PROCEEDINGS, Thirty-Eighth Workshop on Geothermal Reservoir Engineering
Stanford University, Stanford, California, February 11-13, 2013
SGP-TR-198

CONTROLLING COMPLEX GEOTHERMAL SIMULATIONS USING PYTOUGH

John O'Sullivan, David Dempsey, Adrian Croucher, Angus Yeh, Mike O'Sullivan

Department of Engineering Science, University of Auckland
Private Bag 92019
Auckland 1142, New Zealand
e-mail: jp.osullivan@auckland.ac.nz

ABSTRACT

The TOUGH2 simulator and its parallel version TOUGH2-MP continue to be the industry standards for the development of numerical models of geothermal systems. Increasing processing power enables us to simulate larger, more complex systems, while improved data collection and remote sensing techniques provide an ever greater suite of observations against which to calibrate the model. As a consequence, the raw text input files that control the simulations grow increasingly cumbersome to construct, while post-processing of model output becomes more challenging. We use the Python scripting language and the PyTOUGH library to interact with TOUGH2 in a number of novel ways to control simulation tasks accurately and efficiently. In this paper several examples of the use of PyTOUGH are described. A new method for automatically generating geothermal production wells is described that allows well information to be stored in concise, readable and easily updatable files. Examples are given of sequential modification of model geometries to represent effects such as eruptions and excavation. The generation of atmosphere blocks that vary in space and time due to changes in lake levels and extreme altitude are presented. Methods for controlling sequential simulations with both TOUGH2 and TOUGH2-MP are described including techniques for dealing with numerical non-convergence. Finally, PyTOUGH's ability to combine and post-process results from multiple simulations and different types of output files is discussed.

INTRODUCTION

As numerical simulation has become an established and widely used tool for planning and managing geothermal developments it is increasingly applied to more varied and complex problems [Burnell *et al*., 2012]. The increase in affordable parallel processing power has enabled the simulation of larger, more detailed systems. It has also provided a means for applying inverse modelling techniques to real-world scale problems [Cui *et al*., 2011; Omagbon & O'Sullivan, 2011].

A significant amount of book-keeping has always been required to prepare, run and post-process numerical simulations of geothermal systems. However, as these new simulations are orders of magnitude larger and more complex, this task has become increasingly difficult. The PyTOUGH library [Croucher, 2011; Wellmann *et al*., 2012] was developed to simplify this process and in the face of increasing complexity it has become an essential part of our numerical models. This paper describes a number of recent developments that use PyTOUGH to control key components of a numerical simulation so that accurate results can be obtained as efficiently as possible.

PREPARING SIMULATIONS

Before TOUGH2 simulations can be run several tasks must be performed. The main tasks are the generation of the simulation grid, calculation of the initial and boundary conditions and the preparation of the TOUGH2 data file. For complex simulations each task can require involved calculations that may often need to be carried out many times. As previously noted [Croucher, 2011], the PyTOUGH libraries provide a flexible framework for creating scripts that are able to achieve these tasks multiple times both efficiently and accurately. They also form a record of the preparation of each simulation in such a way that they effectively define the simulation itself.

Examples of complex operations carried out using PyTOUGH libraries are given for each of these tasks in the following subsections. Brief details of the simulation are also given to provide context for the particular challenge being addressed.

