formulation is

$$\hat{S} = \frac{S_{liq} - S_{r,liq}}{1 - S_{r,liq} - S_{r,gas}} \tag{5.7}$$

$$k_{r,liq} = \hat{S}^4 \tag{5.8}$$

$$k_{r,gas} = 1 - k_{r,liq} \tag{5.9}$$

with restriction $S_{r,liq} + S_{r,gas} < 1$.

For $S_{liq} < S_{r,liq}$ we use $k_{r,liq} = 0, k_{r,gas} = 1$.

For $S_{gas} < S_{r,gas}$ we use $k_{r,liq} = 1, k_{r,gas} = 0$.

Linear

$k_{r,liq}$ increases linearly between $S_{lower,liq}$ and $S_{upper,liq}$, with restriction $S_{lower,liq} < S_{upper,liq}$.

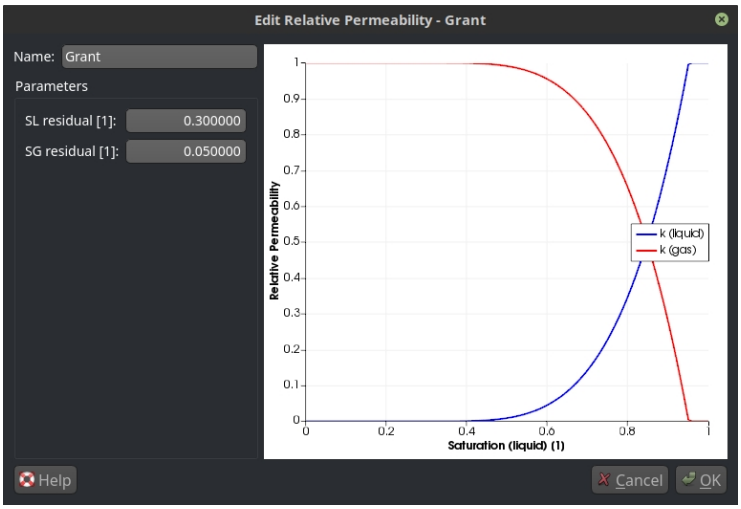

Figure 5.3: Grant Relative Permeability Function

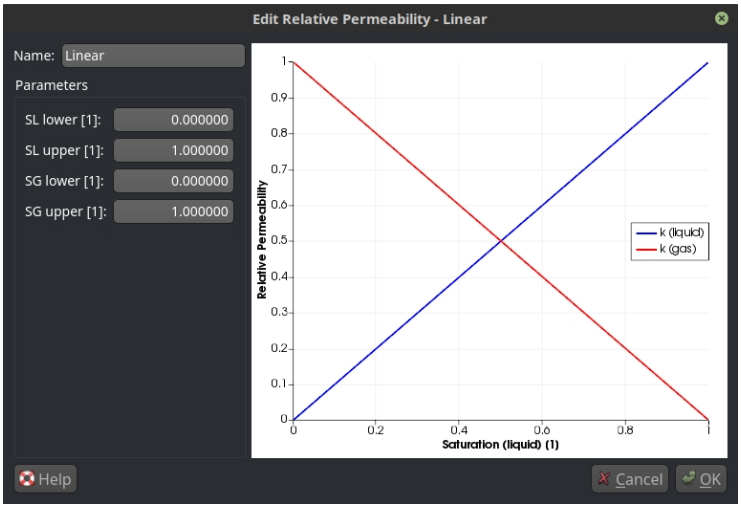

Figure 5.4: Linear Relative Permeability Function

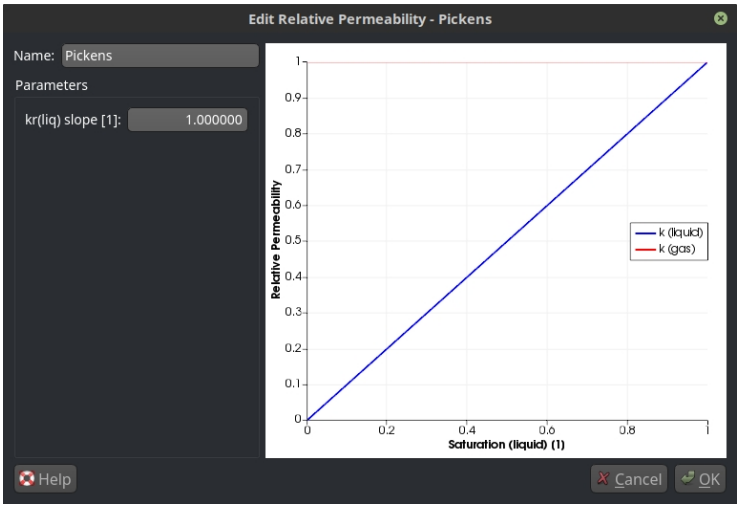

Figure 5.5: Pickens Relative Permeability Function

$k_{r,gas}$ increases linearly between $S_{lower,gas}$ and $S_{upper,gas}$, with restriction $S_{lower,gas} < S_{upper,gas}$.

None

This is a special relative permeability function introduced in *Volsung*. It calculates the relative permeability in such manner that the resulting flowing fluid composition equals the static fluid composition.

Perfectly Mobile Phases

The relative permeabilities are $k_{rel,liq} = 1$ for all S_{liq} and $k_{rel,gas} = 1$ for all S_{gas}.

Pickens

$k_{r,liq}$ increases linearly using $k_{r,liq} = k_{r,liq}^{\cdot} S_{liq}$ with restriction $k_{r,liq}^{\cdot} > 0$.

$k_{r,gas} = 1$ for all S_{gas}.

Richards S-Curves

Richards S-Curves use the modified Richards function:

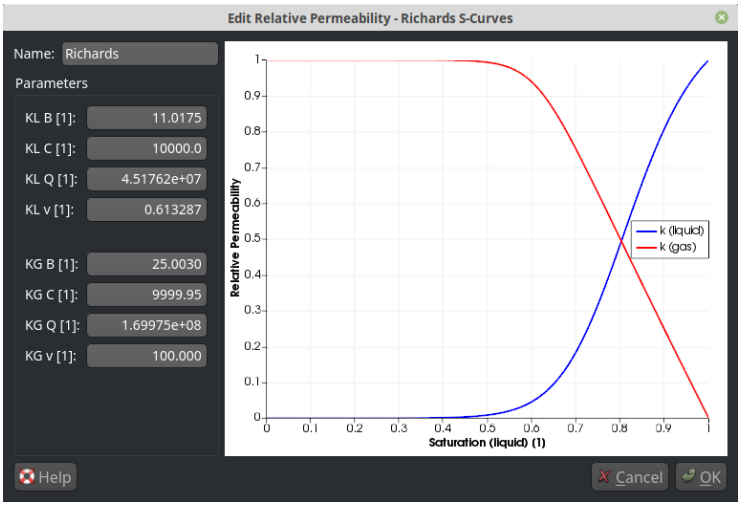

Figure 5.6: Richards S-Curve Relative Permeability Function

$$k_{rel}(S) = A + \frac{K - A}{(C + Q \cdot \exp(-B \cdot S))^{1/v}} \tag{5.10}$$

K and A are computed internally to satisfy $k_{rel}(S) = 0$ and $k_{rel}(1) = 1$; using auxiliary variables u and w we get

$$\begin{aligned} u &= (C + Q)^{1/v} && (5.11)\\ w &= (C + Q \cdot \exp(-B))^{1/v} && (5.12)\\ A &= w/(w - u) && (5.13)\\ K &= -A \cdot (u - 1) && (5.14) \end{aligned}$$

The parameters are complex to describe; Wikipedia gives a good overview over the parameters as well as some examples how the function shape changes by modifying the individual parameters. See `https://en.wikipedia.org/wiki/Generalised_logistic_function` for details.

The main advantage for using Richards curves is that they are smooth, i.e. their first derivative is continuous. This can vastly aid the convergence of the Newton iteration scheme in *Fafnir*. We hence highly recommend the use of this relative permeability function.

The disadvantage of the curves is that one needs to manually find a parameter set to approximate another relative permeability function. To make this process easier you can find the `fitRichards.py` script in the examples which you can modify to find customized parameters.

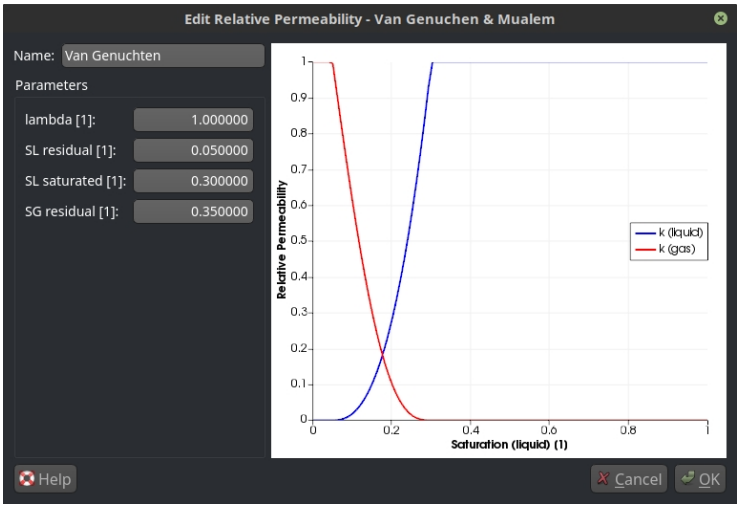

Figure 5.7: Van Genuchten - Mualem Relative Permeability Function

Van Genuchten - Mualem

For the liquid phase the relative permeability formulation is

$$\tilde{S} = \frac{S_{liq} - S_{r,liq}}{S_{s,liq} - S_{r,liq}} \tag{5.15}$$

$$k_{r,liq} = \sqrt{\tilde{S}} \cdot \left(1 - \left(1 - \tilde{S}^{1/\lambda}\right)^{\lambda}\right)^2 \tag{5.16}$$

if $S_{liq} < S_{s,liq}$, else $k_{r,liq} = 1$.

The gas phase relative permeability is

$$\hat{S} = \frac{S_{liq} - S_{r,liq}}{1 - S_{r,liq} - S_{r,gas}} \tag{5.17}$$

$$k_{r,gas} = (1 - \hat{S})^2 \cdot (1 - \hat{S}^2) \tag{5.18}$$

if $S_{r,gas} > 0$, else $k_{r,gas} = 1 - k_{r,liq}$. A restriction is that $0 \leq k_{r,liq}$ and $k_{r,gas} \leq 1$.

Verma

The default formulation for the liquid phase relative permeability is

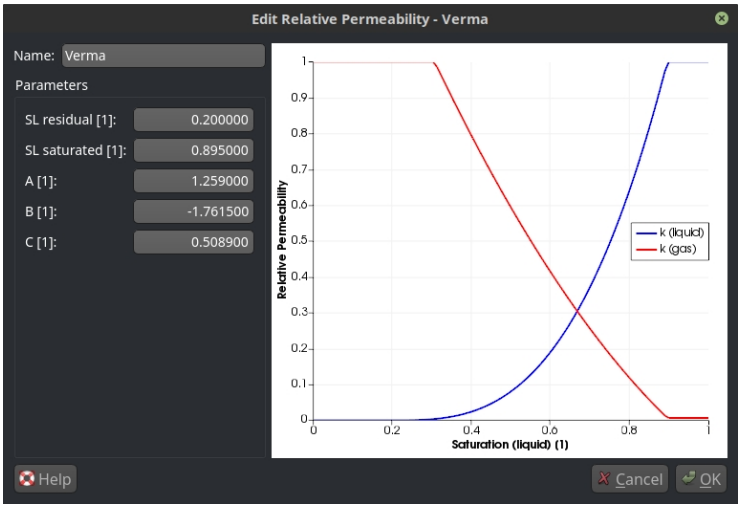

Figure 5.8: Verma Relative Permeability Function

$$\tilde{S} = \frac{S_{liq} - S_{r,liq}}{S_{s,liq} - S_{r,liq}} \tag{5.19}$$

$$k_{r,liq} = \tilde{S}^3 \tag{5.20}$$

The gas phase relative permeability is

$$k_{r,gas} = A + B \cdot \tilde{S} + C \cdot \tilde{S}^2 \tag{5.21}$$

The default parameters are

$$S_{r,liq} = 0.2 \tag{5.22}$$

$$S_{s,liq} = 0.895 \tag{5.23}$$

$$A = 1.259 \tag{5.24}$$

$$B = -1.7615 \tag{5.25}$$

$$C = 0.5089 \tag{5.26}$$

5.1.2 Capillary Pressure Functions

The capillary pressure function describes the "suction" effect between the rock matrix and the fluid. It reduces the effective pressure available for transport; see equation 4.14.

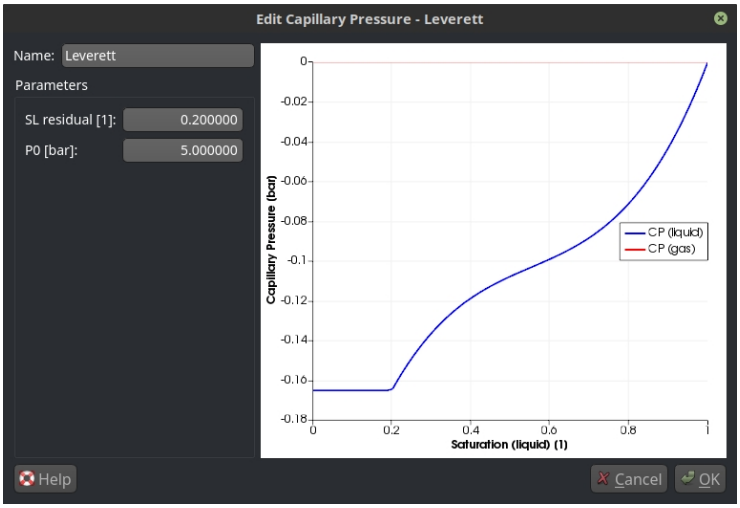

Figure 5.9: Leverett Capillary Pressure Function

While in general both liquid and gas phases can be effected by capillary pressure the gas phase effect is quite small and is usually neglected. Hence we only give the liquid capillary pressure functions below and assume $P_{cap,gas} = 0$.

Note that when a solid phase is present the capillary pressure functions re-normalize their gas/liquid phase saturations, i.e. the solid phase does not have an effect on the capillary pressure.

Leverett

The default formulation is

$$P_{cap} = -P_0 \cdot \sigma(T) \cdot f(S_{liq}) \tag{5.27}$$

$$f(S_{liq}) = 1.417 \cdot (1 - \tilde{S}) \cdot (1 - \tilde{S})^2 + 1.263 \cdot (1 - \tilde{S})^3 \tag{5.28}$$

$$\tilde{S} = \frac{S_{liq} - S_{r,liq}}{1 - S_{r,liq}} \tag{5.29}$$

where $\sigma(T)$ is the surface tension and we have the restriction $0 \leq S_{r,liq} < 1$. For $S_{liq} < S_{r,liq}$ we use $\tilde{S} = 0$.

Linear

The capillary pressure increases linearly between $S_{lower,liq}$ and $S_{upper,liq}$. The lower value of the capillary pressure is $-P_0$ and the upper value is zero.

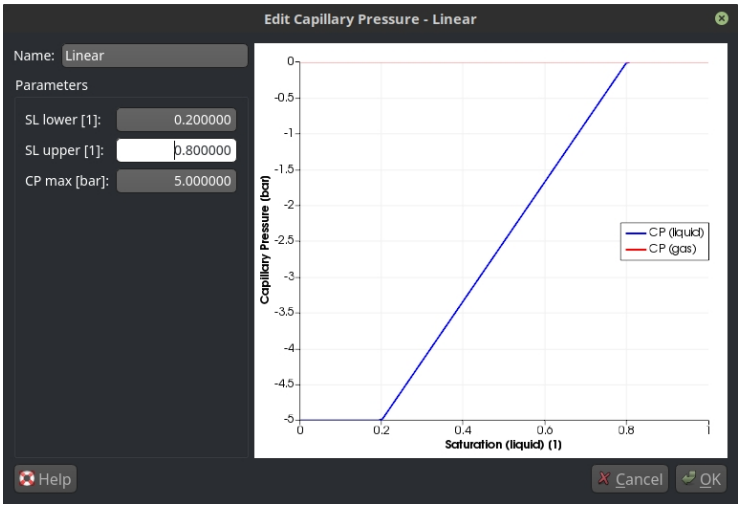

Figure 5.10: Linear Capillary Pressure Function

Milly

This capillary pressure function is provided for compatibility reasons with *TOUGH2* and is implemptend using the *TOUGH2* description. However it looks like this capillary pressure function is not correctly implemented and should preferably not be used.

None

This capillary pressure function has zero capillary pressure.

Pickens

This capillary pressure function is provided for compatibility reasons with *TOUGH2* and is implemptend using the *TOUGH2* description. However it looks like this capillary pressure function is not correctly implemented and should preferably not be used.

Van Genuchten

This capillary pressure function is implemented using the *TOUGH2* description.

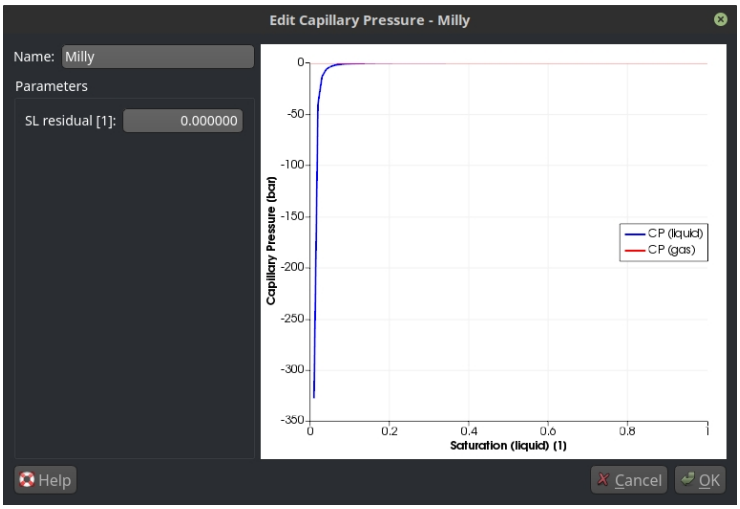

Figure 5.11: Milly Capillary Pressure Function

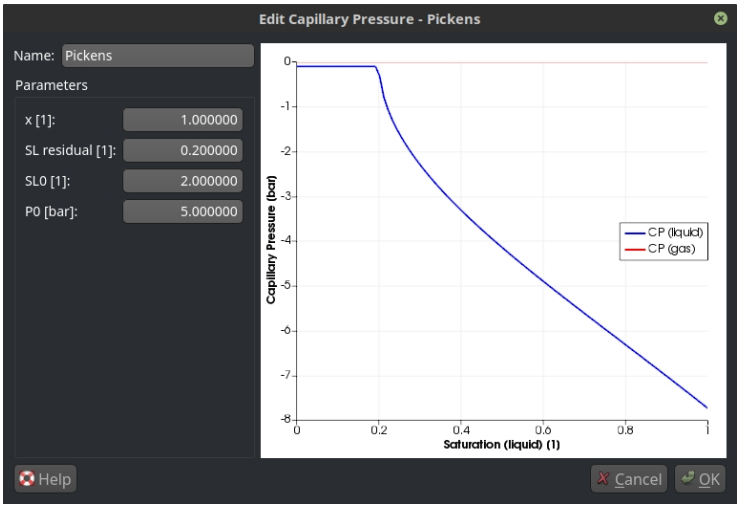

Figure 5.12: Pickens Capillary Pressure Function

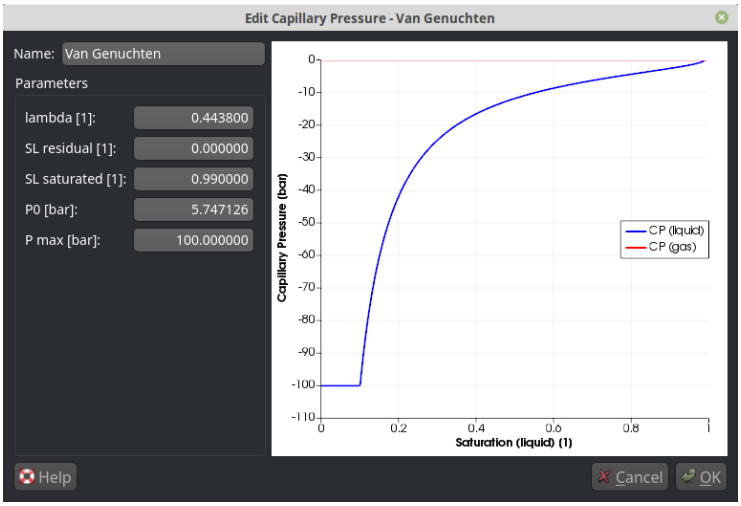

Figure 5.13: Van Genuchten Capillary Pressure Function

5.1.3 Permeability Modification Functions

Permeability modification functions provide a method to change the permeability of an element via changes in its effective relative porosity:

$$\hat{\phi} = \phi_f / \phi_0 \tag{5.30}$$

Here ϕ_0 is the original porosity as assigned via the rock properties. ϕ_f is the current effective porosity:

$$\phi_f = \phi(t) \cdot (1 - S_{solid}) \tag{5.31}$$

This takes into account that the porosity itself can change over time (for example through geomechanical or thermal effects) and that some part of the porosity may be occupied by a solid phase (salt or chemical deposition).

The permeability modification function describes how the permeability changes relative to its original permeability:

$$k = k_0 \cdot f_{pm}(\hat{\phi}) \tag{5.32}$$

None

The default permeability modification function is $f_{pm} = 1.0$, i.e. the permeability is not changed.

Verma and Pruess

This permeability modification is built on the function given by Verma and Pruess [1988] in Pruess et al. [1999]. It has the form:

$$\omega = 1 + \frac{1/\Gamma}{1/\hat{\phi}_r - 1} \tag{5.33}$$

$$\tag{5.34}$$

$$\theta = \frac{\hat{\phi} - \hat{\phi}_r}{1 - \hat{\phi}_r} \tag{5.35}$$

$$\tag{5.36}$$

$$f_{pm} = \theta^{\alpha} \frac{1 - \Gamma + \Gamma/\omega^{\alpha}}{1 - \Gamma + \Gamma \cdot (\theta/(\theta + \omega - 1))^{\alpha}} \tag{5.37}$$

Here $\hat{\phi}_r$ is the residual relative effective porosity below which the permeability will become zero. Γ is the relative length of the parallel-in-series or fractures-in-series (see Pruess et al. [1999] for details). The exponent α needs to be set according to the desired geometry; use the following:

- straight capillary tubes in series: $\Gamma = 0$, $\hat{\phi}_r = 0$, $\alpha = 2$
- tubes in series: $\alpha = 2$
- parallel plate fractures in series: $\alpha = 3$.