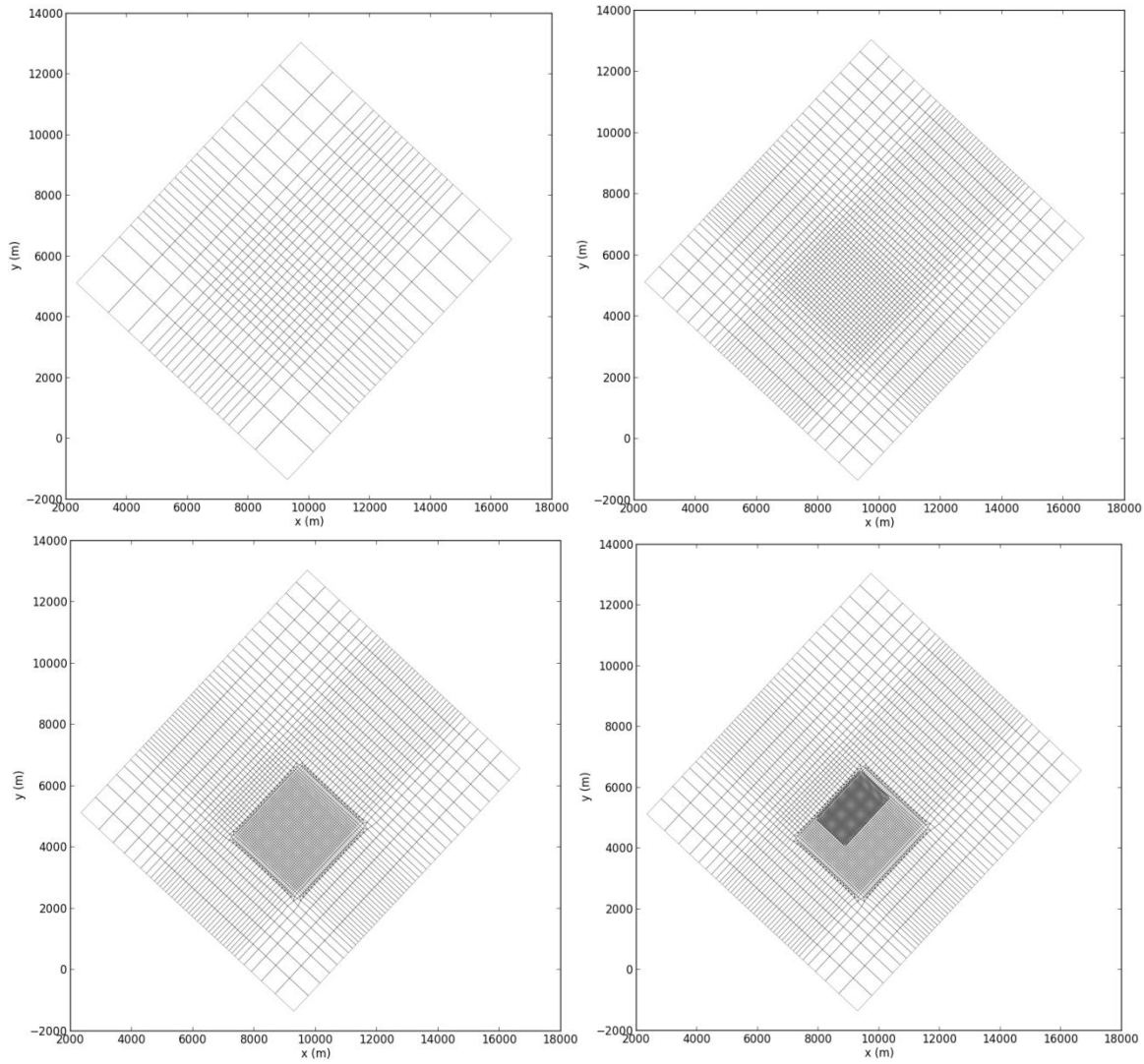


Figure 1: Series of grids with increasing refinement

Grid generation

Generating grids efficiently and accurately is important in simulations of geothermal systems. For a standard forward simulation several attempts are often required to achieve a satisfactory grid. PyTOUGH provides a simple mechanism for not only controlling and altering grid dimensions, resolution and position but also for fitting topography and optimising grid structure. Details can be found in the PyTOUGH documentation [Croucher, 2012]. For more complex simulations these capabilities are not only important but become essential tools for grid generation.

Recently we began developing a new model of the geothermal system on Lihir Island in Papua New Guinea. The objective is to use the current well-calibrated model [O'Sullivan *et al*., 2011] as a basis for a set of nested models based on the important geological structures. The inverse modelling tool PEST [Doherty, 2010] is used to calibrate the new model and will eventually be used for uncertainty quantification of the models' predictions. The computationally intensive nature of inverse modelling makes it expensive to apply to the current Lihir model which is comprised of approximately 80000 blocks. It will be significantly more expensive to apply inverse modelling to the new model as it is estimated that a grid containing 120,000 blocks is required to satisfactorily resolve the important geological structures and both the deep and shallow zones of the reservoir.

This computational cost can be greatly reduced by using simpler, less refined grids for calculation of the numerical Jacobians required for the inverse modelling process. PyTOUGH has been used to carry

out the complex task of generating and optimising the nested grids. Adjustments to the nested grids have been necessary and will continue to be necessary as the calibration process continues. The PyTOUGH scripts greatly simplify this otherwise cumbersome task. Figure 1 shows the four levels of the nested grid which is aligned NE-SW along the direction of the fault that is considered to be most important for controlling upflow in the system. The grids have 9700, 50000, 96000 and 120000 blocks respectively.

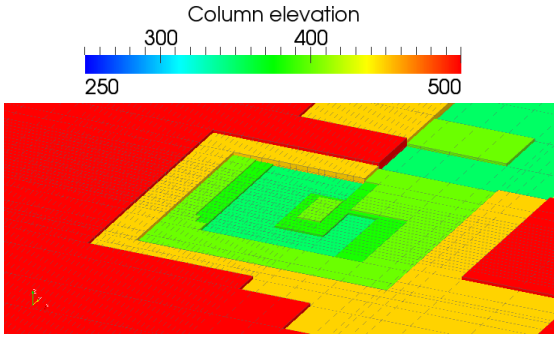
Grid modification

In some cases the topography of the geothermal system being simulated may change during the timeframe considered. Examples where this may occur include eruptions, subsidence and excavation. PyTOUGH can be used to generate sequential grids that represent the changing topography. It can also be used to control the sequential simulations as was described for the Lihir island simulations in O'Sullivan *et al*. [2011].

Another example of this procedure is shown in Figure (2). In this case the simulation is of the Waimangu Valley in New Zealand with topography altered by a basaltic dike eruption that destroyed the famous Pink and White Terraces in 1886. The objective of the project was to use TOUGH2 simulations to reproduce the surface features observed in the valley both before and after the eruption in order to better understand volcanic perturbation of the sub-surface heat flow and permeability. Unlike the Lihir Island simulations this catastrophic event took place over a very short period of time. In the model this is represented as a step-change between pre- and post-eruption topographies. PyTOUGH was used to generate the initial simulation grid using pre-1886 survey maps and then to calculate and remove the volume of earth necessary to arrive at today's topography.

As noted previously [O'Sullivan *et al*., 2011], care must be taken to ensure that the internal indexing of TOUGH2 data files and initial condition files is consistent when using sequential grids with modified topography. This can be achieved by the PyTOUGH scripts used for running the simulations as described below.

Note also that the grid geometry manipulation methods in PyTOUGH make use of the MULgraph geometry file format for representing geothermal model grids [O'Sullivan & Bullivant, 1995]. This format can represent arbitrary unstructured horizontal grids, projected down through a series of layers. The upper layers can be incomplete, and the surface elevations in the top layer can be specified to



(a) Pre-eruption grid

(b) Post-eruption grid

Figure 2: Pre- and post-eruption Waimangu grids

represent varying topography. This geometry format can be used independently of the MULgraph graphical user interface. Also, users who prefer to use other geometry formats for grid generation can still use the rest of the PyTOUGH library for manipulating TOUGH2 data files, simulation output etc.

Atmosphere Blocks

Many simulations of geothermal systems extend to the surface of the earth and hence contain the atmosphere as one of the boundary conditions. In simple cases this can be represented easily by connecting an atmosphere block with an extremely large volume to the top layer of the model and then setting the initial conditions for this block to atmospheric temperature and pressure. However, for more complex simulations the properties of these blocks are difficult to determine and may vary with time. Two examples are given below of different atmospheric conditions and how PyTOUGH is used to calculate them.

The first example is from the simulation of a geothermal system under a dormant volcano in the high Andes on the border between Chile and Bolivia. In this case the altitude of the simulated area varies from 3900 to 5200m such that both temperature and pressure change significantly over the surface of the model. PyTOUGH is used to calculate the pressure and temperature for each block and assign it in the TOUGH2 initial condition file.

The pressure p is calculated using Equation (1) [Wikipaedia, 2013]:

$$p = p_0 \left(1 - \frac{Lh}{T_0}\right)^{gM/RL}, \quad (1)$$

and the temperature T is given in Celsius using:

$$T = T_0 - Lh - 273.15, \quad (2)$$

where T_0 is the temperature at sea level in kelvin, h is the altitude in meters and the constants are given in Table (1).