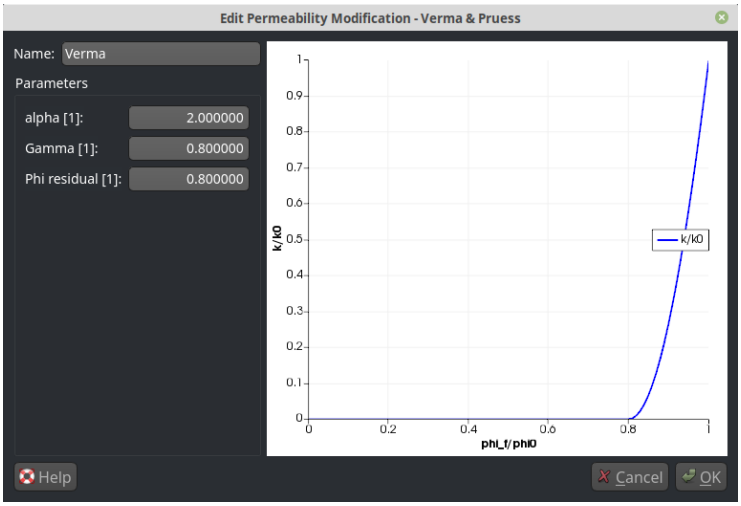

Figure 5.14: Verma and Pruess permeability modification function.

5.2 MINC

Fafnir makes use of the MINC (Multiple INteracting Continua) approach to better approximate the transport of fluid and heat in fractured media. A usually small outer layer - the fracture - is connected to the neighbouring blocks using relatively high permeability. The first inner nested layer - the first matrix layer - is connected to the fracture layer in the same block and to subsequent further inner layers. This model has also been known as a "dual porosity" model if only the fracture and one matrix layer are present.

This model in various variations has been described by Warren and Root [1963], Pruess et al. [1999], Pruess [1983] and Pruess [1992]. The mathematics can be a bit hard to understand; we recommend you to read Thunderhead [1999] which gives a very concise and detailed description of the equations and concepts used; *Fafnir* uses the equations given by them.

The basic concept is that fluid is transported quickly between fractures of neighbouring blocks due to the high permeability used for the fractures. Matrix elements can provide heat (via a relatively high surface area) to the fracture and can hence buffer temperature transients in the fracture. They can also provide some pressure support to the fracture since they can contain large amounts of fluid which is slowly transported into the fracture if the pressure in the fracture drops.

The key parameters for the MINC system are the fracture spacings and the volume fractions. The fracture spacings determine the connection area and distance between fracture and matrix; the smaller the fracture spacings the faster heat and fluid are transported from the matrix into the fracture. Systems with very small fracture spacings closely resemble a single porosity model while systems with

large fracture spacings are more similar to chunks of hot rock which will keep their temperature stable over long periods of time.

The volume fractions determine how much volume relative to the total block volume is associated with a fracture/matrix layer. Typically fracture layers have a small volume fraction ($< 5\%$) and a high porosity (like 95%) while matrix layers have lower porosities ($\approx 5 - 20\%$). The volume fractions of the matrix layers depend on how many matrix layers are present; all layer volume fractions together must add up to 1.0.

5.3 Units

Blocks in *Fafnir* are filled by assigning them with lithological units. These units describe the MINC (or single porosity) structure of the grid block, contain the basic rock properties and an initial state.

A lithological unit is described via its unique name in the lithology data base. A colour can be used to better identify geological elements or grid blocks associated with the unit.

The base element state can be chosen so that blocks filled with a lithological unit can contain enabled, fixed state or disabled elements. This base state can be subsequently modified by the boundary conditions.

The type describes if the unit is single porosity or a $1D$, $2D$ or $3D$ fracture/matrix system.

If a MINC system is chosen then the number of MINC matrix layers can be chosen individually for each unit; this allows to keep the overall number of computational elements small. Depending on the $1D$, $2D$ or $3D$ nature of the MINC system you will need to enter up to three fracture spacings for the unit.

Depending on the number of MINC layers chosen the dialog in *Brynhild* will contain one or more tabs to define the rock properties of the fracture and the individual matrix layers. You will need to set the layer volume fractions if a MINC system is chosen. All layer volume fractions must add up to 1.0; this is done automatically by adjusting the fracture volume fraction accordingly.

The basic properties are rock density, porosity, specific heat and wet/dry thermal conductivities. Compressibility and expansivity describe the change of porosity with pore pressure and temperature, respectively; their use is described in more detail in section 7.

Material functions can be chosen individually for each unit and fracture or matrix layer.

The permeability is entered as a tensor by specifying permeabilities in x, y and

z direction plus giving up to three angles to rotate the ellipsoid to the desired orientation. Usually only the first azimuth angle needs to be entered to define the main orientation of the grid; see equation 4.22 for details. Note that matrix layers only allow you to enter a scalar permeability. You can also choose to make the layer impermeable in which case it can only exchange heat but no fluid with neighbouring blocks.

You will also need to enter an initial state for the unit; this is used for cold starting the reservoir simulation only, i.e. this state will be overwritten if you load the initial reservoir state from a previous SAVE file. You can use the thermodynamic table tool (see section 12.2.1) to help you with selecting a valid state. The hydrostatic correction can be used to correct the pressure via a hydrostatic gradient using the mixture density of the state, but note that this can in some cases lead to illegal states (like negative pressures). However using the hydrostatic correction can improve the initial convergence of a model significantly so its use is highly recommended.

The Geology Model

The purpose of the geology model in *Brynhild* is to determine how to fill the individual grid blocks with lithological units. There are a different number of models available to facilitate this task:

- You can provide a simple text file based model which contains a list of lithological unit names and specifies the lithological unit for each named block in the grid,
- you can build a geology model directly in the reservoir editor of *Brynhild* using surface layer shapes,
- or you can use *Leapfrog* as an external model building tool and use LFVM files describing volumes of different lithological units.

The default method in *Brynhild* is to build the model within the GUI using two dimensional shapes to describe lithological layers or faults. You can change the geology model type by right-clicking on the Geology Model item in the navigator and selecting another model type.

6.1 Common Features

6.1.1 Geology Barriers

Barriers are $2D$ shapes which can modify individual connection permeabilities. A connection is modified if the connection vector between two cells intersects the barrier manifold. Barriers are only applied between connections of the outermost (fracture) layer. Their order in the navigator determines their priority; when conflicts between barriers are present then the barrier higher up in the navigator list is prioritized.

Two types are available: relative barriers use a scalar factor to multiply both permeabilities (k_1 and k_2) and the heat conductivities (both dry and wet) of the individual connections. You can select if you want to scale only permeabilities, only heat conductivities, or both.

Absolute types use a permeability tensor and replace both permeabilities of the connection with the scalar permeability given by the connection vector projection.

See section D for explanations about the different manifold types.

6.2 Layer Based Geology Model

In this model *layers* and *faults* are used to fill the grid blocks with lithological units. Both are similar; when creating a new layer or fault you need to specify its type which the same as using the manifolds. You can only select $2D$ manifold types for the layers but faults can be any of the manifold types supported.

First you will need to select a default lithological unit. When building the geology model all grid blocks are first filled with this unit before the layers and faults are applied.

6.2.1 Geology Layers

The layers are an ordered structure. The unit for a grid block is determined by shooting a ray upward from the grid block centre. The layer list is processed from the bottom layer upwards; if the ray intersects with the layer in the list then it will be associated with the unit of this layer. Further layers higher up in the list will not be tested.

See section D for explanations about the different manifold types.

6.2.2 Geology Faults

The faults are also an ordered structure. They will be processed from the top fault in the list downwards. Any block which is touched by a fault will be associated with the unit of the fault. The faults hence override the unit selection of the layers and can also override each other, i.e. the lowest fault in the list may ultimately decide the unit for a block.

Note that while we're talking of "faults" here they can really be used for other shapes as well. For example you could model a surface river as a "fault".

See section D for explanations about the different manifold types.

6.3 File Based Geology Model

In a file based geology model a file - which is created and edited externally to *Brynhild*- is used to associate the cells of the grid with lithological units. You will need to set the file name of this external model; you can also opt to put a watch on this file so if it changes some visual parameters in *Brynhild* are updated automatically.

The file format is best explained on a short example:

```
# Block Lithology
# Comments, optional
# Comments, optional
#,LithoCode,Lithology[,R,G,B]
#,0,BaseRock,255,255,255
#,-1,BaseLithology
#,1,Rock1
#,2,Rock2,0,125,0
BlockName,LithoCode
bv 1,0
ca 1,0
ch 1,0
ci 1,2
cj 1,0
cn 1,0
co 1,1
cp 1,0
cq 1,0
```

The first line must contain the words `Block Lithology`; this is used internally as a file type identifier.

The next lines can contain optional comments.

The line `#,LithoCode,Lithology` marks the beginning of the section where the lithological units are described. The following lines must contain an integer identifier for the unit and a unique unit name. R,G and B are optional colour information for the units (red, green and blue); each value is an integer in the range $[0, 255]$. You don't need to give this colour information for each unit, but when you do it will take preference over colour choices you make within *Brynhild*.

The last section starts with `BlockName,LithoCode`. The following lines contain the name of the grid cell followed by the integer unit identifier.

Note that this file format is compatible with *TOUGH2* output from *Leapfrog* so you can use *Leapfrog* directly to build your geological model.

6.4 Cell Based Geology Model

The cell based geology model is similar to the file based geology model, i.e. it assigns a lithological unit to each of the cells in the grid. However this model is fully editable from within the GUI.

Cells which have not manually been assigned a lithological unit will use the default unit which you can set in the navigator panel.

This geology model features a preview mode so you can see which cell is associated with which lithological unit. You can get into the preview mode by selecting the `Lithology Unit (Preview)` cell array in the viewer.

To change the unit for cells select them with the mouse (see section 3.7). Then right-click on the selection to bring up the context menu which allows you to assign a lithology unit to the selected cells.

6.4.1 Re-gridding and Probing Existing Model

Since cell based geology models only contain an association between a cell index and the lithological units their association is void once you move a model to a new grid. If you simply keep using your current geology model you will see that once your model has run with a new grid the lithological units may be distributed in a weird pattern.

To facilitate re-gridding and preserving as much as possible of your geology model you can probe a model with a new grid using the results from a previous model. You will need to prepare for re-gridding using the following steps else you will void your geology model:

1. *Before* re-gridding your model run or test run your previous version at least once so you have a valid output file.
2. Create a safe copy of this output file, i.e. copy `Results.sigurd` to `old-model.sigurd`.
3. Re-grid your model.
4. Test-run your model as is, i.e. without changing anything in the geology model.
5. Right-click on the *Geology Model* item in the navigator panel and select *Probe existing model*.
6. When queried select your safe copy of the old model, e.g. `old-model.sigurd`, for probing.

The new geology model uses the new cell centres to probe the lithology of the old model results. If it can't determine the old lithology for a location it will select the currently selected default lithological unit. It will also create new blank lithological units in the data base if the old model contains an unknown lithological unit name.

Rock Mechanics

Volsung currently does not have a full module to deal with the thermal stresses and strains occurring in a simulation. However since we at a later stage plan to introduce rock mechanics simulations we treat this section as a minimalistic placeholder which can be expanded later on as needed.

7.1 Fundamental Elasticity Equations

Consider a piece of homogeneous, isotropic rock containing some porous space. We denote the bulk volume of the rock with V_b and the pore volume as V_p. We can relate the pore volume to the bulk volume by using the porosity:

$$\phi = V_p / V_b \tag{7.1}$$

Consider that this piece of porous rock is subjected to a confining pressure p_c and a pore pressure p_p exerted by the pore fluid. We can then define four compressibilities [Jaeger et al., 2007]:

$$c_{bc} = -\frac{1}{V_b^i} \cdot \left(\frac{\partial V_b}{\partial p_c} \right)_{p_p} \tag{7.2}$$

$$c_{bp} = +\frac{1}{V_b^i} \cdot \left(\frac{\partial V_b}{\partial p_p} \right)_{p_c} \tag{7.3}$$

$$c_{pc} = -\frac{1}{V_p^i} \cdot \left(\frac{\partial V_p}{\partial p_c} \right)_{p_p} \tag{7.4}$$

$$c_{pp} = +\frac{1}{V_p^i} \cdot \left(\frac{\partial V_p}{\partial p_p} \right)_{p_c} \tag{7.5}$$

where the superscript i denotes the initial, unstressed state. The bulk and pore

strain increments can be expressed in terms of the porous rock compressibilities:

$$d\epsilon_b = -\frac{dV_b}{V_b^i} \tag{7.6}$$
$$= c_{bc} \cdot dp_c - c_{bp} \cdot dp_p \tag{7.7}$$

and

$$d\epsilon_p = -\frac{dV_p}{V_p^i} \tag{7.8}$$
$$= c_{pc} \cdot dp_c - c_{pp} \cdot dp_p \tag{7.9}$$

where ϵ_b and ϵ_p describe the bulk and pore strain, respectively.

Similarly we can define thermal expansivities. However if we assume that pore fluid and rock are in thermal equilibrium then we need to only consider the bulk and pore thermal expansivities

$$\beta_b = -\frac{1}{V_b^i} \cdot \left(\frac{\partial V_b}{\partial T}\right) \tag{7.10}$$
$$\beta_p = +\frac{1}{V_p^i} \cdot \left(\frac{\partial V_p}{\partial T}\right) \tag{7.11}$$

and thermal strains

$$d\epsilon_b = -\beta_b \cdot dT \tag{7.12}$$
$$d\epsilon_p = -\beta_p \cdot dT \tag{7.13}$$

Note that in above equations β is a volumetric expansivity and is related to the linear expansivity coefficient α_L by $\beta = 3 \cdot \alpha_L$.

7.2 Geomechanical Simulators

Volsung has the capability to select different geomechanical simulators as required. To change the geomechanical simulator right-click on the *Geo Mechanical* item in the navigator and replace the current type with the one you want to use.

7.2.1 None

This simulator ignores all geomechanical interactions. The material properties entered (compressibility, expansivity et etc.) are ignored.

7.2.2 Poroelasticity

This is a basic geomechanics simulator which will only consider the change in porosity due to changes in pressure and temperature, i.e.

$$\frac{dV_p}{V_p^i} = \beta_p \cdot dT + c_{pp} \cdot dp_p \tag{7.14}$$

The bulk volume is not changed, i.e. $dV_b = 0$ and no change in confining pressure is considered, i.e. $dp_c = 0$.

The change in pore volume is treated *implicitly*, i.e. updates are performed within the Newton iterations by considering the change in temperature and pressure versus the initial state of the current time step. At the end of the time step the porosity and rock density arrays will have changed - the later to reflect compression of the rock mass since total rock mass must be conserved. If you want to continue a simulation from a SAVE file you should check that the simulator reloads these arrays from the initial state file.

You can select if this simulator is active during the natural state, calibration and scenario periods; if not active then the porosity remains unchanged.

The Surface Network Simulator

The surface network simulator *Otter* simulates the flow of fluid through various parts of surface facilities. The basic strategy is simple: Fluid enters the system through wells or other sources. Various surface network objects are interlinked via connections to route this fluid through the network. Each object follows its own rules on how to process fluid from its input ports to its output ports. Some objects have complex behaviour; for example a power plant can be used to control the amount of fluid flowing through the system.

Mass and energy are conserved throughout the network unless a special object acts as an internal source or sink. Flow rates at a given time t make use of the properties $(w, q, p, h, \bar{X})$, where

- w is the fluid mass rate (always $+ve$ in the network)
- q is the energy rate - in general this corresponds to electric power
- p is the pressure
- h is the specificy enthalpy of the fluid
- $\bar{X}$ is the fluid component mass fraction vector

The pressure is matched in the system using an upstream-downstream-upstream approach. Upstream objects are first set up in such manner that they keep the system pressure as high as possible. Once the fluid has been routed through the system the algorithm turns around and allows objects to request higher or lower pressures from their upstream objects in order to match all pressures in the system. If a match can not be achieved then the current state is deemed an illegal state and errors will be presented to you.

In scenario mode wells can be throttled in automated mode by one or more power plants in order to match their required power output.

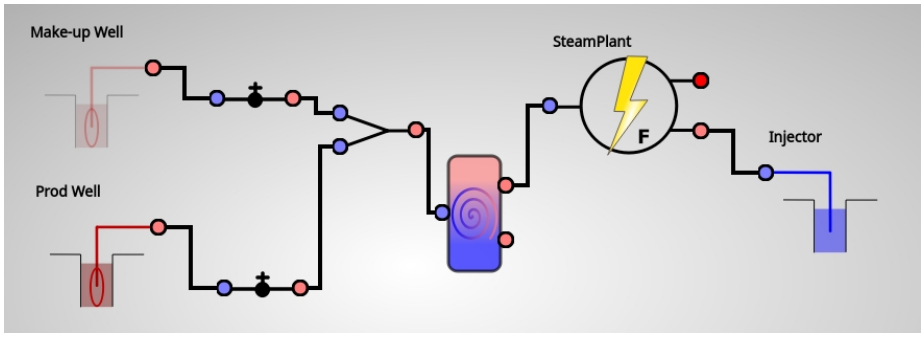

Figure 8.1: The Flow Network Editor.

There are three basic types of objects in a network:

- Port Objects have one or more input and/or output ports.
- Connections always connect the output port of one port object with the input port of another object.
- Data Labels can be used to quickly display some key flow parameters.

All objects are placed inside the Flow Network Editor which will show up when you double click on the *Flow Network* item in the tree navigator.

8.1 The Flow Network Editor

You can create new port objects in the editor by clicking on an object symbol in the editor's tool bar. The object will be placed in the centre of the editor and blink for a moment.

If you want to move an object to a better location you need to activate the move mode by clicking on the crossed arrow symbol. Once you have shifted the objects around click on the mouse symbol to go back into editing mode.

You can double-click on a port to start and finish connections. If the port already has a connection then double-clicking will detach the connection so you can re-attach it to another port. To delete a connection right-click on a port on which it begins/end and select the delete option.

8.2 Simple Port Objects

In edit mode you can interact with port objects by double-clicking on them. This will bring up their property and results dialogs. Right-clicking on an object will give you the option to delete the object or to bring up its help dialog.

8.2.1 Surface Source

A surface source can be used to introduce fluid or heat into the network from a source which is not coupled to the reservoir. For example this could be used to simulate injection of river water into the system.

The source uses a fully defined rate table, i.e. you need to enter all of the fluid properties (t, w, q, p, h, X) into the table.

8.2.2 Producer

A producer is a well which uses fixed fraction feedzone allocations. You can set its flow rate and wellhead pressure over time using a flow rate table; the wellhead pressure is required only to balance the pressure constraints in the surface network.

Use the welltrack table to define the producer's location in space. The feedzone table allows you to define one or more feedzones. You can use either true elevation or measured depth to define feedzone locations. The feedzone fractions must add up to 1.0.

Note that a producer with an empty flow rate table can also be used to easily query reservoir conditions along the welltrack since these form part of its output.

8.2.3 Regin Producer

A *Regin* Producer is a fully wellbore-modelled well. It will query feedzone and reservoir conditions along its welltrack and compute its deliverability curve using the *Regin* wellbore simulator module. Details for the wellbore simulator can be found in section 9.1. All the results from the wellbore simulation will be present giving you a complete picture of the well status.

You have the option to declare this object to be a makeup well. In this case the well will only be brought only if a power plant in the surface network requires more fluid and this well is in its makeup list.

The flow rate table allows you to specify a mass rate or wellhead pressure over time. The flow control options for the natural state, calibration and scenario

periods allow you to select if the well is operated at given mass rate or wellhead pressure. Note that linear interpolation is used between the points given by the deliverability curve so small deviations from the targets may be possible.

If your specified flow rate is larger than the output curve can deliver then it will be reduced to the maximum discharge rate from the output curve. If your specified wellhead pressure exceeds the maximum discharge pressure of the well then it will be shut in.

During the scenario period you have the additional option to use the well as a throttle, i.e. to allow the simulator to set the mass rate from the well to meet the overall generation requirements. The throttling range can be reduced to honour the minimum pressure (`Throttle (p min)`) or the maximum mass rate (`Throttle (w max)`) given in the flow rate table.

Sometimes it is necessary in surface network scenario simulations to completely shut in wells with low pressure in order to obtain a pressure balance in the system. You can do so by using a non-zero shut probability. Note that if multiple poor producers are present which flow into the same union then you may need to increase the shut probability in order to make it more likely that all of these wells are shut in for a given trial solution.

8.2.4 Regin Surface Spring

A *Regin* surface spring is a realistic representation of a geothermal surface feature. Internally it is based on a wellbore model and hence realistically represents flow towards the surface, including boiling and friction. The internal wellbore model is set so it will flow against a surface pressure as set by the ambient conditions.

The surface spring has three basic modes of operation. It can be set to use a mass flow rate from a table; in this case it acts as a simple sink to the geothermal model and no wellbore modelling is performed. This is a good mode of operation for the natural state period, when the flow from the spring is known.

In wellbore modelled mode the spring can uses the simplified Darcy-Forchheimer equation (see 9.3). You can specify the permeability-height product ($k \cdot h$) manually. Alternatively $k \cdot h$ can be automatically calibrated. In this case at the first time the calibrated value needs to be used it is determined by matching to the flow rate given in the table. If that flow rate is too high then the best estimate for $k \cdot h$ is used in the simulation forward.

8.2.5 Andvari Pipeline

This object aims to realistically model pressure drop and heat losses along a pipeline. This can become an important aspect of a simulation if the maximum

power of a plant is limited by the length and diameter of the pipes feeding fluid to it.

Note that including a pipeline model with pressure drop in your surface network simulation makes it much harder for the simulator to converge. You should hence only use these pipeline models if they are really required, and keep their number of nodes to a bare minimum. It is also a good idea to relax the input pressure requirements for unions into which pipelines feed (see section 8.2.10).

The general description of how to setup a pipeline model can be found in chapter 10.

8.2.6 Pumped Well

A pumped well is a single-feedzone liquid fluid producer which makes use of a pump to obtain its required surface pressure.

You will need to specify a flow rate table including the desired wellhead pressure. The well will not produce fluid if its pump model can not obtain the required wellhead pressure. Also it may reduce the desired flow rate if the pressure difference between reservoir and wellbore can not produce enough fluid given the selected pump/feedzone parameters.

The productive index (PI) of the feedzone is calculated as

$$PI = 2\pi \cdot \Upsilon \cdot (k \cdot h)/(\log(r_0/r_w) + S) \tag{8.1}$$

with the mobility Υ, the permeability-height product $k \cdot h$, wellbore radius r_0 and well skin S. The radius r_0 is calculated automatically from the geometry of the grid block which encompasses the feedzone.

If the required pressure difference between wellbore and reservoir is too high then boiling might occur. You can opt to adjust the flow rate in this case to prevent drawing two-phase fluid; a safety minimum pressure between the boiling point and the wellbore pressure can be specified.

The pressure gradient inside the wellbore is calculated using the density of the fluid in the wellbore at feedzone elevation. Friction is calculated using single-phase approximation and a rugosity parameter. The pressure at wellhead is then calculated using this single-valued pressure gradient; the pump may add the missing pressure difference towards the required wellhead pressure. Note that boiling could occur in the upper parts of the wellbore; you can set a maximum allowable gas saturation at wellhead elevation.

The pump can have a maximum pressure difference and a maximum volumetric flow rate. The object will also calculate the required pump power using the

following equation for the pump efficiency:

$$\eta = (A + B \cdot Q + C \cdot Q^2) \cdot (D + E \cdot dP + F \cdot dP^2) \quad (8.2)$$
$$P = -dP \cdot Q/\eta \quad (8.3)$$

where Q is the volumetric flow rate and dP the pressure differential supplied by the pump. The negative sign indicates that the power must be supplied to the pump.

8.2.7 Separator

A separator object can be used to split a fluid stream into a liquid and a gas stream. You can set its pressure over time using a table.

Note that for pressure balance the incoming fluid stream must be at a pressure equal or higher to the set pressure.

8.2.8 Valve

A valve is an object whose sole use is to reduce the pressure of a fluid stream. They are required in circumstances when it is necessary to combine multiple fluid streams with different pressures.

8.2.9 Pressure Controller

A pressure controller can be used to ensure a minimum or maximum pressure of a fluid stream. The pressure threshold can be varied over time.

8.2.10 Union

Unions are used to combine multiple fluid streams. The resulting fluid stream will have the lowest pressure of its input ports.

The pressure balance is achieved if all input ports have the same pressure; this can be achieved by placing valves upstream of the union. However this is often cumbersome from the point of view that additional objects need to be created. Also, when using pipes with dynamic pressure drops due to fluid composition, it can be quite hard for the flow network simulator to find a solution. In order to avoid these difficulties you can opt to relax the pressure match at the input ports;

in this case the output pressure is still the lowest pressure from the input ports but the input ports are allowed to differ from each other.

Note that if an input port has a zero mass flow rate then it will not contribute to the pressure balance.

8.2.11 Heat Exchanger

A heat exchanger is a passive object which attempts to heat a secondary fluid stream (which is not part of the surface model) by exchanging heat with the working fluid.

The maximum amount of heat harnessed requires thermal equilibrium between the working and the secondary fluid. You can set the temperature of this equilibrium using a table; the temperature for a given time is then linearly interpolated from this table.

Using the incoming fluid pressure and the set temperature the theoretical final enthalpy h_{final} of the fluid stream is calculated. The amount of heat harnessed is then calculated using the efficiency η as

$$\begin{aligned} \eta &= f \cdot \exp\left(-w/w_0\right) && (8.4)\\ h_{out} &= h_{in} - (h_{in} - h_{final}) \cdot \eta && (8.5)\\ Q &= w \cdot (h_{in} - h_{out}) && (8.6) \end{aligned}$$

with an efficiency factor f and a decay constant w_0 which models the decrease of efficiency with higher flow rates.

8.2.12 Splitter

Splitters have multiple output ports; the incoming fluid is distributed according to the split method.

A fraction splitter has a table containing a chronological order of split fractions. On the other hand a margin splitter has a chronological table of margins. The incoming fluid is compared to the margin of the first port; if the incoming mass rate is smaller than this margin then all the fluid is going towards the first port. If the incoming mass rate is larger than the margin then the first port receives the mass rate up to the margin; any excess is going towards the second port. In this fashion the incoming fluid is distributed until it is all allocated. The last output port does not have a margin associated with it and will take the remainder of the fluid. Note a margin splitter does not allocate dry heat or power q.

8.2.13 Injector

The injector object is built similar to the fixed fraction producer. It splits its flow rate into the feedzone using fixed fractions. However the source of its fluid stream can vary.

One way of defining the fluid stream is to provide data in a flow rate table. Since this is an injection fluid you need to provide enthalpy and component mass fractions as well.

The other way for defining the fluid stream is by using the input port of the injector, which makes sense if the fluid is coming from other surface network objects.

Quite often it makes sense to use both the flow rate table and the input port fluid at different stages of a simulation. For example you may want to use the flow rate table over the calibration period but then switch over to using the input port fluid for the scenario period. You can do this by selecting for which periods the flow rate table should be used, i.e. via the natural state, calibration and scenario period check boxes.

Note that the injector has a hidden output port, i.e. a port to which you can't connect other objects but for which results are displayed. This output port reflects the fluid stream which is currently used, i.e. it will either mirror the input port or the flow rate table rate.

8.3 Power Plant Objects

Power plants are objects in a surface network which allow some control over the amount of fluid produced and injected and when makeup wells should be brought online.

Your first choice is to make a plant passive. In this mode the power plant will generate power according to the amount of fluid passing through it and its own characteristic; however it will not try to alter the amount of fluid in order to optimize its output.

You can provide an electric power generation target in table form. The plant will try to match this power by throttling wells - note that of course you will need to set some wells in its flow path to be throttlers.

An objective function is used internally to achieve this target. If you have more than one power plant in the surface network they might be competing with each other for fluid and hence it may be impossible to achieve an overall generation target; in this case it can be advisable to give different weighting factors to the power plants in order to prioritize one over another.

You can use the relative target tolerance for determining if an optimization was successful; the higher the tolerance the faster the optimization will run so it is advisable to chose a reasonable tolerance here.

If an optimization run can not achieve the target then the plant which is not achieving its target can bring a makeup well online. You need to set these wells using an ordered list, i.e. the well in the list which is highest up will be brought online first. Only wells which you have marked as makeup wells will appear in this list.

Power plants in general have one output port associated with electric power; you can use this port to directly monitor their output. You can also make use of unions to combine electric power from different port objects to find out the net power generated by the surface network objects.

Above are common features of all power plants; the following sections will explain different power plant characteristics.

8.3.1 Generic Steam Turbine

The generic steam turbine generates electric power via a simplified conversion process. You need to specify its output pressure and enthalpy plus a turbine efficiency. The power generated is then

$$Q = \eta \cdot w \cdot (h_{in} - h_{out}) \tag{8.7}$$

Note that for a pressure balance the input pressure must be the same or larger than the set output pressure.

8.3.2 Mass Rate Controller

This object is not a power plant per se; it simply routes all incoming fluid straight to its single output port without changing fluid properties. However it contains the full functionality of a power plant to regulate the mass rate flowing through it by throttling producing wells or by bringing makeup wells online.

This object is useful for scenarios where the mass take from the reservoir is specified over time. It is still possible to add a passive power plant further downstream from it in order to measure the power generated given a certain mass rate in the system.

8.3.3 Ambient Conditions

Ambient condition settings can be used to link heat loss calculations in pipelines to a varying outside temperature. Other uses are to set ambient pressures, for example springs in the surface model use the ambient pressure as a constraint.

You can specify the ambient temperature or pressure in a chronological table; linear interpolation with constant end members will be used for determining the temperature during the simulation for a particular time. You can opt to only specify one annual cycle in the table, for example by giving average monthly temperatures. In this case the year you give is arbitrary.

8.4 Surface Network Optimization

The surface network is optimized by using a global optimization method and simultaneously achieving a pressure balance. If a pressure balance can not be found within the pressure tolerance set then the current state is declared illegal and another iteration with different free parameters is attempted. The result of the optimization process should be the best achievable target (like exact power plant generation selected) with a legal state (i.e. all pressures are matched within tolerance).

The following sections explain the available optimizer schemes and their associated parameters. You can switch between them by right-clicking on the *Optimizer* item.

8.4.1 Temperature Parallel Simulated Annealing with Adaptive Neighbourhood (TPSAAN)

This is an algorithm described by Miki et al. [2002]. It is very fast and efficient and can be thought as the working horse for surface network simulation.

The number of threads sets the number of parallel surface network simulations; each of them with their own fixed temperature. You should chose a multiple of the number of threads your computer can handle (i.e. four, eight etc); a good recommendation would be 32.

You can set the maximum number of iterations allowed; if your simulation struggles to find a solution you can increase this number. However if this happens it is also advisable to check if you can't decrease the number of free parameters, i.e. by decreasing the number of wells which can be throttled.

The swap parameter describes the interval at which solutions are swapped between the worker threads. The idea of the algorithm is that the better solutions are moved towards "colder" temperatures for fine tuning while poorer solutions are

moved to "hotter" temperatures to allow coarser searches through parameter space.

8.4.2 Simulated Annealing

This is a more basic sequential simulated annealing algorithm based on Ingber [2017]. It usually performs less efficient than TPSAAN but is available in *Volsung* for comparison purposes.

You can set the maximum number of iterations for this algorithm.

The number of restarts defines how often the algorithm will reset itself to the best solution found so far if it ceases to improve.

λ and τ are parameters required for finding the temperature of a particular iteration k:

$$T = T_0 \cdot \exp\left(-\lambda * k^{1/\tau} / N_{maxiter}\right) \tag{8.8}$$

8.4.3 Producers Common Scale Factor

This optimization algorithm is simpler than the previously described optimizers. It is tailored towards a widely used scenario in geothermal reservoir simulations where producing wells are set to a certain wellhead pressure. If the flow of these wells exceeds to demand of the plant then all wells are throttled back using a common scaling factor. On the other hand, if the flow at the given wellhead pressure can't meet the plant target then makeup wells are brought online.

To create a scenario like this you need to set the throttling behaviour of the wellbore modelled producers to `Throttle (p max)`. Only wells which are set to a throttle mode will be scaled back; the others will continue to flow as prescribed.

Note that this algorithm does not attempt to satisfy the pressure constraints in the system; you as a user need to set the target wellhead pressures so that they are above the separator pressure etc. The GUI will still display pressure match warnings (blinking objects) when displaying the results but apart from that the mismatches are ignored.

The Gudrun GUI for Wellbore Modelling

The *Gudrun* graphical user interface enables you to set up stand-alone wellbore simulations. These are useful if you want to study the deliverability/injectivity of a well in detail without having to deal with a complex coupling to a reservoir simulation.

Gudrun acts as a wrapper for the *Regin* wellbore simulation module. We will hence focus first on the description of the features available in the *Regin* module.

9.1 The Wellbore Simulator

The *Regin* module forms the computational basis for wellbore simulations inside *Volsung*. *Regin* is available in two forms: It forms the backbone for all computations inside *Gudrun* if a stand-alone wellbore simulator is required. Or it can be used in fully coupled mode as a well in a surface flownetwork (see section 8.2.3).

9.1.1 The Welltrack

The welltrack describes the geometry of the well in $3D$ space. It must contain at least two points with different z-coordinates. Non-unique z-coordinates will result in the elimination of some of them, so make sure you only enter unique z-coordinates.

The welltrack serves multiple purposes:

- In a coupled simulation it determines the reservoir grid blocks through which the well passes.

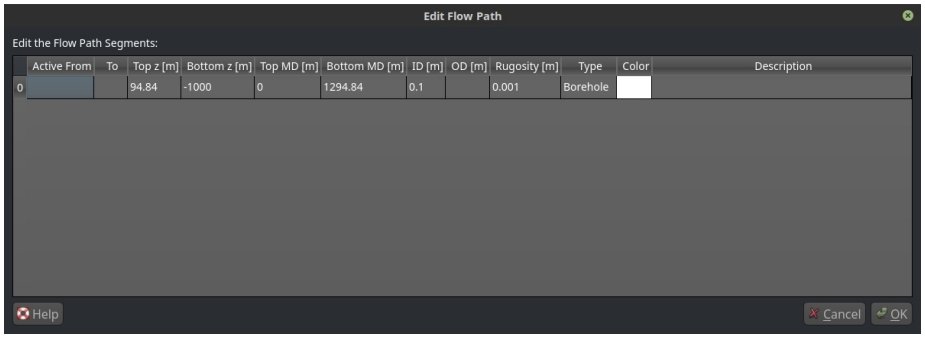

Figure 9.1: The Flow Path Dialog

- It is used to calculate the measured depth, i.e. the curved coordinate along the welltrack.
- Using elevation versus measured depth the wellbore simulator determines the inclination angle which is an important parameter for determining the hydrostatic head.

Entering the welltrack in $3D$ coordinates is the most versatile and error-proof option. However in stand-alone wellbore simulations it is sometimes desirable to enter the welltrack using $2D$ coordinates, for example measured depth and elevation. The welltrack dialog gives you this option for pasting data from the system clipboard; simply click on the "Paste from Clipboard" button and select which format the data in the clipboard has. An artificial $3D$ welltrack will be calculated which has the x-coordinate set up in a way to mimic the inclination of the clipboard data. However note that it is important that the clipboard data is numerically correct; for example if you use the measured depth/elevation format then the measured depth spacings must be equal or larger than the elevation spacings. Else the welltrack will contain *nan* entries. Small numerical truncation errors in the *mm* range will be tolerated.

9.1.2 The Flow Path

The flow path parametrizes the cross sectional area through which fluid can flow in the wellbore. The cross sectional area determines the velocity of the fluid and hence is an important parameter for friction pressure gradients.

The dialog for the flow path is shown in 9.1.

A flow path segment can be made active within a certain time period by using the "Active From" and "To" fields. These fields take times as inputs; leaving them blank will make the flow path segment active at all times. This feature is very useful if changes in the flow path configuration appear over time, like adding a

sleeve, a fish or to describe scaling over time.

You can set the top and bottom coordinate of the flow path segment. You have the choice of entering either the elevation or measured depth columns; the corresponding alternative coordinate is automatically calculated using the current welltrack.

Inner, outer diameter and rugosity are required for determining the flow path characteristic. You can also give the segment a colour and an optional description.

The segment type is used to calculate representative flow path parameters when you have multiple segments overlapping, which is often the case in a well. For this purpose segment types are grouped into permeable and non-permeable types:

Type	**permeable**
Bore Hole	no
Casing	no
Fish	yes
Insulation	no
Pipe	no
Scale	yes
Cement	no
Sleeve	no
Slotted Liner	yes

The free cross-sectional area is determined as follows: First the largest wetted diameter for fluid flow is found by looking at the outer diameters of all permeable segments. Then it is determined if a non-permeable segment lies outside this diameter, in which case its inner diameter is used as the wetted diameter. The cross sectional area is calculated from the wetted diameter and is subsequently reduced by the material thicknesses of the nested inner segments.

The Reynolds length is approximated by using the maximum free length between the inner segments. If your segments form co-centric rings then this is the innermost diameter of the smallest segment.

The representative rugosity is determined as the maximum rugosity of all segments which contribute to the cross sectional area above.

The overall heat transfer coefficient is determined in a similar fashion. Here it is assumed that the void space between adjacent non-permeable outer segments is filled with formation material and the segments themselves are made from either steel or insulating material.

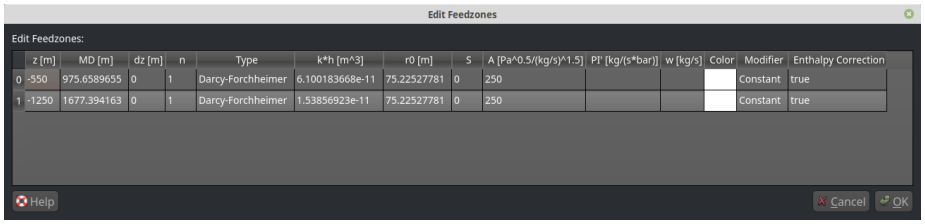

Figure 9.2: The Feedzones Dialog

9.1.3 Feedzones

The dialog for editing feedzones is shown in figure 9.2.

You can enter the feedzone base location either as elevation or measured depth; the current welltrack is used for the conversion between these two coordinates. If the simulation is coupled to a reservoir then the base elevation will be shifted to coincide with the centre elevation of the grid block the feedzone resides in. This shift is necessary since the pressure of the grid block is defined at its centre location; without the shift the well's deliverability/injectivity would be afflicted. Shifting the elevation is less error-prone then trying to calculate a hydrostatic pressure correction.

An elongated feedzone can be modelled by discretizing the single feedzone into smaller chunks (n) and spreading them out over a distance (dz). Each chunk will have $1/n$ times the overall feedzones productivity index or mass rate. Note that this feature usually only makes sense in stand-alone wellbore simulations since in coupled simulations all feedzone chunks would be mapped back to the single grid centre location, unless the elongated feedzone passes through multiple grid blocks.

The feedzone type determines the relationship of the draw/injection rate between the wellbore and the reservoir.

The *fixed* type feedzone will let you set the mass rate manually. Use $+ve$ for flow from the reservoir to the feedzone and $-ve$ for an outflow to the reservoir. Note that you can not use this feedzone type if you want to create general deliverability/injectivity curves for a well; however it is very useful for investigating PTS surveys.

A *lumped* type feedzone exchanges fluid according to the simple equation

$$w = PI \cdot (p_r - p_w) \tag{9.1}$$

with reservoir pressure p_r and wellbore pressure p_w.

The *Darcy-Forchheimer* type feedzone uses the equations

$$PI = 2\pi \cdot \Upsilon \cdot (k \cdot h)/(\log(r_0/r_w) + S) \cdot f_{EC} \tag{9.2}$$

$$p_r - p_w = \frac{w}{PI} + \frac{A \cdot w \cdot |w|}{\sqrt{PI}} \tag{9.3}$$

Here Υ is the total mobility (see equation 4.10). The permeability-height product $k \cdot h$ is sometimes also referred to as a productive index. r_w is the wellbore radius at the feedzone; r_0 needs to be manually entered in stand-alone wellbore simulations or is automatically approximated from the grid block geometry in a coupled simulation. S is the dimensionless skin factor. The Forchheimer factor A describes the quadratic term in the equation; in most cases it can be set to 0. The quadratic equation is solved for the flow rate w.

Alternative formulations for Darcy-Forchhheimer have been used by other wellbore simulators. Hence a version V2 of the above equations has been implemented to replace equation 9.3 using the mixture density ρ_f of the fluid:

$$p_r - p_w = \frac{w}{PI} + \frac{A \cdot w \cdot |w|}{\rho_f^2} \tag{9.4}$$

Because the Forchheimer term A is used in different equations you need to ensure that you set it in the correct SI units, which are $Pa^{1/2}/(kg/s)^{3/2}$ when using equation 9.3 or $Pa \cdot s^2/m^6$ when using equation 9.4.

The f_{EC} term is the optional enthalpy correction factor (ECF) which accounts for changes in fluid properties between the reservoir and the wellbore. It is calculated according to the equations given by Pasikki et al. [2017]; however note that their equations are only valid for pure water so f_{EC} should not be used with fluid containing other components.

Many other wellbore simulators define the PI' slightly different in order to eliminate the r_0 and S parameters. In *Volsung* this definition is called *Simplified Darcy-Forchheimer*, and the feedzone productivity is described by the $(k \cdot h)$ permeability-height product alone:

$$PI = \Upsilon \cdot (k \cdot h) \cdot f_{EC} \tag{9.5}$$

All other fluid properties like enthalpy and component mass fractions are determined as follows: If the well pressure is higher than the reservoir pressure then we have an outflow and the fluid properties from the well are used. Else we have an inflow to the well and the fluid properties are calculated by using the phase mobilities as weights.

Feedzone Modifiers

Feedzone modifiers are used to vary feedzone behaviour beyond the scope of the simple equations above. The modification factors are multiplied onto either the productive index PI or - in the case of fixed mass rates - w.

The default is a *constant* modifier, i.e. $f = 1$.

A *chronological* feedzone modifier allows changing feedzone behaviour over time, i.e. $f = f(t)$. Linear interpolation is used on the specified table; the extrapolation behaviour is to used the table end members. If the table is empty then $f = 1$ is used.

Feedzone Mobility Pressure Correction (MPC)

The mobility term as calculated using equation 4.10 assumes that the variables defining the mobility do not change significantly between the reservoir and the inflow into the wellbore.

However this assumption may not be correct if the fluid transitions from single to two-phase conditions in the vicinity of the wellbore. In this case a pressure correction of the mobility can be useful.

If the mobility pressure correction is enabled then calculation of the total mobility for the purpose of calculating the feedzone inflow/outflow is replaced by:

$$\Upsilon = \frac{1.0}{p_r - p_w} \cdot \int_{p_w}^{p_r} \sum_{\beta} \Upsilon_\beta(p, h, \bar{X}) \cdot dp \tag{9.6}$$

In this case the relative permeabilities required for the calculation of $\Upsilon_\beta(p, h, \bar{X})$ are approximated by using the phase saturation instead.

Note that the pressure correction for the mobility term does not affect the determination of the flowing fluid composition for which equation 4.10 continues to be used in equations 4.11 and 4.12.

9.1.4 Heat Loss

The dialog for the heat loss parameters is shown in figure 9.3.

Heat loss to the formation can be switched on or off.

Particularly in low flow situations heat loss can lead to numerical instabilities due to overshooting the formation temperature. *Regin* has an advanced feature which

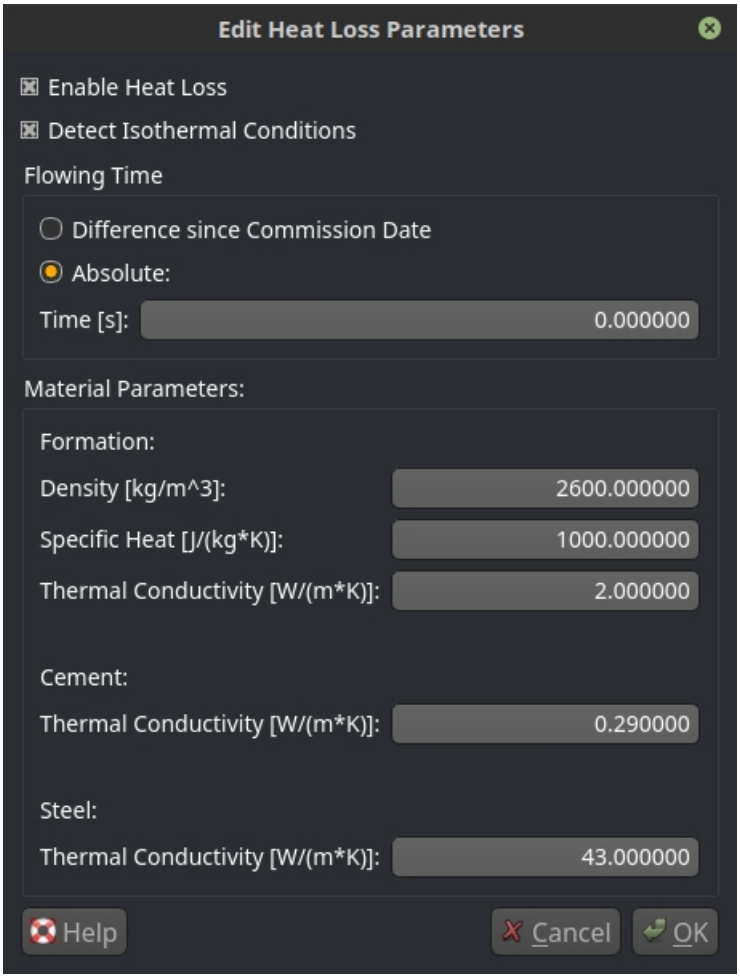

Figure 9.3: The heat loss parameters

will detect these situations and avoid temperature overshoots, hence extending the valid flow rate range. If detected then the wellbore fluid will attain isothermal conditions with the formation. However at really low flow rates even this detection mechanism may fail.