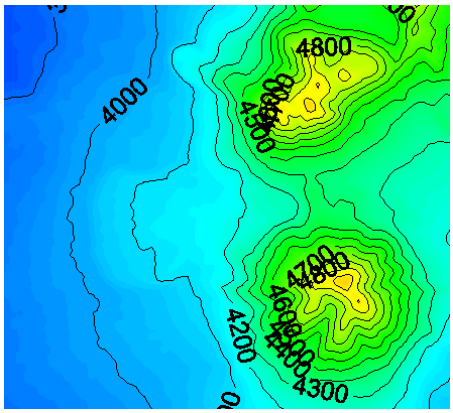
Table 1: Constants for calculation of high altitude temperature and pressure

p_0	Pressure at sea level in Pa	101325
L	Temperature lapse rate K/m	0.0065
g	Gravity m/s^2	9.80665
M	Molar mass of dry air kg/mol	0.0289644
R	Universal gas constant J/(mol.K)	8.31447

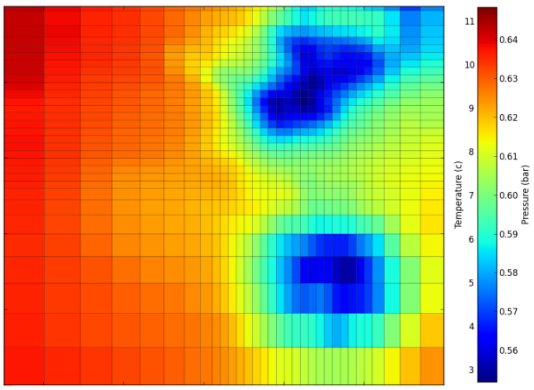
A contour map of a portion of the simulated area is shown in plot (a) of Figure (3). The corresponding pressure and temperature boundary conditions that were applied for the same area are shown in plot (b). It can be seen that the temperature varies from 3°C to 11°C and the pressure from 0.55 bar to 0.65 bar for the area shown.

The second example shows how PyTOUGH is used to change the atmospheric conditions during the timeframe of a simulation. Many simulations of geothermal systems include bodies of water at the surface. Examples include Wairakei, adjacent to the Waikato River, [O'Sullivan & Yeh, 2010], Lihir Island, adjacent to and beneath the Pacific Ocean, Rotorua, abutting Lake Rotorua and Waimangu [Simmons & O'Sullivan, 2010]. The hydrostatic pressures that these bodies exert on the surface represent a different set of atmospheric boundary conditions. In the case of the Orakei-Korako geothermal system, which spans the Waikato River, these hydrostatic pressures have changed over simulation timeframes due to the dam impoundment and valley flooding.

The objective of this project was to simulate the natural state of the river valley both before and after flooding and to calibrate the model using the known



(a) Elevation contours of Andean volcano



(b) Simulation atmosphere boundary conditions

Figure 3: Pressure and temperature in atmosphere blocks above Andean model.

surface features in both cases. The model can then be used to predict the possible impact of future changes to the lake level on the remaining, protected geothermal expressions.