Once a well starts flowing it will exchange heat with its surroundings; this heat flow will decrease over time once the surroundings start to equilibrate with the wellbore temperature. The flowing time t_f can be set manually by the user or - in coupled simulations - automatically by taking the difference between commission time and current time.

The material parameters for the formation include the rock density ρ_r, specific heat c_r and thermal conductivity λ_r. Also required are the thermal conductivity of steel λ_s and cement λ_c.

To calculate the heat loss we first introduce the dimensionless time t_D:

$$t_D = \lambda_r \cdot t/(\rho_r \cdot c_r \cdot r_w^2) \tag{9.7}$$

We then introduce the dimensionless temperature T_D using an approximation given by Hasan and Kabir [2002]:

$$T_D = \ln\left[\exp^{-0.2t_D} + \left(1.5 + 0.3719 \cdot \exp^{-t_D}\right) \cdot \sqrt{t_D}\right] \tag{9.8}$$

The overall heat transfer coefficient U_{to} is then calculated using the follwing equation; this takes the thermal conductivity of steel and cement of the cased sections into account:

$$U_{to} = \left[\sum_i r_{to} \cdot \ln(r_{i,outer}/r_{i,inner})/\lambda_i\right]^{-1} \tag{9.9}$$

where r_{to} is the radius of the outermost cased flow path segment. The summation is over all concentric rings of the stacked outer segments; λ_i is either the thermal conductivity of steel or cement as appropriate for each summation index.

For a finite U_{to} we next calculate the linear heat transfer coefficient H_{tc}:

$$H_{tc} = -2\pi \cdot \frac{r_{to}U_{to}\lambda_r}{\lambda_r + r_{to}U_{to}T_D} \tag{9.10}$$

If U_{to} is infinite (i.e. the wellbore radius r_w equals r_{to}) then H_{tc} is calculated instead by

$$H_{tc} = -2\pi \cdot \lambda_r/T_D \tag{9.11}$$

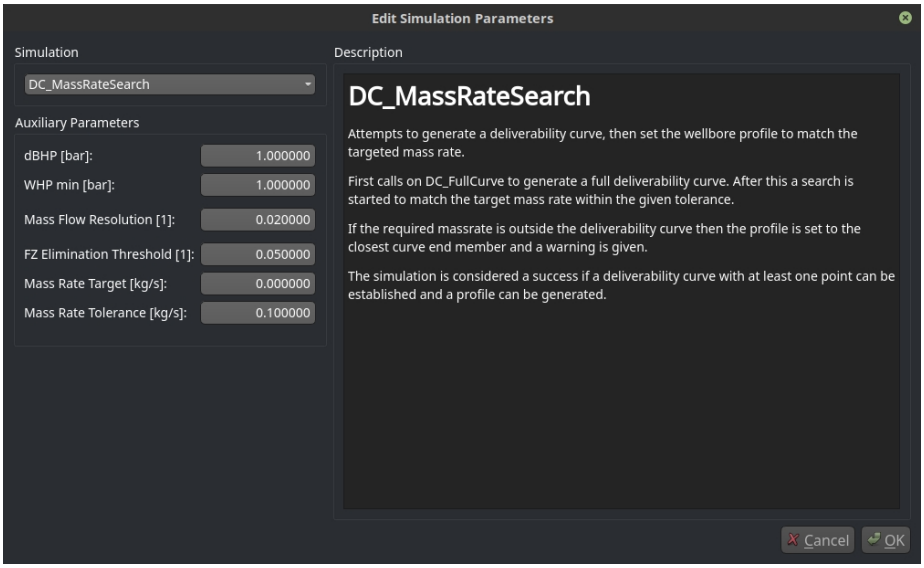

Figure 9.4: A sample dialog for simulation parameters

Finally the heat transfer to the formation per unit length is

$$\frac{dQ}{dL} = H_{tc} \cdot (T_w - T_r) \tag{9.12}$$

with the wellbore fluid temperature T_w and the reservoir temperature T_r.

9.1.5 Simulation Parameters

Figure 9.4 shows an example simulation parameters dialog.

The actual simulation parameters depend on the type of the simulation. For example, an "upwards" simulation requires an initial state and an initial elevation to start the simulation. A brief explanation is given to each simulation type directly in the dialog.

9.1.6 Auxiliary Parameters

The auxiliary parameter dialog is shown in figure 9.5.

The integrator defines the numerical integration scheme to solve the ordinary differential equations. You can select a simple Euler step method which is fast

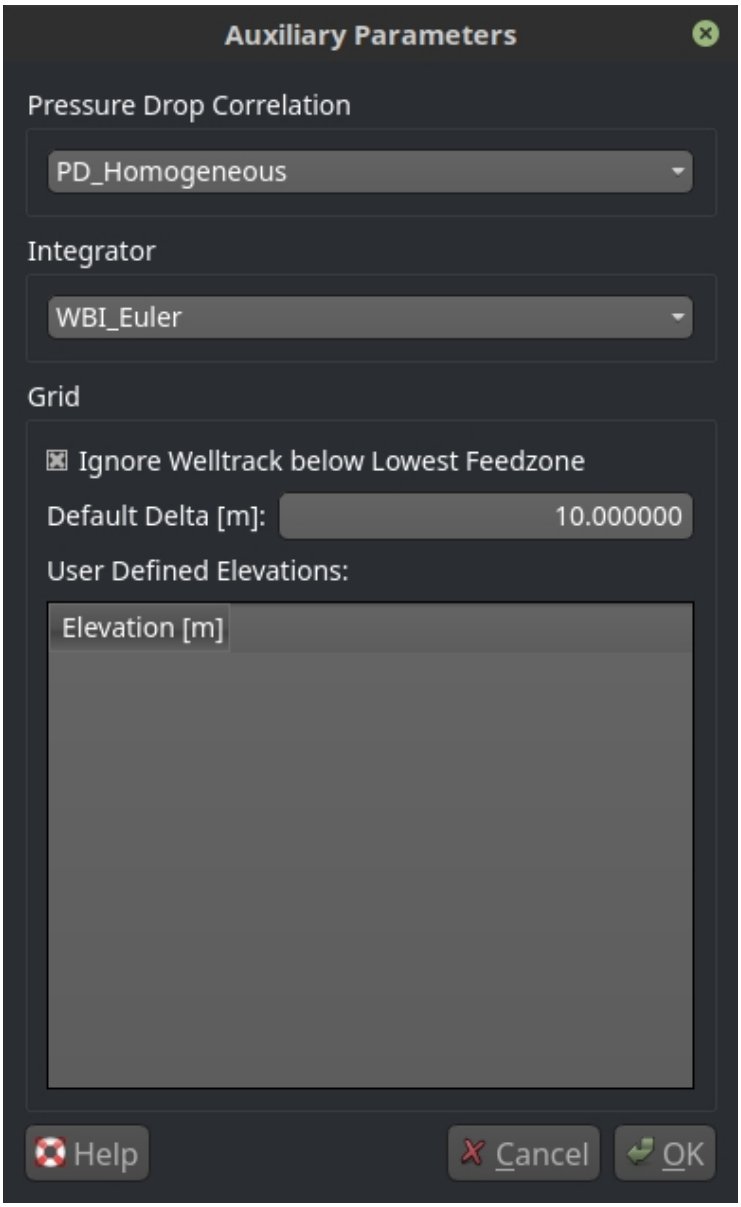

Figure 9.5: The auxiliary parameters for a wellbore simulation

and usually accurate enough. If you require a higher accuracy then you can select the slower Runge-Kutta method of fourth order.

The grid parameters aid you in defining the grid discretization. The default delta determines the basic grid spacing (in elevation coordinates). You can add more user-defined elevations if you require output at a particular location. Usually you want to ignore elevations below the lowest feedzone.

The pressure drop correlations are explained in section 9.2.2.

9.2 Wellbore Model Governing Equations

The wellbore simulator is based on the principles of conservation of mass, energy and momentum. Since mass is only exchanged at feedzones (see 9.1.3) we will focus here on the equations for momentum and energy conservation.

9.2.1 Coordinate System

We first need to introduce the coordinate system for a wellbore model. We will distinguish between the elevation which corresponds to the z-coordinate in the usual cartesian system and the measured depth (MD). The measured depth is the coordinate starting at the wellhead and follows the welltrack down. Due to the slope of the welltrack we have

$$dz = -dL \cdot \cos\theta \tag{9.13}$$

where dL is a segment along the welltrack (downward direction) and θ is the slope of the welltrack expressed as a zenith angle, i.e. zero for vertical and $\pi/2$ for horizontal sections.

The measured depth is then the integral

$$MD(z) = -\int_{z_0}^{z} \frac{1}{\cos\theta} \cdot dz' \tag{9.14}$$

where z_0 is the wellhead elevation.

9.2.2 Pressure Gradient

Conservation of momentum is described through the pressure gradient, which is the sum of the hydrostatic, friction and accelerational gradient:

$$\frac{dp}{dz} = \left(\frac{dp}{dz}\right)_{hs} + \left(\frac{dp}{dz}\right)_{f} + \left(\frac{dp}{dz}\right)_{acc} \tag{9.15}$$

The hydrostatic gradient is a function of the mixture density ρ_m and the gravitational acceleration g:

$$\left(\frac{dp}{dz}\right)_{hs} = -\rho_m \cdot g \tag{9.16}$$

The friction gradient is generally expressed as

$$\left(\frac{dp}{dz}\right)_{f} = -\frac{f_m \cdot |v| \cdot v \cdot \rho_m}{2D \cdot \cos\theta} \tag{9.17}$$

where f_m is the Moody friction factor, v is the fluid velocity along the wellbore ($+ve$ for upwards) and D is the flow path diameter.

The last part describes the change of momentum due to acceleration:

$$\left(\frac{dp}{dz}\right)_{acc} = -v \cdot \rho_m \cdot \frac{dv}{dz} \tag{9.18}$$

Pressure Drop Correlations

The above equations are well-defined when we're dealing with single phase fluid. However in two-phase conditions we face the problem that some of the entities are ill-defined, for example we don't have a single velocity but instead have a liquid and a gas velocity. Since there is no consistent model available for this case we use auxiliary pressure drop correlations which modify the above pressure gradient terms.

Pressure drop correlations are based either on empirical data or make use of "mechanistic" models which aim to describe the interaction between the liquid and the gas phase. Unfortunately comparisons between different pressure drop models will rarely find results in agreement due to the following issues:

- Empirical pressure drop correlations were established over a limited data range, for example a given working fluid, temperature/pressure ranges, elevation angles, bore diameters etc.
- Parametrization often follows simple approaches like polynomial fits rather than using an underlying physical model.

- The publications are sometimes very poorly written, making the implementation ambiguous, i.e. users may not agree on how to calculate entities.

- Some correlations are numerically unstable, i.e. may contain discontinuities etc.

- When comparing two different wellbore simulators other factors like a different approach for solving the differential equations, different thermodynamic tables etc will often lead to quite significant deviations for the same wellbore model.

Due to the above issues we feel necessary to express how much trust we put into the pressure drop correlations implemented in *Regin*.

The **Homogenous** pressure drop model assumes that liquid and gas phase move with the same velocity. This simplifies the treatment of the pressure drop gradients, however friction is usually underestimated since friction between the liquid and gas phase is neglected. The model is stable and is the default method for single-phase models.

Duns and Ros [1963] describe an empirical model. This correlation usually gives good results for geothermal wells; the model is quite stable and the authors have provided good methods to avoid discontinuities.

Hasan et al. [2009], use a mechanistic model. This model is usually in good agreement with the Duns and Ros [1963] and has the benefit that it does not make use of fitted data.

Hagedorn and Brown [1965] use an empirical model. Their model is quite stable. The authors note that their correlation often produces smaller friction gradients than found in other studies; this can be of interest for some geothermal wells if other correlations fail to model high maximum discharge pressures.

Hadgu [1989] uses the correlations described in Hadgu's PhD thesis.

The **Chisholm-Armand** and **Chisholm-Orkiszewski** correlations are an attempt to implement pressure drop correlations similar to the ones used in *GWELL* [Aunzo et al., 1991]. However it was found that the results produced by *Regin* can differ significantly from *GWELL* results. We decided to keep these correlations in the code for comparison purposes but do not recommend their use.

Orkiszewski [1967] use an empirical model derived on oil wells. We implemented this correlation for comparison purposes only and do not recommend their use since they are known to suffer from discontinuities.

The **Lockhart and Martinelli [1949]** equations are adequate for flow in horizontal or nearly horizontal pipelines. The equations given in Hasan and Kabir [2002] have been used for the friction factor. However the equation given by Hasan and Kabir

[2002] for the liquid holdup is erroneous and alternative equations have been used[1].

9.2.3 Specific Enthalpy Gradient

Conservation of energy in the wellbore simulation is achieved by the specific enthalpy gradient. Again this has three parts:

$$\frac{dh}{dz} = \left(\frac{dh}{dz}\right)_{pot} + \left(\frac{dh}{dz}\right)_{acc} + \left(\frac{dh}{dz}\right)_{hl} \tag{9.19}$$

The change in potential energy leads to

$$\left(\frac{dh}{dz}\right)_{pot} = -g \tag{9.20}$$

Acceleration of the fluid is described with the accelerational term:

$$\left(\frac{dh}{dz}\right)_{acc} = -v \cdot \frac{dv}{dz} \tag{9.21}$$

Heat exchange with the formation is described via

$$\left(\frac{dh}{dz}\right)_{hl} = \frac{1}{w} \cdot \frac{dQ}{dL} \cdot \frac{dL}{dz} \tag{9.22}$$

where dQ/dL is the heat loss per unit length as given in equation 9.12.

9.3 The Reservoir Thermodynamic State

In a fully coupled simulation the reservoir conditions - which are required for determining the fluid composition at the feedzones and the temperature for heat loss calculations along the wellbore - are taken automatically from the reservoir model. However in stand-alone mode you need to provide a profile for the thermodynamic state of the reservoir. Figure 9.6 shows the dialog for setting this one-dimensional reservoir profile.

In order to set up a reservoir profile you first need to define the method for setting the state. Your options will depend on the thermodynamic table which is currently used by the simulation. You will also need to set a relative permeability function

[1] https://www.sciencedirect.com/topics/engineering/liquid-holdup

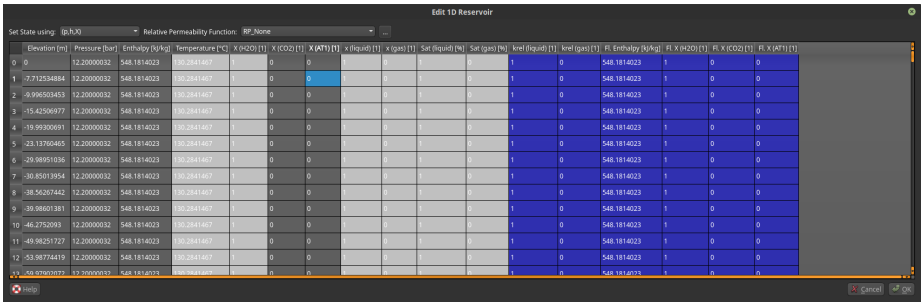

Figure 9.6: The 1*D* Reservoir Dialog

which calculates the flowing fluid properties from the static reservoir properties which you set. See section 5.1.1 for the different relative permeability function options.

You need to set the reservoir profile within the table of the dialog. Once you have entered the elevation and required primary variables for a state the table will display the most important secondary variables (like static gas mass fraction x and saturations). Further - in the right hand side of the table - you will see the corresponding flowing fluid properties.

You will need to enter thermodynamic reservoir profile conditions at least at the feedzone positions. The simulation will determine missing conditions for all elevations by using a linear interpolation over the variables $(p, h, \bar{X})$. A note of caution about this:

The linear interpolation uses the primary variables $(p, h, \bar{X})$ since this avoids abrupt phase changes which frequently occur if one would use $(p, T, \bar{X})$ instead. However most reservoir engineers are more used to providing pressure and temperature profiles. While you can use these in combination with the according setter tool you should carefully check the resulting thermodynamic conditions; it is quite easy to cross phase boundaries and the flowing reservoir state temperature as shown in the wellbore profile plots may have kinks in it. It is much better to focus on using the correct $(p, h, \bar{X})$ variables since this avoids common mistakes like having a pure steam zone wedged between liquid layers.

9.4 Miscellaneous

Gudrun allows you to enter some additional parameters and offers you some more tools.

9.4.1 Simulation Time

In a fully coupled simulation the current simulation time is automatically determined. But when running a stand-alone wellbore simulation you can set the time manually using the *Simulation/Time* menu. The simulation time is used to determine the current flowpath configuration and is also for heat loss calculations in fully coupled mode.

9.4.2 Field Data

You can enter field data for either the wellhead or wellbore conditions. This data is used for comparison in plots only.

The Sigrun GUI for Pipeline Modelling

The *Sigrun* graphical user interface enables you to set up stand-alone pipeline simulations. *Sigrun* acts as a wrapper for the *Andvari* pipeline simulation module. The mathematical and physical concepts used in this module are very similar to the ones used in *Regin*/*Gudrun*. We will hence only give a short overview over the differences between pipeline and wellbore simulations.

10.1 The Pipeline Simulator

The *Andvari* module forms the computational basis for pipeline simulations inside *Volsung*. *Andvari* is available in two forms: It forms the backbone for all computations inside *Sigrun* if a stand-alone pipeline simulator is required. Or it can be used in fully coupled mode as a well in a surface flownetwork (see section 8.2.5).

10.1.1 The Pipe Track

The pipeline track describes the path of the pipeline in $3D$ space. Unlike a welltrack it is not uniquely defined by its z-coordinates, hence an ordered track is required. You need to input the track using $3D$ coordinates; the first coordinate given will be assumed to be the fluid input side of the pipeline. The principal coordinate to describe parameters along the pipe track will be *measured distance (MD)* instead of the elevation coordinate used in wellbore simulations.

10.1.2 The Flow Path

The flow path parametrizes the cross sectional area through which fluid can flow in the wellbore. The cross sectional area determines the velocity of the fluid and hence is an important parameter for friction pressure gradients. Its definition is analogue to a wellbore flow path (see section 9.1.2); however only a limited choice of segment types is available for pipeline tracks.

10.1.3 Heat Loss

Heat loss calculations along the pipeline are very similar to the ones described in section 9.1.4. The main difference is that the outside temperature is not determined by a rock formation but instead by ambient conditions. You can opt to set a fixed temperature or link the outside temperature to an ambient condition setup (see 8.3.3).

Further the heat loss model does not need to take heat storage in the surroundings of the pipe into account. This simplifies the the calculation of the linear heat transfer coefficient:

$$H_{tc} = -2\pi \cdot r_{to} \cdot U_{to} \tag{10.1}$$

Note that U_{to} will be inf if the pipe wall thickness is zero. Hence check your flow path configuration if you encounter problems with large heat losses.

10.1.4 Auxiliary Parameters

The auxiliary parameters are similar to wellbore auxiliary parameters (see 9.1.6). However they make use of measured distances as base coordinate instead of elevation.

The angle tolerance parameter indicates where additional nodes should be inserted into the pipe profile when the pipe track changes.

The pressure drop correlations are explained in section 9.2.2. Note that since pipelines are generally horizontal and welltracks are mostly vertical you will be offered a different selection of pressure drop correlations here which will be more suitable for pipeline simulations.

The Swanhild GUI for Numerical Pressure Transient Analysis

Pressure Transient Analysis (PTA) is a powerful tool for inferring reservoir properties using data from a staged flow test. It has been used in the oil and gas industry for decades.

However due to the non-isothermal, multi-phase nature of geothermal reservoirs its usage by geothermal reservoir engineers has been limited. The main problem is that traditional PTA relies on analytical solutions to the pressure transient problem based on assumptions which do not hold in the geothermal context.

McLean and Zarrouk [2017], Zarrouk and McLean [2019] and McLean [2020] developed a numerical PTA framework which addresses many of the pitfalls of using analytical solutions. Their framework depends on numerically solving the pressure transient problem, allowing them to introduce non-isothermal conditions, channel/linear boundaries and fractional flow patterns.

Swanhild is a cut-down version of *Brynhild* which allows you to setup numerical PTA models according to the framework of McLean and Zarrouk [2017]. It also allows for inverse modelling, i.e. retrieving the underlying reservoir parameters from the flow test data. This is achieved by coupling it to the *PEST* inverse modelling package [Doherty, 2015].

While we have tried to make *Swanhild* as user friendly as possible we strongly recommend to familiarize yourself with the *PEST* manuals. *Swanhild* will find the solution to many PTA problems by using the default parameter settings; however you may need to seek guidance in the *PEST* manuals if the algorithms fail to find a solution. You can download these from `https://pesthomepage.org/documentation`.

We wish to thank Katie McLean for her help in developing *Swanhild*.

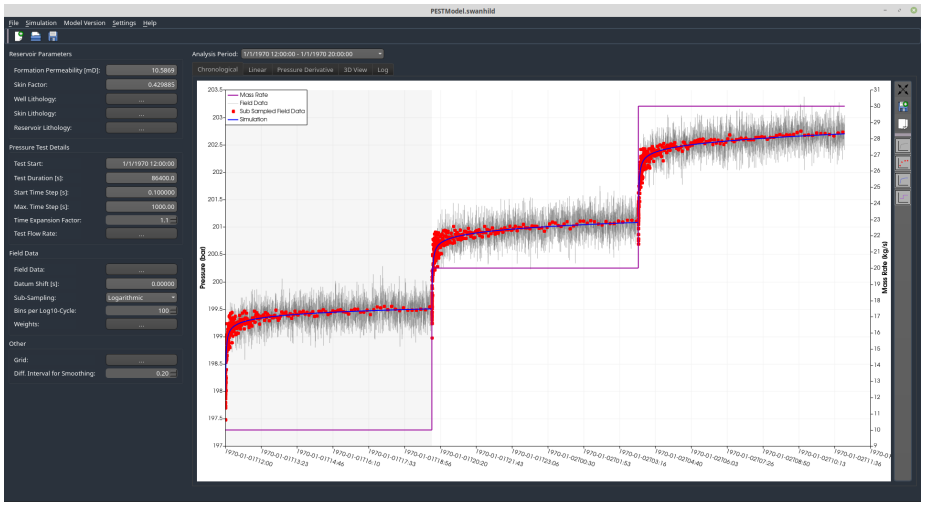

Figure 11.1: The *Swanhild* graphical user interface showing the chronological chart. Depicted are raw field data in grey, subsampled field data in red and simulated data in blue. The mass rate steps are shown in purple using the secondary axis.

11.1 Overview

The basic view of the *Swanhild* user interface is shown in figure 11.1.

The left panel gives options for changing the reservoir and flow test parameters. The right panel contains:

- The chronological plot shows the pressure and mass rate over time. This chart plots all data, regardless of the transient period it belongs to and hence gives a good overview over all data.
- The linear plot (figure 11.2) shows the delta-pressure as function of delta-time for a single transient period.
- The pressure derivative plot (figure 11.3) is a log-log plot of delta-pressure versus delta-time for a single transient period. It can display Bourdet derivatives with respect to either delta-time or superposition-time. This chart is useful for determining the base model and boundary types.
- The 3D view (figure 11.4) is a cut-down version of the reservoir view in *Brynhild*. It can be used to query thermodynamic conditions of all cells in the model. Note while the model is depicted as a radial slice it is still set up as a true radial model.
- The log view shows the simulation run log from the last model run. It can be used to identify errors, i.e. when the model failed to run due to out-of-range model parameters.

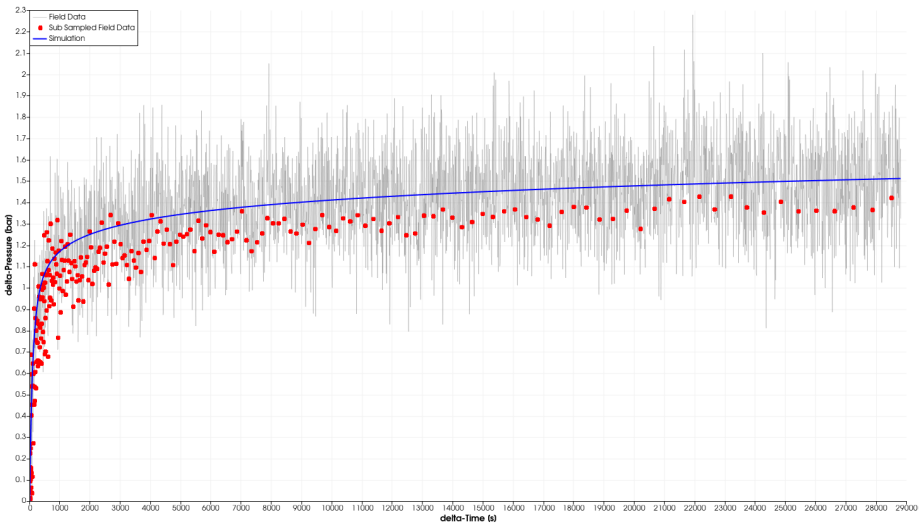

Figure 11.2: The linear plot depicting the delta-pressure as function of delta-time for a single transient.

11.1.1 Reservoir Parameters

The first group in the left hand side panel are the reservoir parameters. The scalar formation permeability is used as the permeability parameter in the outer reservoir section of the grid. The skin factor S links the permeability in the skin zone to the formation permeability via the relationship

$$\frac{k_{skin}}{k_{reservoir}} = \left(\frac{S}{\ln r_{skin}/r_{well}} + 1 \right)^{-1} \tag{11.1}$$

The scalar permeability of the well block is set to $k_{well} = k_{skin} \cdot 1000$.

The parameters for the remaining reservoir properties can be set by clicking the push button for the well, skin or reservoir lithology. Setting these parameters is analogue to setting them in the *Brynhild* GUI (see section 5). Note that the types of the material functions are fixed although you can still alter their parameters. The dialog also contains the settings for the initial state of the zone.

11.1.2 Pressure Test Details

The pressure test details section in the left panel contains the time control and mass rate settings. Test start and duration are self-explanatory. The simulation will start with the time step specified. Internal time step expansion is controlled using the standard settings from *Brynhild*; you can't alter these. However print times

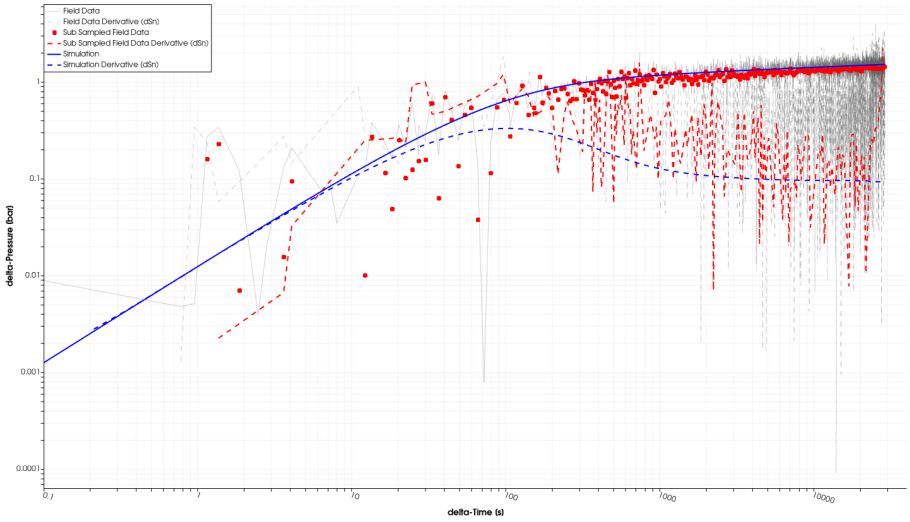

Figure 11.3: The log-log plot depicting the delta-pressure as function of delta-time as a log-log chart. Dashed lines show the Bourdet derivatives which can be used to investigate boundary and flow dimensionality types.

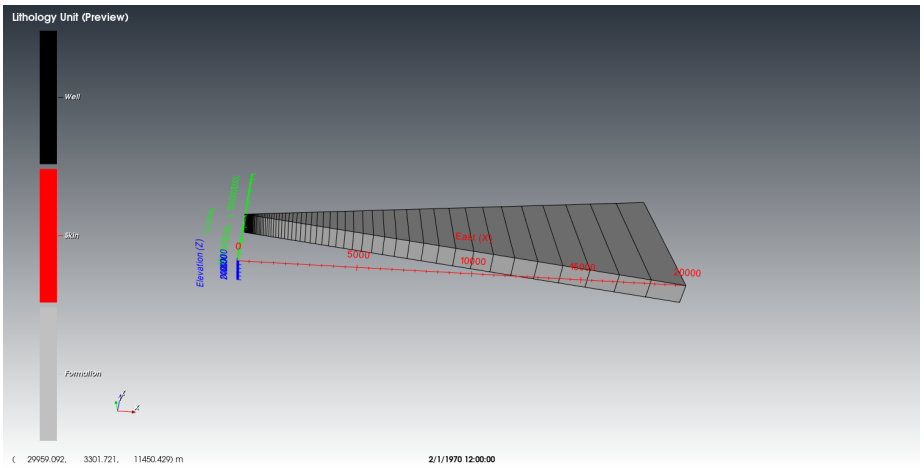

Figure 11.4: The 3D view depicting a radial slice of the PTA model.

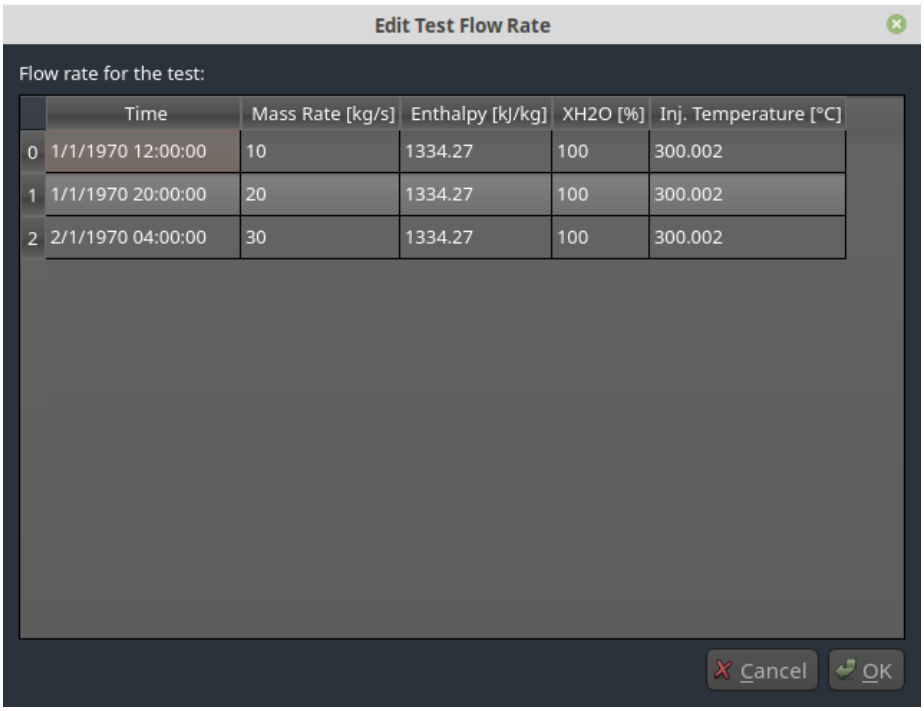

Figure 11.5: The flow rate table for the PTA test.

for data output are controlled via the time expansion factor and the maximum time step. This results in logarithmically spaced print times up to the maximum time step specified.

The test flow rate can be entered into a table, see figure 11.5. Injection tests must use a $+ve$ mass rate and specify the enthalpy and component mass fractions of the fluid. Production tests use a $-ve$ mass rate and ignore enthalpy and component mass fractions. The fluid temperature shown is indicative and calculated using the current pressure from the well lithology initial state settings.

Each change of mass rate in the flow rate table is interpreted as a unique *period*. At the beginning of each period the time step is reset to the start time step and the time control again begins the expansion process. The results are grouped by the same periods.

11.1.3 Field Data

It is useful for both manual and automatic calibration to plot the field data into the same charts as the simulation outputs. The data is entered into a table using time and pressure columns (see figure 11.6).

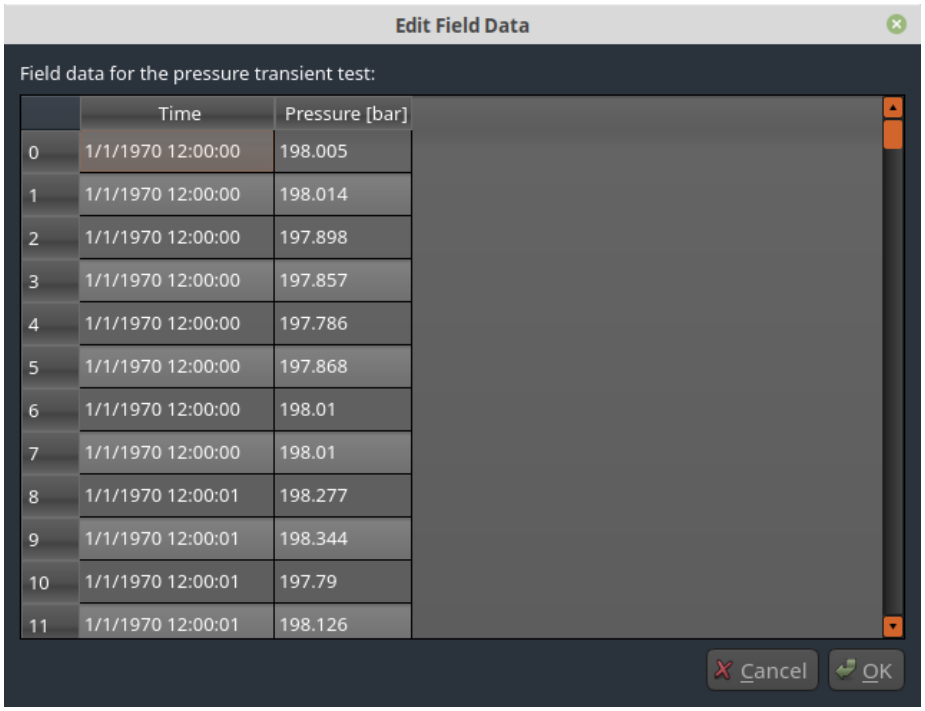

Figure 11.6: The pressure field data dialog

The datum shift correction is an important parameter which needs to be determined manually before calibration can be performed. It expresses the fact that there is a time lag between the mass rate change at the pump used in the test with respect to the time that the pressure at the gauge down the well starts changing. Adjust this parameter until the initial slope of the Bourdet derivative in the log-log chart approaches unity, i.e. overlaps with the initial delta-pressure signal. This can be hard to do if the data is noisy; nonetheless it is an important parameter to adjust before calibration can begin.

To smooth the noise in the field data it is subsampled. The default method is to use logarithmically spaced bins for each period. Each field data in a bin is averaged in both delta-pressure and delta-time to create the subsampled data set. This is usually a good method to use in inverse modelling to give more weight to early data. Data sampled with constant frequency would give more weight to later data and make it hard to estimate meaningful parameters (like skin factor, wellbore storage) which are only influential at early delta-times. However you can still do so by disabling the subsampling and provide your own tabulated weights.

11.1.4 Other

The remaining section in the left panel deals with setting up the grid and providing the differentiation interval for smoothing data. The later only affects plotting of the data in the charts, i.e. it does not affect inverse modelling.

The details for setting up the logarithmically spaced radial grid are given in section 3.4.2.

11.2 Running Forward and Inverse Modelling Simulations

You can run a forward simulation by using the *Simulation/Run* menu or by pressing `F5`. After the simulation has run the charts will update and you can inspect them to match the field data manually.

Once you have achieved a good initial fit to the data you can use *PEST* to automatically run an inverse modelling process to improve the fit to the test data and provide sensitivity analysis details.

Start the parameter estimation by using the *Simulation/Parameter Estimation* menu or by pressing `F9`. A dialog will appear from where you can add or edit free parameters and start the inverse model run.

11.2.1 Control Data

The control data tab mimics the control data section in a *PEST* input file. We have taken effort to re-organize this control data in a meaningful way where the *PEST* manual is sometimes hard to understand. When you hover with the mouse over the input fields the tool tip will show you which *PEST* control parameter the field corresponds to. Refer then to the *PEST* manual for more details on the particular setting.

11.2.2 Parameter Groups

The parameter groups section is pre-filled with one group per available parameter. You can use the data in this section to change behaviour of numerical differentiation.

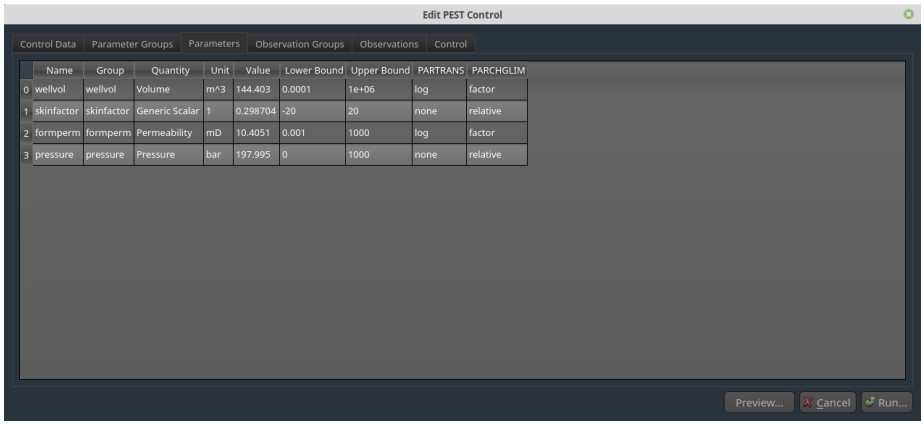

Figure 11.7: The parameter data tab for *PEST* inverse modelling

11.2.3 Parameters

The parameter data tab (figure 11.7) is the most important section of the dialog. Here you can add and edit the free parameters that *PEST* should use for inverse modelling.

Create a new line in the table then select the parameter name. The row will populate with sensible default values for that parameter. The value column will automatically be filled with the current value of that parameter. You must then select sensible lower and upper bounds which will meet your requirements.

Table 11.2.3 lists the available parameter choices.

Parameter Name	**Description**
formperm	Formation Permeability
skinfactor	Skin Factor
rsvrpor	Reservoir Porosity (Fracture)
skinpor	Skin Porosity (Fracture)
wellpor	Well Porosity (Fracture)
wellcomp	Well Compressibility (Fracture)
wellvol	Well Block Volume
pressure	Pressure (all zones and MINC layers)
enthalpy	Specific Enthalpy (all zones and MINC layers)
xX	Component Mass Fraction, e.g. xco2 for CO_2 mass fraction
fracdim	Fractional Dimension
bnddist	Boundary Distance
layerthickn	Layer Thickness

Table 11.1: Available parameters for inverse modelling

We have found it very important to **always** include the pressure as a free parameter

and to set its initial value close to the value observed in the field data. Failure to do so often results in *PEST* either failing find a proper solution to the inverse problem or displaying a large skew in the modelled pressure.

11.2.4 Observations and Observation Groups

The data in these two tabs will be generated automatically from the subsampled data and the optional tabulated weights.

11.2.5 Control

On the control tab you can select the *PEST* engine settings. You can run the inverse model in serial, in which case the normal `pest` executable is used for running the inverse model.

Alternatively you can opt to run the inverse model in parallel using `beopest`. This will speed up the runtime significantly, in particular if you have a modern CPU with multiple cores. You can select the maximum number of worker processes you want to run in parallel.

Note that *BEOPEST* requires the use of an open TCP/IP port which you need to select. The selected default port is commonly available on most machines; however it may be a good idea to check first if you can use that particular port on your machine. See section E.2.2 for details. You may also need to adjust your firewall settings to allow using this port.

Further note that *BEOPEST* may keep this port occupied for some time after finishing a run. For this reason you may need to wait about a minute before restarting an inverse model run.

11.2.6 Results

The parameter results from a successful inverse model run are shown in figure 11.8. The most important tabs are the *sensitivity* and *record* tabs. You can inspect these for more important details of the inverse model run.

All inverse model runs are performed in temporary directories in order to keep things tidy. If you wish to keep the information then you need to manually make a copy of the pilot directory *before* accepting the results. Once you have accepted the results the temporary directories will be automatically deleted and all *PEST* related data and information will be lost.

On accepting the results the program will run the forward model one more time

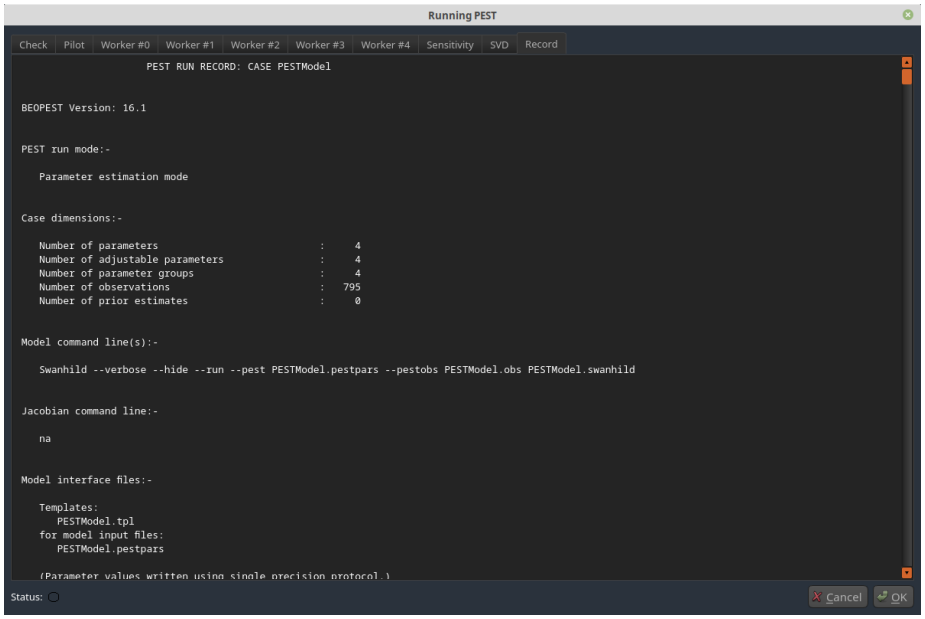

Figure 11.8: Results from an inverse model run

with the updated parameter results from the inverse model run. You should then see the final match to the field data of the pressure transient test.

Miscellaneous

12.1 Plots

On most of the plots you can hold the `Shift` key to disable zoom of the y-axis, or the `Ctrl` key to disable the x-axis zoom.

Data in the plots can be exported to a file by clicking on the export symbol and selecting the file format to export values. Alternatively the plot can be saved as a `png` file directly or copied to the clipboard so you can use it in reports.

12.2 Tools

12.2.1 Thermodynamic Table Tool

The thermodynamic table tool is a useful feature which allows you to calculate all thermodynamic properties for a given state. Figure 12.1 shows the dialog for the tool. Note this tool is also available as stand-alone application *Signy*.

You will need to decide on which set of primary variables you want to use with the tool; the available choice is dependent on the thermodynamic table in use.

After you have selected your state setter you can enter the corresponding variables into the dialog. The status LED will show green for a legal and red for an illegal state. All other secondary variables will be shown for a valid state.

Note that *Signy* can also be started from the command line, e.g. use

```
$ signy --help
```

for all available command line options. When using the `--command` option *Signy* can accept further inputs (like pressure, enthalpy) and print the thermodynamic

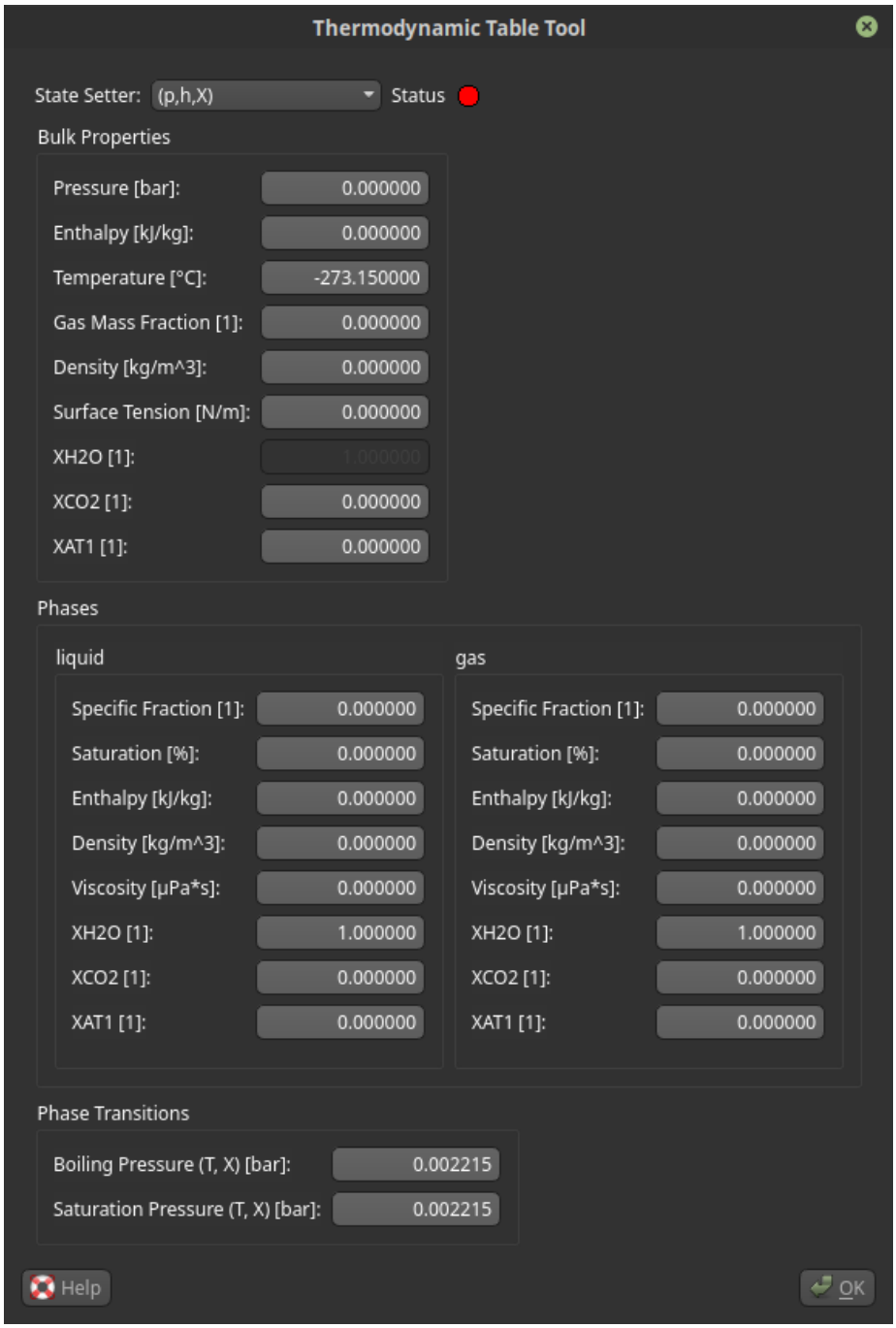

Figure 12.1: The Thermodynamic Table Tool

state to the standard output. There are also some *Python* wrappers available for *Signy* which allow you to use the thermodynamic table tool within the *Python* environment. The `examples` subfolder contains some *Python* scripts demonstrating the usage.