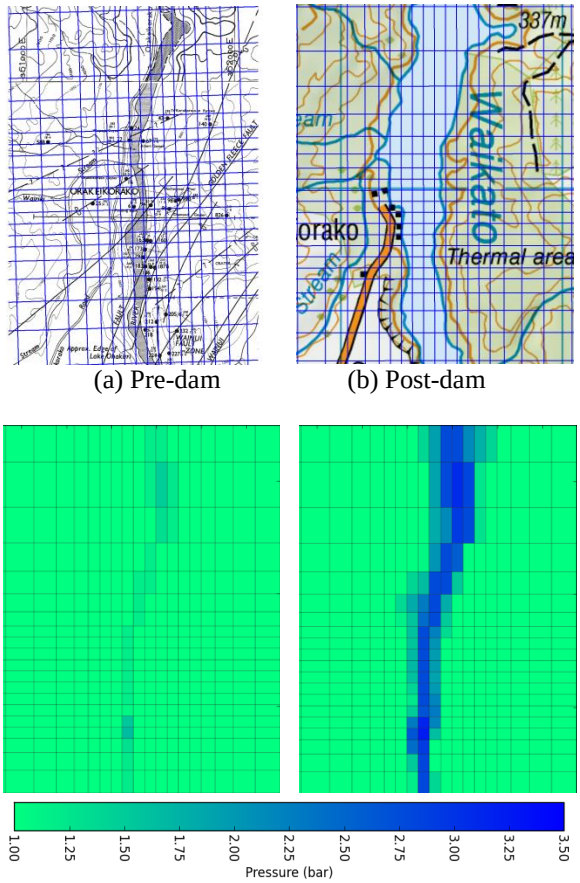
Pre-flood contour maps were used to determine the topography and a section of the map is shown in Figure (4a). For reference the computational grid is superimposed on the map in blue. The same area is shown as it is today in plot (b). The average depth of the pre-flood river was estimated to be 2m and this was used to calculate the topography beneath the river. The map was digitised and PyTOUGH was used to fit the surface of the model to the topography. PyTOUGH was also used to adjust the topography to account for a coordinate transformation between two maps in Figure 4. This difference can be seen as the skewed representation of the computational grid in plot (a).

Having created the geometry the pressures of the atmosphere blocks corresponding to the river were calculated and the values assigned in the initial conditions file. To determine the pressures for the

blocks flooded by the damming process PyTOUGH was used to "fill" the lake to its present level and calculate the correct values. The corresponding pressure boundary conditions are shown in plots (c) and (d) respectively. Once again running sequential simulations of this nature is achieved efficiently using PyTOUGH to select the appropriate boundary conditions based on the simulation time. Initial simulations have been carried out using a step change from one state to the next. However, future simulations in which the flooding of the valley is a gradual process are planned.

Creating Generators for Geothermal Wells Automatically

One of the implications of generating several versions of the same model is that the blocks that contain the feed zones of geothermal wells may change between versions. For geothermal fields with many production wells it becomes a cumbersome task ensuring that the allocation of a feed zone to a block for each well is correct.



(a) Pre-dam (b) Post-dam

(c) Pre-dam (d) Post-dam

Figure 4: Pressure in atmosphere blocks before and after flooding of Orakei-Korako.

Similarly, for a typical reservoir model where the grid has not changed but the data for a well is regularly modified or updated, it is easy to neglect to update the TOUGH2 data file and hence run an incorrect simulation. One cause of this error may be that the field data used to update the well information might have come from one of many different sources and may be stored in one of a number of files.

To overcome these issues we have used PyTOUGH to develop a script which automatically generates each geothermal well at the start of a simulation and writes the appropriate generator entry in the TOUGH2 data file. This has several advantages. The first is that because all of the generators for the geothermal wells are cleared and reproduced before a simulation is started, the most up to date well information is used every time. Second, all of the information for each well is stored in a single file. This file then acts as a master record for the well and can be updated and referred to easily.

The format of the well file is a follows:

```
#Well Name
GW017

#Feed zones
#From(mRL) To(mRL) Prop     Delv
402        373     0.2      1e-13
142        104     0.2      2e-13
-14        -148    0.6      1e-12

#Well Track
8832.10   5011.60   1099.50
8831.80   5011.60    979.50
8830.90   5011.30    859.50
…
9646.40   4677.90   -162.80
9652.40   4677.70   -168.10

#Downhole Temperature
#mRL        Temp
1092.65     39.40   #from spread.xlsx
1083.15     42.50
1074.05     41.60
…
420.02      264.20

#Mass Flow
#kg/s       Date
2003.00     1.051084E+00
2003.08     1.259917E+00
2003.16     1.251020E+01

#Enthalpy
#kJ/kg      Date
2003.00     1246.93
2003.08     1394.46
2003.16     1201.85
```

The well file is easy to read, concise and can contain comments which note where information has been obtained (eg. from spread.xlsx). It is also easy to update by simply appending sections when new field

measurements for mass flows, enthalpy etc. are obtained.