12.2.2 Coordinate Transformation Tool

You can use the coordination transformation tool to transform $3D$ coordinates using

1. a first translation
2. a set of rotations
3. a second translation

The rotation uses

1. a left-hand-rule rotation around the z-axis using an azimuth angle
2. a right-hand-rule rotation around the x-axis using an inclination angle
3. a second left-hand-rule rotation around the z-axis using a second azimuth angle

These series of transformation have been created for maximum flexibility. Quite often you will only need the first azimuth angle rotation; you can use the first and second translation to shift the point around which the rotation takes place if required. You can also use copy and paste or the import/export functionality in the tables to move data around.

12.2.3 Interpolation of Initial State Conditions

You can interpolate the initial conditions for a model using the results from a previous model version (the *source*). This can be useful when

- creating a small "sector" model from a full-field model
- faster running to steady state after re-gridding a model

To interpolate the initial state conditions you will need an existing copy of the source results, i.e. the `Results.sigurd` file.

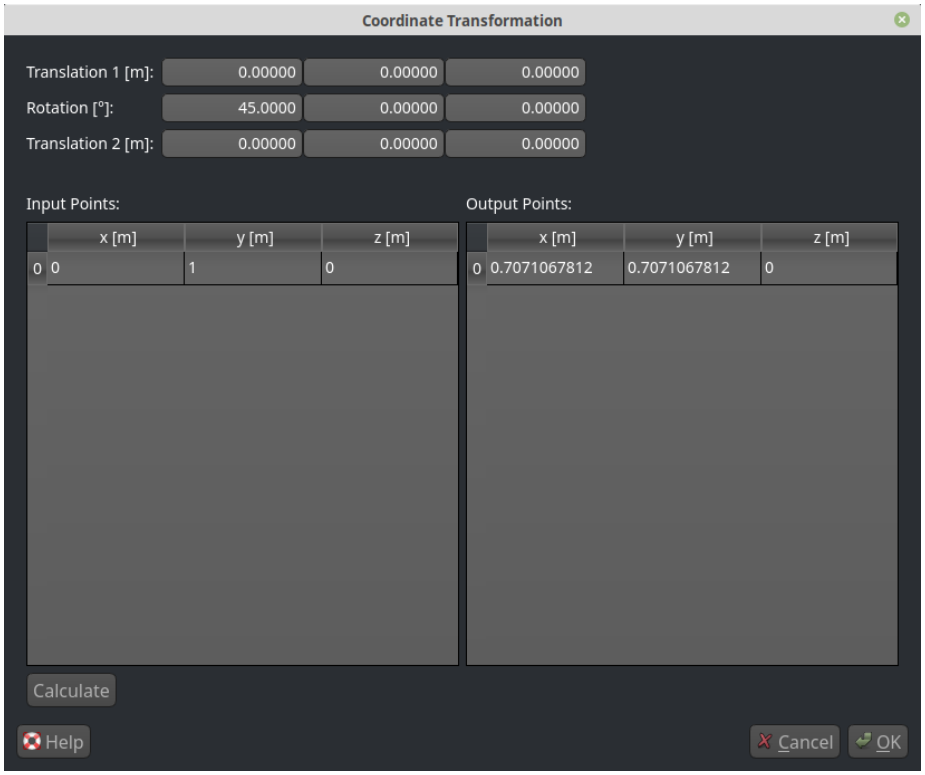

Figure 12.2: The Coordinate Transformation Tool, showing a simple rotation in the x/y plane using the first azimuth angle.

Prepare your new model as normal by creating a grid for it. Then run a simulation over it to create the reservoir output; it is sufficient to use the "test" run option for this step.

Next select the *Tools/Interpolate Initial State* menu to start the wizard guiding you through the process. You will need to select the source results file and the time in it from which the initial state is sourced.

You will need to select how many points (cell vertices from the source) are used for the linear interpolation. For recti-linear grids the default is 8 points. However if a grid with polyhedra is detected then the default choice is set to 12 to reflect the possibility that the 8 closest neighbours could all belong to the same face. Depending on your grid you can vary this parameter between 4 and 20 points.

Outliers are cells from the new model whose centres are not located within the boundaries of the source model. The default choice is to use extrapolation; however you can also opt to use the state from the current model.

If successful then the wizard will generate an `INCON.fafnir` file which you can use as an initial condition file in the simulation.

Bibliography

G Andersen, A Probst, L Murray, and S Butler. An Accurate PVT Model for Geothermal Fluids as Represented by H2O-CO2-NaCl Mixtures. In *Proceedings, Seventeenth Workshop on Geothermal Reservoir Engineering*, pages 239–248, Stanford, 1992. Stanford University.

Z.P. Aunzo, G. Bjornsson, and G.S. Bodvarsson. *Wellbore Models GWELL, GWNACL, and HOLA User's Guide*. LBL-31428, 1991.

Alfredo Battistelli. Improving the Treatment of Saline Brines in EWASG for the Simulation of Hydrothermal Systems. In *Proceedings, TOUGH Symposium 2012*, number 2007, Lawrence Berkeley National Laboratory, Berkeley, California, 2012.

Alfredo Battistelli, Claudio Calore, and Karsten Pruess. The Simulator TOUGH2/EWASG for Modelling Geothermal Reservoirs with Brines and Non-Condensible Gas. *Geothermics*, 26(4):437–464, 1997.

James Michael Chappell, Mark John Chappell, Azhar Iqbal, and Derek Abbot. The Gravity Field of a Cube. *Physics International*, 3(2):50–57, 2012. ISSN 1948-9803. doi: 10.3844/pisp.2012.50.57.

Yuguang Chen, Bradley T. Mallison, and Louis J. Durlofsky. Nonlinear two-point flux approximation for modeling full-tensor effects in subsurface flow simulations. *Computational Geosciences*, 12(3):317–335, 2008. ISSN 14200597. doi: 10.1007/s10596-007-9067-5.

John Doherty. *Calibration and Uncertainty Analysis for Complex Environmental Models*. Watermark Numerical Computing, Brisbane, Australia, 2015.

H. Duns and N.C.J. Ros. Vertical flow of gas and liquid mixtures in wells. In *6th World Petroleum Congress*, pages 451–465. World Petroleum Congress, 1963.

Teklu Hadgu. *Vertical Two-Phase Flow Studies and the Modelling of Flow in Geothermal Wells*. PhD thesis, University of Auckland, 1989.

Alton R Hagedorn and Kermit E Brown. Experimental Study of Pressure Gradients Occurring During Continuous Two-Phase Flow in Small-Diameter Vertical Conduits. *Journal of Petroleum Technology*, 17(4), 1965.

A R Hasan, C S Kabir, and X Wang. A Robust Steady-State Model for Flowing-Fluid Temperature in Complex Wells. *SPE Production & Operations*, (May), 2009.

A Rashid Hasan and C Shah Kabir. Modeling Two-Phase Fluid and Heat Flows in Geothermal Wells. *SPE*, 121351, 2009.

A.R. Hasan and C.S. Kabir. *Fluid Flow and Heat Transfer in Wellbores*. Society of Petroleum Engineers, 2002. ISBN 1-55563-094-4.

IAPWS. *The International Association for the Properties of Water and Steam - Release on the IAPWS Formulation 2008 for the Viscosity of Ordinary Water Substance*. Number September. 2008.

IAPWS-2014. *The International Association for the Properties of Water and Steam Revised Release on Surface Tension of Ordinary Water Substance*. Number June. IAPWS, 2014.

IAPWS-IF97. *The International Association for the Properties of Water and Steam: Revised Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam*. Number August 2007. Lucerne, Switzerland, 2007.

Lester Ingber. Adaptive Simulated Annealing (ASA), 2017. URL `https://www.ingber.com/{#}ASA`.

J.C. Jaeger, N.G.W. Cook, and R.W. Zimmermann. *Fundamentals of Rock Mechanics*. Blackwell Publishing, 4th edition, 2007.

E W Lemmon and R T Jacobsen. Viscosity and Thermal Conductivity Equations for Nitrogen , Oxygen , Argon, and Air. *International Journal of Thermophysics*, 25(1), 2004.

A. C. Liakopoulos. Darcy's coefficient of permeability as symmetric tensor of second rank. *International Association of Scientific Hydrology. Bulletin*, 10(3): 41–48, 1965. ISSN 00206024. doi: 10.1080/02626666509493405.

R.W. Lockhart and R.C. Martinelli. Proposed Correlation of Data For Isothermal Two-Phase Two-Component Flow in Horizontal Pipes. *Chem.Eng.Prog.*, 45: 39, 1949.

K McLean and Sadiq Zarrouk. Pressure transient analysis of geothermal wells: A framework for numerical modelling. *Renewable Energy*, 101:737–746, 2017.

Katie McLean. *Pressure Transient Analysis of Geothermal Wells : a Numerical Modelling Approach*. PhD thesis, 2020.

Mitsunori Miki, Tomoyuki Hiroyasu, Masayuki Kasai, Keiko Ono, and Takeshi Jitta. Temperature parallel simulated annealing with adaptive neighborhood for continuous optimization problem. *Second International Workshop on Intelligent Systems Design and Application*, pages 149–154, 2002. URL `http://dl.acm.org/citation.cfm?id=774964.774989`.

J. Orkiszewski. Predicting Two-Phase Pressure Drops in Vertical Pipe. *Journal of Petroleum Technology*, 19(6):829–838., 1967. ISSN 0149-2136. doi: 10.2118/1546-PA.

Riza G. Pasikki, Peter, and Frederick Libert. Impact of Enthalpy and Pressure Change to Geothermal Inflow Performance Relationship (IPR). In *The 5 th Indonesia International Geothermal Convention & Exhibition (IIGCE)*, 2017. ISBN 2449400818.

J.W. Pritchett, M.H. Rice, and T.D. Riney. Equation-of-State for Water-Carbon Dioxide Mixtures: Implications for Baca Reservoir. Technical report, DOE/ET/27163-8, 1981.

K. Pruess. *GMINC - A Mesh Generator for Flow Simulations in Fractured Reservoirs*. LBL-15227, 1983.

Karsten Pruess. *Brief Guide to the MINC-Method for Modeling Flow and Transport in Fractured Media*. LBL-32195, 1992.

Karsten Pruess, Curt Oldenburg, and George Moridis. TOUGH2 USER'S GUIDE, VERSION 2.0. *Contract*, (November):210, 1999.

Youcef Saad and Martin H Schultz. Gmres: a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*. *SIAM J. Scl. STAT. COMPUT. Vo—*, 7(3):856–869, 1986. ISSN 0196-5204. doi: 10.1137/0907058.

Bijendra Singh and D. Guptasarma. New method for fast computation of gravity and magnetic anomalies from arbitrary polyhedra. *Geophysics*, 66(2):521–526, mar 2001. ISSN 0016-8033. doi: 10.1190/1.1444942. URL `http://library.seg.org/doi/abs/10.1190/1.1444942`.

Phanish Suryanarayana, Phanisri P. Pratapa, and John E. Pask. Alternating Anderson–Richardson method: An efficient alternative to preconditioned Krylov methods for large, sparse linear systems. *Computer Physics Communications*, 234:278–285, 2019. ISSN 00104655. doi: 10.1016/j.cpc.2018.07.007. URL `https://doi.org/10.1016/j.cpc.2018.07.007`.

F.M. Sutton and A. McNabb. Boiling curves at Broadlands geothermal field, New Zealand. *New Zealand Journal of Science*, 20:333–337, 1977.

Thunderhead. *Understanding and Using MINC*. 1999. URL `https://www.thunderheadeng.com/files/com/petrasim/Understanding and Using MINC.pdf`.

H. A. van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J. Sci. Comput.*, 13(2):631–644, 1992.

A. Verma and K. Pruess. Thermohydrological conditions and silica redistribution near high-level nuclear wastes emplaced in saturated geological formations. *Journal of Geophysical Research*, 93(B2):1159, 1988. ISSN 0148-0227. doi: 10.1029/JB093iB02p01159.

E Warren and P.J. Root. The Behavior of Naturally Fractured Reservoirs. *Society of Petroleum Engineers Journal*, pages 245–255, 1963.

Sadiq J. Zarrouk and Katie McLean. *Geothermal Well Test Analysis*. Elsevier, 2019. ISBN 978-0-12-819266-5.

NVIDIA GPU Selection

A.1 General Considerations

Fafnir can use a Graphical Processing Unit (GPU) instead of a Central Processing Unit (CPU) for solving the linear system 4.41 using the iterative solver method. We need to have a brief look into why a GPU performs better than a CPU for large models in order to justify which GPU specifications are a priority when purchasing a GPU.

Fafnir uses a sparse matrix for storing the Jacobian matrix. An equal size matrix is used to store the ILU0 decomposition which is needed for the preconditioner in the iterative method. For rectilinear grids the byte size of the matrix can be estimated to be

$$\hat{S} \approx (6+1) \cdot N_e \cdot (N_c+1)^2 \cdot (B+4) \tag{A.1}$$

The number six corresponds to the average number of connections each cell has; it will hence be larger for a Voronoi grid and smaller if MINC layers are present. The size increases linearly with the number of computational elements N_e. The square of number of compenents plus the heat component further increases the requirements. Lastly, we need B bytes for storing a single matrix coefficient and four bytes for storing its column index in compressed-sparse-row (CSR) format. B can be either four bytes for single precision or eight bytes for double precision.

Assuming double precision, single porosity and using 2 components (say water and CO_2) then the total storage size for the matrix and the ILU0 preconditioner is

$$S \approx 1512 \cdot N_e \tag{A.2}$$

CPUs have a cache, i.e. some portion of memory which can be stored inside the CPU chip itself. The size of this cache depends on the CPU; low-spec CPUs often

have around $3MB$, mid-spec around $12MB$ and high-spec around $25MB$ cache. Using the above equation we can see that $25MB$ cache of a high-spec CPU can hold only a linear system of around 17000 elements; the lower-spec CPU with $3MB$ can only hold around 2000 elements.

Once the linear system is so large that it can't fit into the CPU's cache anymore then the whole system needs to be reloaded from the main RAM memory each iterative cycle of the linear solver. This is a comparatively slow process since it is now memory-bandwith limited. Here's where GPUs start to outshine CPUs: While the best CPUs currently have around $50-60GB/s$ memory bandwidth good GPUs have around $500-1000GB/s$ and hence can handle the time intensive linear solve operation much faster. The key selection criterion for purchasing a suitable GPU for *Fafnir* is hence its memory bandwidth.

A.2 Operating System Specific GPU Requirements

The GPU linear solvers in *Fafnir* are built on the *NVIDIA* CUDA programming interface. Hence you will need an *NVIDIA* GPU; *Fafnir* will not work on other GPUs from other manufacturers. *NVIDIA* has published a list of GPUs which are CUDA enabled on `https://developer.nvidia.com/cuda-gpus`. *Fafnir* will require a compute capability of *at least* 3.5.

Note when selecting a GPU you need to consider power and space requirements, i.e. you need to ensure that your desktop power supply unit (PSU) has sufficient capacity to support your GPU and that it will fit into your computer's casing. Different GPUs may need one or more power connectors to your PSU; sometimes you need to purchase adaptor cables separately, for example to change from an 8pin to a 6pin connector.

Laptops in general come with a fixed GPU, i.e. you can not upgrade your GPU at a later stage.

NVIDIA has three main streams for their GPUs: *Tesla* for high-performance computational applications, *Quadro* for professional visualizations and *GeForce* for gaming applications. The next sections will explain which GPU family to select for which operating system.

A.2.1 Linux

Fafnir does not make use the additional features built into the Tesla and Quadro streams and will perform very well on the much cheaper GeForce series. We have found no issues under *Linux* with the GeForce cards but note the next section for *Windows* if you want to dual boot into your computer. We recommend you purchase a GPU with a high memory bandwidth according to your available budget.

A.2.2 Windows

Under *Windows* GeForce and Quadro cards unfortunately suffer from a device driver penalty and driver instability issues. The reason lies in *NVIDIA*'s driver support for GeForce cards under *Windows*. More information on this issue can be found at:

`https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html#driver-model`

At the current time we can hence give the following advice:

- If you want to use a GeForce or Quadro card then run it under *Linux*; we have disabled use of GeForce/Quadro cards under *Windows* for the moment.
- You can also setup a PC as a remote using *Linux*, i.e. you can develop your geothermal models under *Windows* then run them remotely in the *Linux* machine. See appendix B for more details.
- If the above options are not suitable for you then the next best solution is to purchase a TITAN card since - according to *NVIDIA*- these can be run using the TCC driver model. However you will also need a second, inexpensive card to use with your display. Also prior to purchasing a TITAN card we would recommend you check with a qualified vendor that it really can run the TCC driver model.
- Last not least you can purchase a Tesla family card. They are more expensive than the GeForce cards but the difference in price may not be significant to your project. Again you will need a second, inexpensive card to connect your display to.

Running on a Remote

Volsung can be run remotely. Several options exist for setting up a remote environment but the most common options are a dedicated fast server or workstation located onsite or a cloud hosted environment such as an Amazon Webserver (AWS).

The requirements for the remote machine are:

- must have a compatible *Linux* operating system to run the *snap* version of *Volsung*
- it must be accessible via `SSH` and `SFTP` by the user running the *Brynhild* GUI
- `SSH/SFTP` identification must be via a private/public key pair setup

The following section will focus on the generic steps to set up a remote. This assumes you have already installed the *Linux* operating system on the machine. The later sections give brief details on how to set up a remote instance on cloud computing services.

B.1 Installation on Remotes

Install the compatible *Linux* operating system on the remote and attach the remote to the network. If this is a local machine you may need to install an additional package to connect via `SSH`. On cloud computing instances this package is usually already installed, else under *Ubuntu* use:

```
$ sudo apt install openssh-server
```

B.1.1 Identification

The first things you need to check now is if you can access the remote from the computer on which you're running *Volsung*. If you're running *Linux* on your local machine simply open a terminal and type

```
$ ssh user@host
```

where `user` is your username for accessing the remote and `host` is its host address or IP.

If you're running *Windows* on your local machine you need to establish the `SSH` connection using an adequate terminal program; we highly recommend the use of *MobaXTerm*[1].

If everything is ok then you will be asked for your password for login. If the connection has been successfully established you will now be logged into the remote and be able to give commands via the terminal. Quit the connection at this stage:

```
user@host $ exit
```

The next step is to generate a secure key pair which you will use in future instead of the password. This step may be different for cloud servers; you will need to follow the instructions given by the operator to obtain your key pair. The following instructions apply in the generic case, i.e. when you setting up your own remote.

Start by creating a key pair on your local machine:

```
$ ssh-keygen -t rsa -f myKeyName
```

Hit enter each time a passphrase is requested. The above command will create both a private key called `myKeyName` and a public key called `myKeyName.pub`. Save these files in a secure location.

Next we register and transfer the key pair to the remote:

```
$ ssh-copy-id -i path/to/myKeyName user@host
```

Now you should be able to log into the remote with the private key instead of using your password:

```
$ ssh -i path/to/myKeyName user@host
```

This time you should be logged into the remote without it requesting a password. You are now be set up to use this key pair with *Volsung*.

Note: While we here call the private key `myKeyName` it may be called differently when you get a key pair from a cloud server operator. Most often it will be called

[1] `https://mobaxterm.mobatek.net/download.html`

`myKeyName.pem` which is fine to use. However if you receive a key pair in another format (like `myKeyName.ppk`) you need to convert it to the required format first.

B.1.2 Installation

For the remainder of the installation instructions we assume that you are connected to the remote machine via SSH:

```
$ ssh -i path/to/myKeyName user@host
user@host $
```

Further you will need to run *Volsung* in headless mode using a virtual framebuffer. For this you need to install

```
user@host $ sudo apt install xvfb
```

If you want to run a *Volsung* application from the command line of the headless machine you first need to start a virtual frame buffer and associate it with a virtual display:

```
user@host $ Xvfb :99 & export DISPLAY=:99
```

You only need to issue the above command once per session; it won't matter if you issue it multiple times but it may return an error message.

Next follow the installation instructions in section 2.3.

Afterwards you can then test if you can run a *Volsung* application from the command line. The following should give you a normal response without error messages by showing the available options for *Gudrun*:

```
user@host $ volsung.gudrun --help
```

B.2 Run Models on the Remote

B.2.1 Remote Settings

In the *Brynhild* GUI use the *Settings/Remotes* menu to open the dialog for selecting the currently active remote. If your remote is new then you will have to add it in; see figure B.1 for our example.

The *SSH Port* is usually set to 22.

The *Log Port* is usually set to 2050. Note that on some cloud web servers you may need to open this port on the server's security settings for inbound connections.

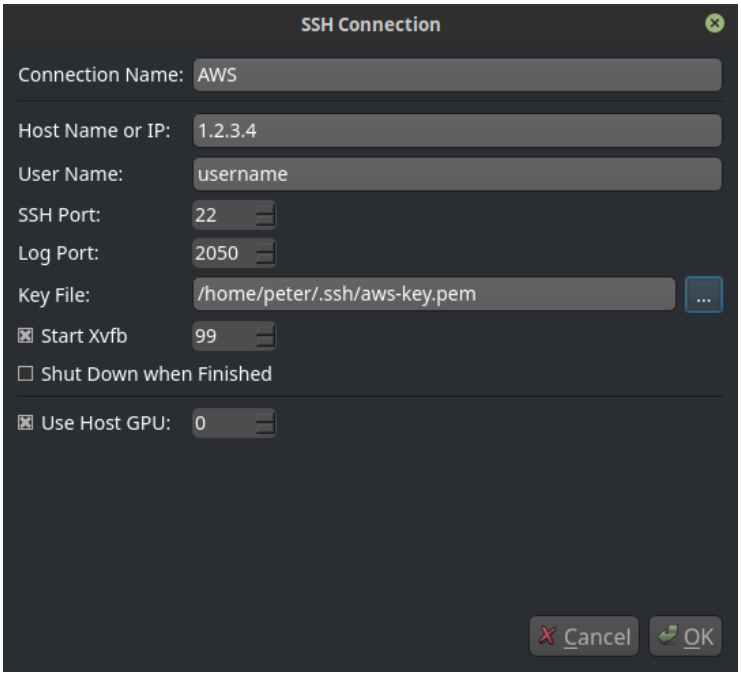

Figure B.1: The dialog for editing a remote's settings

The *Xvfb* settings relate to running a virtual frame buffer. This is required on cloud computing instances if they do not have displays attached.

You can opt to shut down the remote instance once the model run has finished. This can be useful for cloud servers which charge by the time used in order to minimize charge-out costs.

Note many systems only allow the superuser to perform shut downs. Hence to enable shut downs you need to modify the remote system settings so that any user can shut down the remote without requiring a password. To do so use

```
user@host$ sudo chmod a+x /sbin/shutdown
user@host$ sudo chmod a+s /sbin/shutdown
user@host$ sudo visudo
```

then add the following line to the bottom of the file and press `Ctrl+S` to save it followed by `Ctrol+X` for exiting the editor:

```
%group_name ALL=(ALL) NOPASSWD: /sbin/shutdown
```

You should then be able to test the shut down by issuing

```
user@host$ shutdown -h now
```

which now should not require you to enter a password.

B.2.2 Run

Once you have edited and set the active remote you should be able to run your model on the remote. Select the *Simulation/Run on Remote* or *Simulation/Run Test on Remote* menu. You should see the GUI

- connecting to the remote
- uploading the model specific files
- run the model
- retrieve the model output
- close the connection

B.2.3 Running on Multiple Remotes

You can use multiple remotes at the same time to run multiple models. This can be useful if you want to test different parameter ranges for a model. Instead of sequentially editing a parameter, running the model and then changing the parameter and re-running you can create multiple folders, each containing one instance of the model with a different parameter each.

You then start each model run on a different remote. After you have started the model run simply close the GUI or load the next model file. The model will keep running on the remote even if the GUI is closed. You can re-connect to the model at any time by opening the individual model files. Once the model is finished the GUI will download the results and mark the model run as finished.

Alternatively you can start all model runs then use the auxiliary *Odin* GUI to monitor all runs and download the data once finished.

B.3 Setting up Cloud Servers

B.3.1 Amazon Webserver (AWS)

Create a key pair as described in B.1.1 and store it in a safe location.

Create an account at `https://aws.amazon.com/`. Login and navigate to the EC2 Hosting Dashboard (Services:Compute:EC2). Set your desired region from the drop down menu in the top right hand corner of the screen (next to your name, US-West recommended for availability of GPU instances). Navigate to the

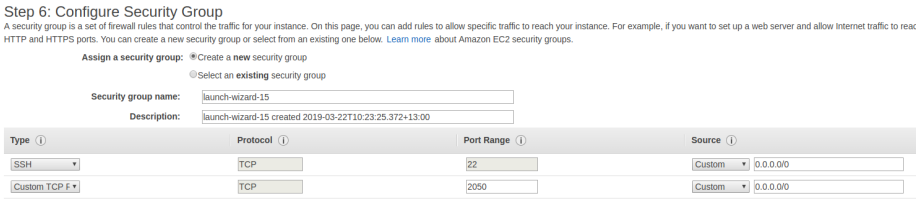

Figure B.2: Security settings to establish SSH access to remote server

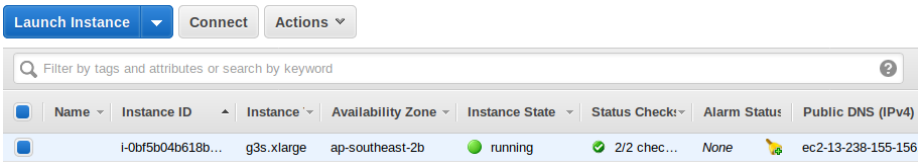

Figure B.3: Instance state

Network and Security: Key Pair. Select Import Key Pair and browse to your public key (*.pub) and select it to load it into the AWS system.

Click on "launch instance". Select the machine image you want (we recommend *Ubuntu Server 18.04 LTS (HVM), SSD Volume Type* 64-bit (x86)) and then Select GPU instances on the following page. We've trialled g3s, p2 and p3 instances but choose your option based on budget and workload. Security settings are established on the following page. Add a new rule so it looks like Figure B.2. We recommend access is restricted to specific IP addresses for added security.

Proceed through the steps to launch the instance. When launching the instance select *Choose an existing key pair* and select the key pair you created earlier. When you launch the instance you may get a screen saying failed because your service limit for this type of instance has been exceeded. If this happens you need to go to `http://aws.amazon.com/contact-us/ec2-request` request an increase in service limit for that particular type of GPU instance.

Once your instance is running you can navigate to the instance page of the EC2 Hosting Dashboard and you should see a list of your instances and information about their status (running, stopped etc) as in Figure B.3. You can stop and start your instance from this page.

The public IP of your instance can be found on the instance page of the EC2 Hosting Dashboard. Use this public IP as IP address for setting up the remote as described in B.1.2 (remember to reboot to install the *NVIDIA* driver) and section 3.6.

Note your public IP changes every time you restart an AWS instance; you will need to update the newest IP into *Volsung*'s remote settings in this case. Alternatively

you can setup *elastic IPs* for AWS.

B.3.2 Google Cloud Compute Engine

First create an account at `https://cloud.google.com/compute/`. You'll probably need to update your quota for GPU instances so go to *IAM and Admin* tab, then *Quotas* and search for *GPUs (all regions)* under Metric. Once there, if your quota is zero then select edit quota and request a quota increase. Once your quota increase is approved go to the VM Instance tab and select *Create Instance*. Under machine type click *customize* to get more options and select desired CPU and GPU options. We recommend 6 vCPUS, 8GB RAM and 1 GPU. Select machine type and OS (Ubuntu 18.04). See Figure B.4 for an example. Select create.

You can connect directly to this virtual machine by clicking on the *ssh* button. To enable terminal access you need to set up a ssh key pair. In your terminal enter

```
$ ssh-keygen -f mykeyname -t rsa
```

to generate a key pair. In the Google Compute Engine dashboard go to the metadata tab, select *SSH keys*, *Edit* and then add your new public key by copying and pasting into the box provided. Public key should be entered in the format *protocol public-key-data username@example* (in our case we simply opened the public key in a text editor and copied and pasted it directly).

You can now use you terminal to access the virtual machine via

```
$ ssh -i pathto/mykeyname username@externalip
```

To allow TCP on port 2050 for logging: Click on the 3 dots to right of instance: Network details: Firewall rules and select "Create new rule" to set up TCP access over port 2050. Check the list of firewall rules to make sure there is something like Figure B.5 in there.

The google cloud instance can be stopped and started from the VM Instance tab of the Google Cloud Platform website. All other steps to setup the remote are almost identical to setting up an AWS instance. SFTP to transfer the volsung installation file, SSH to install *Volsung* and run install scripts. Note the username for the ssh connection when using Google Compute Engine is your local username used when setting up the key pair (whereas for AWS is it *ubuntu*). A simple start and stop is usually sufficient to update the settings on the remote after making these changes.

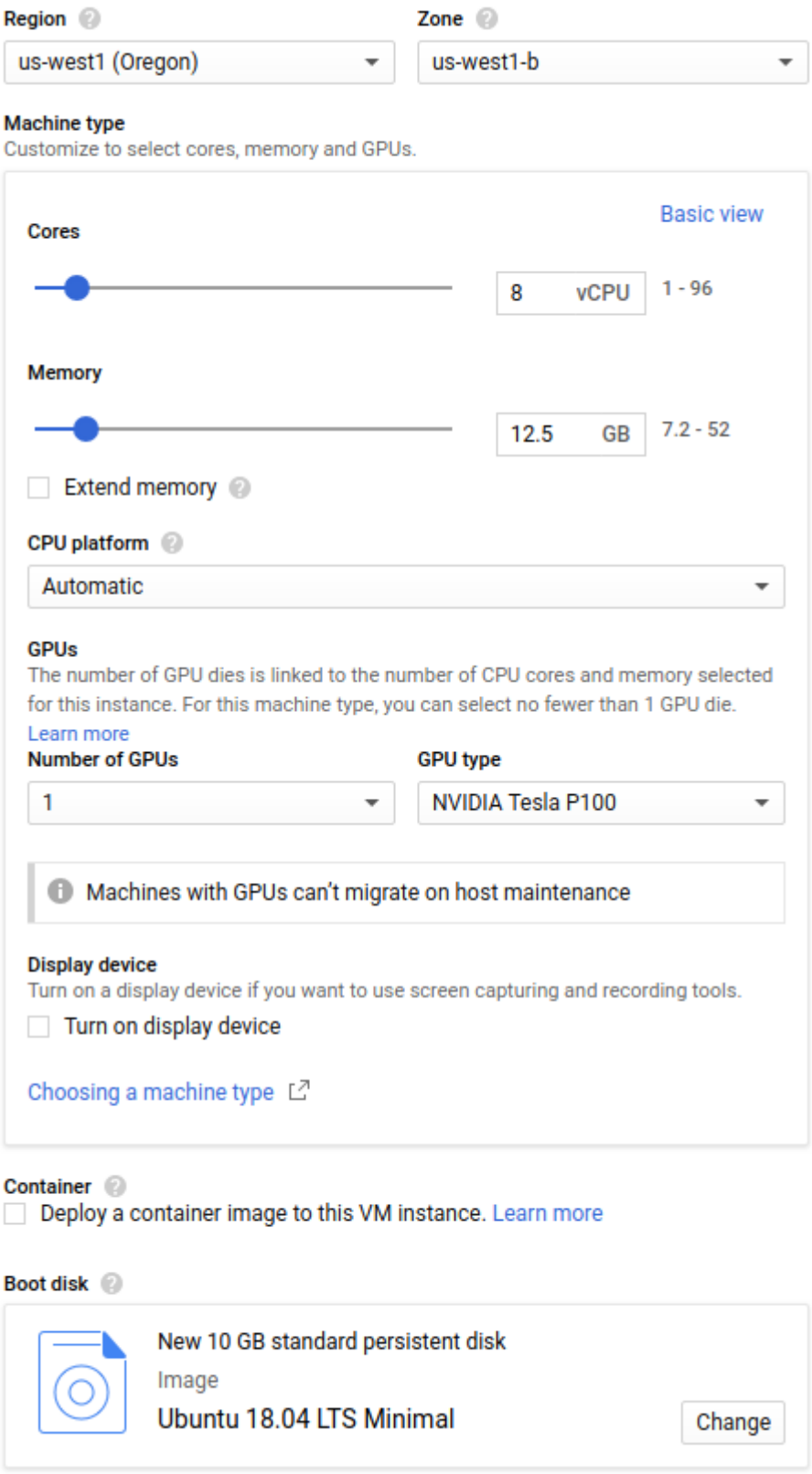

Figure B.4: Google cloud compute engine settings

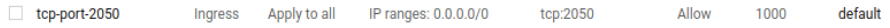

Figure B.5: TCP access over port 2050 on Google cloud compute engine

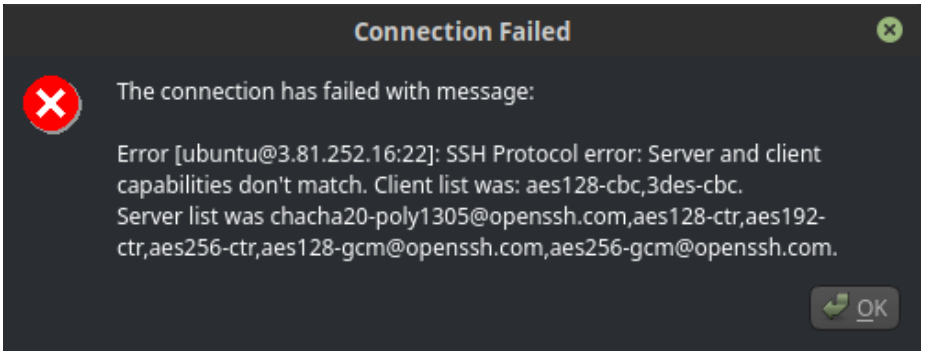

Figure B.6: Failed connection due to incorrect ciphers in ssh_config

B.4 Trouble shooting

Test your connection by selecting *Simulation/Run Test on Remote*. If the connection times out double check the server is running and IP addresses are correct. If you get a message like Figure B.6 it means there was trouble with the ciphers in the shh configuration. If this happens then double check you ran the setup scripts with *sudo* and if this doesn't help contact us for help.

If you run a simulation on a remote and you get a message indicating the solver is returning *nan* values then contact *Flow State Solutions Ltd* as there may be issues with some particular GPUs.

Running TOUGH2 Models

Volsung has the ability to run *TOUGH2* models through its *T2Fafnir* compatibility module. This is a very useful feature for

- making a 1:1 comparison between *TOUGH2* and *Fafnir* results
- *porting* a *TOUGH2* model into *Brynhild* and adding coupled wellbore/flow network simulation to it
- *converting* a *TOUGH2* model so it is completely transferred over to *Volsung*

When using an existing *TOUGH2* model you have two basic choices: You can run the model from the command line, or - if you have an appropriate grid file with the model geometry - you can import it into Brynhild and run it from there.

Volsung has some support for customized *TOUGH2* versions via *flavours*. For example the flavour `AUTOUGH2` can be used when you have an existing *TOUGH2* model using the *AUTOUGH2* simulator. Note however that we have only implemented flavour-specific parameters and functionality as required by some test models we have run so far; if you need more functionality or your own *TOUGH2* flavour please contact us so we can add these in.

A full *conversion* from *TOUGH2* to *Volsung* is possible if the model meets some basic prerequisites. Refer to section C.5 for more details.

C.1 Running from Command Line

When running from the command line the *T2Fafnir* module can be used through the *Sigurd* application. The following listing demonstrates the basic use of this command line tool in *Linux*; usage in *Windows* is similar:

```
# linux on remote: if no screen attached to system
```

```
# use virtual frame buffer
# else ignore this command
$ Xvfb :99 & export DISPLAY=:99

# show sigurd's options
$ volsung.sigurd --help

# a typical GPU run on a TOUGH2 input file T2Input.dat using EOS1:
$ volsung.sigurd --gpusolver --device-id 0 --tough2 EOS1 ...
                        --t2listing -i T2Input.dat -o Results.sigurd
```

C.2 Running using Brynhild

If you have a grid file associated with the *TOUGH2* model then you can simply set up the model in *Brynhild* by selecting the *File/New* menu and setting the new model to use *T2Fafnir*.

The *Brynhild* GUI will now contain a reduced set of options for editing the model.

In the navigator panel you will need to set *Grid/Grid File* to the grid file which must contain the reservoir geometry and the names for the cells; see section 3.4 for details.

In the navigators *Simulator* item you need to set the *TOUGH2 File* to your *TOUGH2* input file. You can opt to generate a *T2Listing* file which generates output in a very similar text format to standard *TOUGH2* output and can be read using *PyTOUGH*. Note however that the units for the physical quantities in this listing file are the custom units you have chosen for use in the GUI. Also note that the order of computational elements in this listing file is exactly as it would be when running *TOUGH2*; this makes a 1:1 comparison with *TOUGH2* results very easy.

Finally you will need to select the appropriate equation of state to use with this *TOUGH2* model using the *Simulation/Change Equation of State* menu.

C.3 Adding Wellbore/Flow Network Simulation

You can easily add wellbore and flow network simulations to the existing *TOUGH2* simulation using the flow network editor in *Brynhild*. However you need to keep in mind that the original *TOUGH2* model may already have contained your wells, hence you need to remove the corresponding *GENER* sections manually from the *TOUGH2* input file before using it in coupled simulation mode.

C.4 Testing Compatibility

If you have previous output from the *TOUGH2* simulation then it is a good idea to check that the results are reproduced accurately in *T2Fafnir*. You can use the *T2Listing* option to generate output files which are easily compared to the *TOUGH2* output. However there are a couple of considerations as to what kind of agreement between the two simulators can be expected.

C.4.1 Initial State Comparison

The first comparison to be made should be if the two simulators *T2Fafnir* and *TOUGH2* have loaded the same initial state. You can use the test run feature in *Volsung* to generate output for the initial state only; in *TOUGH2* you can run the simulation for a single, very short time step like $1s$. When comparing results from these two outputs you should see that - within acceptable tolerances - pressure, temperature, gas saturations and total (average) mass fractions of the components are the same between the two simulators.

One of the major issues which can arise at this stage is that an element is in the wrong phase. This can be due to the fact that *T2Fafnir* and *TOUGH2* make use of different thermodynamic tables; in particular the water vapour saturation curve can differ by a few millibar between the IAPWS-97IF used in *Volsung* and the IC67 formulation used by *TOUGH2*. If you plot a histogram showing the gas saturation differences between the two models you might see this as some elements differ by exactly 1.0. If you encounter such a case you need to manually adjust the pressure for the offending element by a few millibars ($0.1bar$ should do) so that it is recognized in the correct phase by *T2Fafnir*.

C.4.2 Transient Comparison

Once you have agreement for the initial state you can make a comparison between transient results from both simulators by running the model for the same time interval in both simulators while sources/sinks are active to modify the reservoir state.

The first thing you need to be aware off when making this type of comparison is that in order to obtain agreement between the two simulators they must have the same time-stepping behaviour. At the least you need to limit the time stepping to the smaller of time steps observed in one of the simulators. For example, if *TOUGH2* shows a maximum time step of $10^5 s$ and *Volsung* uses $10^6 s$ then you must restrict *Volsung*'s steps to $10^5 s$.

Depending on the simulation you may or may not obtain good agreement between the two simulators. The reason behind this the nature of the method for solving

the differential equations for the simulation combined with possible non-uniqueness of the solution.

When numerically solving the state at the next time step the simulator has a little room to wriggle around since we only require to conserve mass and energy within a (small) tolerance. The direction of how to change the primary variable vector is determined by the linear solver. Since the simulators make use of different solvers there will be small changes in the direction of the solution, and due to the acceptance tolerance the two simulators can diverge over in their solutions. This effect is compounded over time.

Another major issue often dominating why the two solutions may converge is the non-uniqueness of the solution which is mostly due to type of relative permeability curves used. Consider Grant's relative permeability function for in figure 5.3. You can see that towards the low/high saturation tails the relative permeability functions are constant, which means their derivative with respect to the saturation (which is a primary variable) is zero. This gives the iterative Newton method a lot of free room to move the phase saturation and hence divergence between the two simulations is high. You can often see in histograms of the difference in saturations between simulators that the maximum saturation differences reflect the width of these areas of constant relative permeability function.

So while due to the above mentioned effects it may not be possible to obtain direct agreement between the two simulators you can still test if the simulators are generally in good agreement by changing all all relative permeabilities to simple linear functions without constant plateaus, i.e. giving them an X shape. This will of course alter the original transient results but enables you to check if everything else in the model apart from the relative permeability functions is in good agreement between the two simulators.

C.4.3 Steady-State Comparison

Finally you may want to compare the steady states between *TOUGH2* and *T2Fafnir* by running both models for a very long time, i.e. until time steps typically get in the order of 10^15s.

If the model is single porosity you should expect to closely match pressure and temperature between the two simulators. Phase saturations can differ due and to the extent of the effects described in section C.4.2. Differences in mass component fractions are usually small and should be directly attributable to the differences in gas saturations.

When using MINC'd models with fractures and matrix elements present the above expectations should still be correct when looking at the fracture elements. However the matrix elements can show vastly different behaviour. They still need to show the same pressure and temperature as the fracture elements since they are in thermal equilibrium with them. However their phase saturations and mass frac-

tions can vastly differ, since once they have achieved pressure and temperature equilibrium with the fracture element there will be no more exchange of fluid or heat with the fracture. The matrix elements hence retain a memory effect of their initial state, and the comparison between the two simulators may show large differences due to both simulators using different pathways towards the steady state solution.

C.5 Converting a TOUGH2 Model to Volsung

Provided your *TOUGH2* model meets certain prerequisites it may be possible to completely convert it over to *Volsung*. A full conversion means that you can then fully edit and run the model from *Volsung*, i.e. you no longer need to deal with the *TOUGH2* input files and have all features in *Volsung* available to you.

Prerequisites are:

- You need a *Volsung* compatible grid which can link the *TOUGH2* element names to cells. You may be able to reconstruct this grid from the *TOUGH2* input file, see section 3.4 for options and details. If you don't have a compatible format you still may be able to create a *VTK* compatible grid using *Python*- please contact support for more details and help.
- The permeabilities in the `ROCKS` section must relate to a cartesian system, with `ISOT=1` and `ISOT=2` describing permeabilities in the x/y plane, and `ISOT=3` being the vertical permeability along in z direction. Rotations in the x/y plane are allowed as long as the rotation angle is common for all permeabilities in the `ROCKS` section.
- If the model is MINC'd then it must follow the standard *TOUGH2* MINC model. The input file `ELEME` section must only contain the fracture element names and MINC must be activated through the `MINC` generator in the `MESHM` section.
- Only equations of state supported by *Volsung* can be used for the conversion. Currently these are EOS1, EOS2, EOS3 and EWASG.
- *TOUGH2* sometimes allows for alternative ways of giving an initial state, for example by specifying temperature and gas saturation for EOS1. Most of these alternatives have not been implemented, so you should not rely on these to work properly in the conversion.

If your model meets these prerequisites then in *Brynhild* use *File/New* and select `Fafnir - TOUGH2 Import` from the list. An import wizard will help you through the conversion steps.

If the conversion is successful you can start editing and running the model from within *Brynhild*.

Note that it may be useful to delete some boundary conditions from the *TOUGH2* input file prior to importing the model and re-create these with more efficient ones. For example if you have a "rain" boundary condition in *TOUGH2* then you have a long list of `GENER` blocks, one per cell in the top model layer. It is better to delete these and replace them in *Volsung* with a source/sink boundary condition for the top layer. This avoids having thousands of individual sources/sinks present. Similarly it is advisable to replace "atmosphere" boundary conditions or fixed states by more appropriate polygon shapes.

If you have problems converting a model please contact support, we are glad to help.

Manifolds and Surface Shapes

The *Volsung* package makes use of *manifolds* (used in boundary conditions) and *surfaces* (used in geological layers) for describing geometric objects within the reservoir grid. Manifolds describe a topological space that locally resembles an Euclidean space near each point. Surfaces are hence also two dimensional manifolds and need no extra treatment.

We are interested in the following geometric primitives for describing manifolds:

- Points - $0D$
- Lines - $1D$
- Triangles - $2D$
- Tetrahedra - $3D$ (not used in *Volsung*)

Each manifold in *Volsung* can be described as a set of these primitives. Note that if a manifold contains primitives of more than one dimension than the highest dimension found will be used. More complex objects like polygons will always be triangulated internally.

Manifolds will work out their *interaction* with the grid. The interaction is specific to the dimension of the manifold, i.e.

- Points - number of points within a cell
- Lines - length of line segment within a cell $[m]$
- Triangles - area of triangle segment within a cell $[m^2]$

D.1 Manifold Types

Manifolds and surfaces can be introduced by the methods described below.

D.1.1 Loading from File

Manifolds can be loaded from a VTK file, i.e. either using unstructured or structured grids or poly data files. The advantage of using a file is that it can contain multiple shapes at the same time, i.e. it could be a collection of triangles or polygons.

Alternatively *AutoCAD* (DXF) files or *ESRI* shapefiles (SHP) can be used. These file formats make use of small converter tools supplied with the *Volsung* package. These converters may not work perfectly and may not fully support the DXF and SHP file formats; for example SHP files allow for holes in polygons which will not work with the converter. Also note that when using shapefiles you will need all 3 files associated with the shape, i.e. the files with the shp, shx and dbf file extensions.

When using file based manifolds you are usually given the option to apply a geometric transformation, i.e. rotation and translation.

D.1.2 Poly Vertex

Poly vertex manifolds are a collection of points. The total interaction is the number of points in the collection. The interaction with a cell is the number of points within the cell.

D.1.3 Vertex

Vertex manifolds are either empty or contain a single point. The total interaction is the number of points in the collection. The interaction with a cell is the number of points within the cell.

D.1.4 Poly Line

A poly line manifold contains a line which is made up from line segments, i.e. a connection between two points. The poly line can hence be described as an ordered list of points in space.

The total interaction is the total length of the lines in the collection. The interaction with a cell is the sum of the length of line segments within a cell. Line segments are internally subdivided when they cross a cell face.

D.1.5 Polygon

A polygon is an ordered list of points in space. It is implied that the polygon is closed by a connection between the last point with the first point in the list. Note that the *VTK* library that *Volsung* uses may have some issues with concave polygons, so it is best to avoid these and only use convex polygons whereever possible. Also note that you need to ensure polygons are simple, i.e. the sides should not cross each other.

The total interaction is the area of the polygon which is calculated by triangulation. Note that if a polygon is not flat then the calculated area will depend on the order of the points in the list.

The interaction with a cell is the area of the polygon located within a cell. The polygon is internally triangulated and broken into pieces along the faces of the grid.

D.1.6 Multiple Cells

A manifold may contain multiple polygons, lines or points. This is particularly of interest for creating complex surfaces in geological layers.

D.1.7 Delaunay 2D

The Delaunay triangulation can be used to create a two dimensional surface from an unordered point list. The algorithm uses a "best fit" plane to project the points on before performing the triangulation. After the triangulation has been established the points are transformed back into three dimensional space. This usually works quite well but in some occasions numerical artefacts can appear, so you should always check if the shape looks like you would expect it to.

D.2 Editing Manifolds

All manifolds which are internally created within *Brynhild* can be edited via mouse interaction inside the reservoir viewer. To access the edit mode right-click on the navigator item associated with the manifold and select *Edit Shape*, or right-click directly onto the manifold's shape in the viewer.

On the right-hand side of the viewer an additional tool bar pops up (see figure D.1); you can edit the shape with these tools.

The first four tools are for moving existing vertices. Editing in $3D$ can be quite

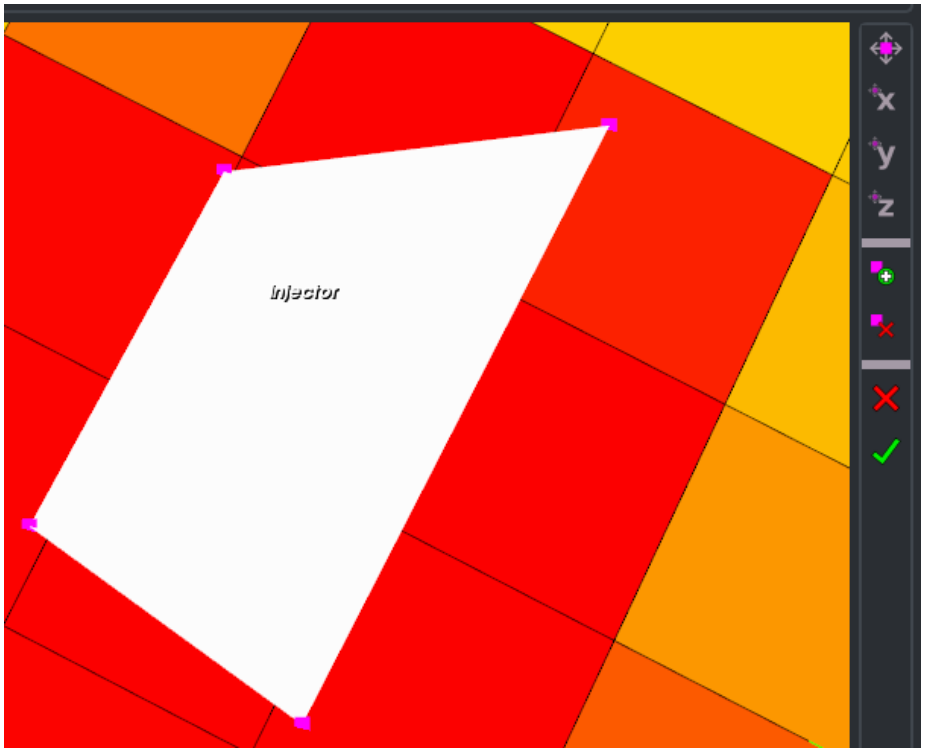

Figure D.1: The toolbar for editing manifolds/shapes.

hard so it is useful to restrict the movement of the vertices in to the desired directions only.

The next two tools are for adding or deleting vertices. The shapes will automatically adapt to the changed number of vertices. For polylines or polygons new vertices will be added at the beginning or end of their ordered point list, whichever one of these two points is closer to the new vertex location. To change this behaviour hold the `Shift` key when adding a new vertex; the new vertex will now be inserted between the two vertices forming the closest edge towards the new vertex location.

The *Add Shape* tool is only accessible if the manifold is of type *multiple cells* (see D.1.6). It starts the generation of a new shape. To add new vertices to an existing shape in this mode click on the existing shape, hold the `Ctrl` key then add new vertices.

Once you're finished you need to accept the changes by selecting the corresponding action in the toolbar. Alternatively you can discard all your changes.

Inverse Modelling and Uncertainty Analysis using PEST

Inverse modelling and predictive uncertainty analysis are powerful tools for the geothermal reservoir engineer. *Volsung* enables the use of these tools through its openness, i.e. you can easily modify input files and analyse output files. Hence it is possible to couple geothermal reservoir/wellbore/surface simulations in *Volsung* with the general-purpose inverse modelling/predictive uncertainty analysis package *PEST*.

PEST is an open-source, free software. It can be downloaded from `http://pesthomepage.org`, together with two very comprehensive user manuals. The theoretical background is described in Doherty [2015] which can be purchased on the *PEST* homepage.

We have compiled and bundled the *PEST* applications as necessary and have placed them with adequate installer tools on our webpage. You can download them and the manuals from `https://www.flowstatesolutions.co.nz/downloads`. Note these files are put there solely for your convenience, and we will not provide direct support for *PEST* specific problems other than demonstrating how to couple it with *Volsung*.

E.1 General Workflow for PEST

PEST has a very simple working structure:

- start with an existing model
- generate a *template* from the model input file to provide *parameter* placeholders
- provide *instruction* on how to extract *observations* from a model output file

- use a *control* file for describing the parameters, observations and the general task to fulfill

With the above in place,

- create a set of parameters
- generate a model input file by replacing the parameters in the template with actual values
- run the forward model over the input file
- analyse the model output file to obtain the observations
- use internal logic to generate the next parameter set
- repeat this process as often as required by the task

In the following sections we will demonstrate how to use *PEST* using the simple `columnPEST` example which you can find in `examples` subfolder of the *Volsung* installation.

For this problem with have a vertical column of 20 blocks. The top block is coupled to an "atmospheric" boundary condition - actually just an extra block to provide a constant pressure boundary. The bottom block contains a source with a constant flow rate of $1kg/s$ and enthalpy of $800kJ/kg$. The model is run for a long time to provide a steady-state. From this steady state we take the pressures in the columns as observations.

We want to treat this model as an inverse modelling task by making the enthalpy a free parameter. We hence use *PEST* to vary this parameter until it finds a match to the observed pressures in the column.

We start by copying the existing model into a fresh folder structure. *PEST* requires some additional files and will also generate a large number of output files; in general we want to keep these apart from our model's folder, so it is advisable to create a copy of the folder and name it like this:

```
columnPEST          # contains the original model
columnPEST-master   # folder for the inverse model setup
```

The reason for this structure and naming will become apparent later on. From this point onwards we assume that we will work in the `columnPEST-master` folder.

E.1.1 The PEST Template File

The model input file is called `columnPEST.brynhild`. Create a copy of it and name it `columnPEST.tpl`. This will become our template file.

Volsung input files are XML based files which can be edited by using any text editor. However editing is easier if you use a specialized editor like *XML Copy Editor*[1].

The first thing we need to do is to insert a new line at the beginning of the template file and inform *PEST* about the character which is used to mark up parameters. The *PEST* manual describes the allowed characters for this task; the most important criterion is that this character may not appear anywhere else in the template file except for marking up parameters. A good choice usually is the # symbol. Hence add the following first line to the template file:

```
ptf #
```

Next we need to find the location in the template file which describes the enthalpy of the source. XML uses a tag system, where a tag in general looks like this:

```
<tagname tagattribute="foo">text</tagname>
```

Note that *Volsung* rarely makes use of tag attributes; they are in general only required as meta information and you most likely do not need to alter any of them.

To find the enthalpy information use the editor's search function and search for the name of the source, i.e. "Bottom Source". You will find a section similar to the following:

```
<boundarycondition class="BCS_GridBottom" id="-1">
<enabled>1</enabled>
<name>Bottom Source</name>
...
<flowratetable class="FlowRateTable">
<tablerow>
<time>-1e+30</time>
<flowrate><p>0</p>
<h>800000</h>
<w>1</w>
<q>0</q>
<xh2o>1</xh2o>
...
</boundarycondition>
```

The <h> tag is the location we're after. Note that *Volsung* input and files **always** store values in SI units, in particular Pascals for pressure, Kelvin for temperature and Joule/kilogram for specific enthalpy. So when you're looking for values to replace keep in mind what they would look like when expressed in SI units and also that they might slightly differ from your expectation due to rounding and truncation errors.

[1] download from http://xml-copy-editor.sourceforge.net/ or *Ubuntu*: sudo apt install xmlcopyeditor

We now replace the value inside the <h> tag with the variable name ENTHALPY and the markup character:

```
<boundarycondition class="BCS_GridBottom" id="-1">
<enabled>1</enabled>
<name>Bottom Source</name>
...
<flowratetable class="FlowRateTable">
<tablerow>
<time>-1e+30</time>
<flowrate><p>0</p>
<h>#ENTHALPY                    #</h>
<w>1</w>
<q>0</q>
<xh2o>1</xh2o>
...
</boundarycondition>
```

Note that we insert additional spaces after the variable name; this tells *PEST* that it can use this additional space to write out more significant digits. Always ensure that the available space to write out parameters is approximately 20 characters wide, else *PEST* may reduce precision in order to match the available space.

Once the template file has been prepared you can check its integrity using

```
$ tempchek columPEST.tpl
```

Constrained Parameters

In some rare cases you need to deal with constrained parameters. The most common constrains are that component mass fractions and volume fractions for MINC need to add up to 1.0. So if for example you use X_{CO_2} as a parameter in *PEST* you will also need to replace the value $X_{H_2O} = 1.0 - X_{CO_2}$. Another example would be that you may want to constrain vertical permeability by being a fixed fraction larger than horizontal permeability, say $k_z = k_x \cdot 10$.

The second part of the *PEST* manual describes how to deal with these situations using the PAR2PAR auxiliary command. In principle you need to create a second template file which passes non-constrained parameters through and calculates the constrained parameters. Then you use this intermediate template file as a target in the control file instead of the actual model template file. In you model command script (see E.1.2) you will need to run the PAR2PAR command just prior to running the actual model.

E.1.2 The Model Command Script

Two requirements for running *PEST* are that the simulation must be able to be started from the command line, and also that the model output must be in `ASCII` file format from which *PEST* can analyse the observations. We solve these requirements by creating a *Python* script which handles both the running of the actual simulation as well as extracting data from the binary `hdf` output file from *Volsung*. This data is then written into an intermediate `ASCII` file.

For the next steps you need a working version of the *Python* modules supplied with *Volsung*. See appendix **??** if you haven't installed these yet.

The `runModel.py` script contains a first section describing some file names and the optional remote:

```
ModelFile = "columnPEST.brynhild"
ObsFile = "Observations.txt"
Remote = ""
```

The `ModelFile` contains the name of the model to run. In the *PEST* control file we will later on describe how this file is generated from the template we created before. The `ObsFile` is our intermediate file to which we want to export the model output values from the binary output file. Usage of remotes will be explained further below.

If you are using constrained parameters (see E.1.1) you will need to activate the `par2parFile` setting, else leave it blank:

```
par2parFile = ""
```

The next sections in `runModel.py` are generic and are used to remove any previous data if necessary and then run *Brynhild* in command line mode. Error handling ensures that the scripts aborts with a non-zero exit code; this exit code will trigger *PEST* to halt if necessary.

Note that you can re-use most of the `runModel.py` file for your own modelling purposes. However the last part of the script are specific to our model; in it we export the pressures of the blocks into the intermediate file:

```
model = VolsungModel("Results.sigurd")
output = open(ObsFile, "w")
# check that the end time was reached
if not model.reservoir.endTimeReached():
        print("Simulation did not reach end time; abort.")
        exit(-1)
# establish the pressure history and
# grab the pressure from the last print time
p = model.reservoir.history("/reservoir/Elements/Pressure")[-1]
cellIds = model.reservoir.history("/reservoir/Elements/Cell Id")[-1]
```

```
for i in range(p.size):
        cid = cellIds[i]
        if (cid >= 0):
                output.write("p%d=%f\n" % (cellIds[i], p[i]))
output.close()
```

After a successful model run the file `Observations.txt` will look similar to:

```
p0=9925052.000000
p1=9672950.000000
p2=9420927.000000
p3=9168982.000000
p4=8917115.000000
p5=8665327.000000
p6=8413617.000000
p7=8161985.000000
p8=7910431.500000
p9=7658956.500000
p10=7407560.000000
p11=7156241.500000
p12=6905001.500000
p13=6653839.500000
p14=6402755.500000
p15=6151750.000000
p16=5900822.500000
p17=5649973.500000
p18=5399157.000000
p19=5136881.500000
```

This is an easy format to parse by the instruction file. We hence recommend that you generate these intermediate files in as simple a format as possible as to avoid unnecessary work generating complicated instruction files.

E.1.3 The Instruction File

The instruction file provides *PEST* with commands on how to parse the model output file in the search for the observations. In our case we will use it to work on the intermediate file, i.e. `Observations.txt`.

PEST allows for a plethora of complex commands to extract the observations from output files of all different kinds of simulators. However since we generated a very simple file we can get away with a simple instruction file as well. Our file `columnPEST.ins` hence looks like this:

```
pif #
l1 #p0=# !p0!
l1 #p1=# !p1!
```

```
l1 #p2=# !p2!
l1 #p3=# !p3!
l1 #p4=# !p4!
l1 #p5=# !p5!
l1 #p6=# !p6!
l1 #p7=# !p7!
l1 #p8=# !p8!
l1 #p9=# !p9!
l1 #p10=# !p10!
l1 #p11=# !p11!
l1 #p12=# !p12!
l1 #p13=# !p13!
l1 #p14=# !p14!
l1 #p15=# !p15!
l1 #p16=# !p16!
l1 #p17=# !p17!
l1 #p18=# !p18!
l1 #p19=# !p19!
```

You can test its integrity of the instruction file by using

```
$ inscheck columnPEST.ins
```

E.1.4 The Control File

The files `columnPEST.pst` contains the control file which *PEST* uses for determining the type of simulation to run, how to generate parameter sets and how to interpret observations. Creating this control file is one of the more daunting tasks when using *PEST*; you will need to consult the *PEST* manual when writing your own control file to figure out how to insert or edit control parameters. We will limit explanation of this file to the relevant sections only.

The top section looks like:

```
pcf
* control data
restart estimation
1 20 1 0 1
1 1 single point
10.0 -3.0 0.3 0.03 10
10.0 10.0 0.001
0.1
50 0.005 4 4 0.005 4
1 1 1
```

The most important parameters to mention in this section are the ones on line 5. You must always use `single` for the `PRECIS` parameter, and `point` for `DPOINT`.

The remaining parameters you will need to set according to your own simulation needs.

Next you need to specify the parameter groups and parameters:

```
* parameter groups
group relative 0.01 0.0001 switch 2.0 parabolic
* parameter data
ENTHALPY none relative 200e3 0.0 1200e3 group 1.0 0.0
```

Here we specify our `ENTHALPY` parameter which we have used in the template file. Note again that since *Volsung* always requires parameters in SI units we need to specify them accordingly. In our case we use Joule/kilogram for the specific enthalpy that the parameter represents.

A note for the parameter groups if you run *PEST* in modes which require the calculation of derivatives: When you use `relative` as `INCTYP` you will need to provide the `DERINC` parameter; in the above example it is set to 0.01. Since *Volsung* in general outputs data using single precision floating point numbers with a relative precision of $\approx 10^{-8}$ you should not set `DERINC` to values smaller than $\sqrt{10^{-8}} \approx 0.0001$; values smaller than this may reduce the precision of the derivatives.

Next we create the observation groups and observations:

```
* observation groups
obsgroup
* observation data
p0 9925052.000000 1.0 obsgroup
p1 9672950.000000 1.0 obsgroup
p2 9420927.000000 1.0 obsgroup
p3 9168982.000000 1.0 obsgroup
p4 8917115.000000 1.0 obsgroup
p5 8665327.000000 1.0 obsgroup
p6 8413617.000000 1.0 obsgroup
p7 8161985.000000 1.0 obsgroup
p8 7910431.500000 1.0 obsgroup
p9 7658956.500000 1.0 obsgroup
p10 7407560.000000 1.0 obsgroup
p11 7156241.500000 1.0 obsgroup
p12 6905001.500000 1.0 obsgroup
p13 6653839.500000 1.0 obsgroup
p14 6402755.500000 1.0 obsgroup
p15 6151750.000000 1.0 obsgroup
p16 5900822.500000 1.0 obsgroup
p17 5649973.500000 1.0 obsgroup
p18 5399157.000000 1.0 obsgroup
p19 5136881.500000 1.0 obsgroup
```

In our case we provide values for the pressures in the 20 blocks of the simulation; the values above are the expected values when run with an enthalpy of $800 kJ/kg$ but in your case the observations may relate to real world measurements.

The next section describes the model command line, which is simple in our case since we hide most of the complexity of running the model in the *Python* script:

```
* model command line
./runModel.py
```

Note in the above it is assumed that your operating system knows how to run a *Python* script. In *Windows* you may need to assign your *Python* interpreter as a default application for `.py` files before using the command like this, and replace `./runModel.py` with `runModel.py` (i.e. omit the colon and forward slash).

The last section tells *Python* how to generate the model input file from the template and over which file to run the instructions:

```
* model input/output
columnPEST.tpl columnPEST.brynhild
columnPEST.ins Observations.txt
```

You can test the integrity of the control file by using:

```
pestchek columnPEST.pst
```

E.2 Running PEST

Once you have prepared the necessary files you can run *PEST*. If run successfully it will generate a number of files containing information about the process; in particular you should check the file ending in `.rec` which records a lot of run information.

Note that when *PEST* complains about missing files it is most likely something in the run went wrong. Hence in this case you need to figure out the underlying problem instead of trying to find out why the files are not there.

E.2.1 Sequential Runs

The easiest way for running *PEST* is in sequential mode. You can do this directly in the `columnPEST-master` folder:

```
$ cd columnPEST-master
$ pest columnPEST.pst
```

E.2.2 Parallel Runs using BEOPEST

PEST becomes a very powerful tool if you can run your forward model many many times over, for example when using inverse modelling or Monte-Carlo methods. Running it in sequential mode on a single computer may result in long runtimes; you can speed-up this process by running it in parallel mode using *BEOPEST* and *Volsung*'s system of remotes. See appendix B on how to setup remotes for *Volsung*.

Once you have setup and started your remotes you need to start *BEOPEST* as both master and worker processes.

Finding a Network Port

Master and workers will communicate with each other; for this you will first find an unused network port for your computer. We will use port 4004 in our example which is free on most machines but you should still check this.

On *Linux* use

```
$ sudo ss -tulw | grep 4004
```

If the above does not generate output in the terminal then port 4004 is free and can be used.

On *Windows* open a command prompt as administrator, then use

```
$ netstat -ab
```

and check if port 4004 is listed as in use.

Master Process

Start the *BEOPEST* master process from the terminal inside the `columPEST-master` folder:

```
$ beopest columnPEST.pst /h :4004
```

BEOPEST will start and listen on port 4004 for incoming workers. It will then assign the workers parameter sets and wait until they have completed their tasks. The master process will not run forward models itself.

Worker Processes

While the master process is set up and waiting you can start worker processes. There's no real limit on how many of these you can start but they are of course limited by the number of remotes you have access to. Also there may be limits on how many of these can actually run in parallel; for example in inverse modelling you can not run more than $N + 1$ processes in parallel where N is the number of free parameters of the inverse model.

For each remote you can access make a copy of the master folder, i.e. it should contain the same set of files which we have created before. We recommend you keep the following folder structure:

```
columnPEST
columnPEST-master
columnPEST-Tex
columnPEST-Kane
columnPEST-Teak
```

where in our example `Tex`, `Kane` and `Teak` are the names of the remotes you want to use.

Then inside each worker's folder edit the `runModel.py` and edit the `remote` variable which was mentioned above in E.1.2:

```
remote = "Tex"
```

Note that capitalization matters for the remote names. Also if you leave this variable empty the worker will start on your local machine, i.e. you can have a worker process which does not use a remote.

Now you can start the worker process. Open another terminal inside the worker's folder and start the worker:

```
$ beopest columnPEST.pst /h localhost:4004
```

The worker will try to connect to the remote it is assigned to, then transfer the input files over and run the forward model. On completion it will transfer the results back, use the instructions file to analyse the observations and pass these on to the master. Then it will wait for the next work assignment.

Once *BEOPEST* is finished the worker folders become superfluous, i.e. you can delete them to free up space.