In the PyTOUGH script that sets up and controls the TOUGH2 simulation the geothermal wells are created simply by calling a library function which indicates the type of behaviour required:

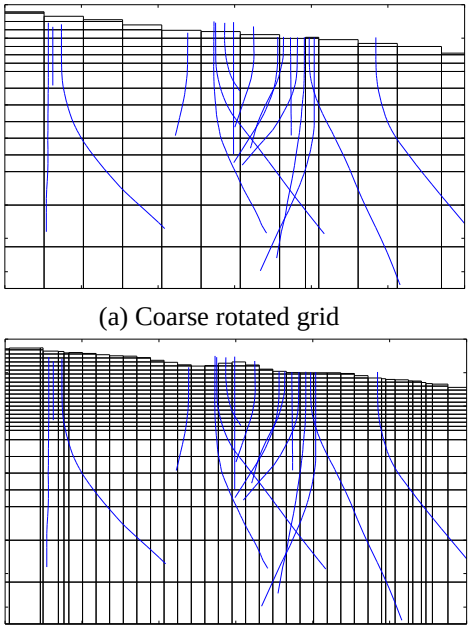
```
add_well('GW017', 'MASS')
```

This function performs several tasks. First it updates the geometry file to include the well track information for the well. Second it interpolates between many points along the well track within each feed zone. PyTOUGH functions are used to determine which simulation blocks contain these points hence determining the model blocks containing the feed zones. A threshold parameter is used to exclude blocks which only contain a small proportion of the feed zone. Next a generator is written to the TOUGH2 data file for each block containing a feed zone. Depending on the type of behaviour required the generator will either create a table of times and mass flows using the mass flow data and feed-zone proportions from the well file. Alternatively it can create a deliverability type well using the productivity information in the feed zone section. Note that switching from mass-flow type behaviour for history matching to deliverability type behaviour for future scenarios can be controlled by the PyTOUGH simulation script. Figure (5) shows the same group of wells generated for two grids of different resolution and orientation.

Finally, post-processing PyTOUGH scripts can access the well file to plot time history and downhole data thus ensuring that the same information is used in both the simulation and the visualisation of results.

RUNNING SIMULATIONS

Controlling complex simulations using PyTOUGH has been discussed in detail previously [Croucher, 2011; O'Sullivan *et al.*, 2011]. This section will only discuss additional work that has been carried out. One area of research has been to develop scripts that can control TOUGH-MP simulations. Because TOUGH-MP uses a fixed file naming convention, directory structures are used for each year of a sequential simulation. Files are prepared and the file structure organised using Python and PyTOUGH. Care must be taken to ensure that the working files are cleared for each simulation to ensure the results are correct.



(a) Coarse rotated grid

(b) Fine NS aligned grid

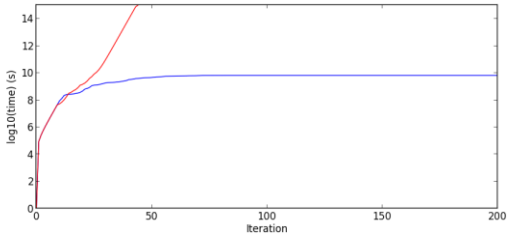
Figure 5: Plot of the same well tracks in grids of different resolution and orientation.

The flexibility of PyTOUGH is such that the same script can be used for both TOUGH2 and TOUGH-MP simulations by simply changing the executable call that is made. PyTOUGH handles the small variations in the data files and listing files automatically.

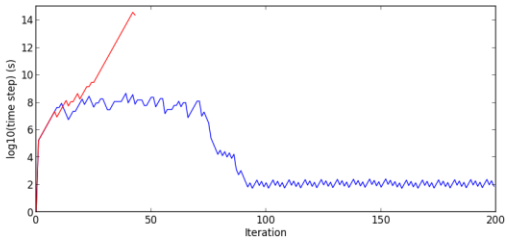
Improvements have been made in the scripts that run the simulations, greatly increasing their efficiency. For example when calculating which blocks need to be added and which need to be removed due to topography changes, Python's efficient 'set' data structures are now used for comparing lists of blocks.

Correcting numerical non-convergence

A significant development in the scripts used to run simulations is the addition of a check for numerical non-convergence. Numerical non-convergence can occur in Air-Water and CO_2-Water models in blocks where phase transitions are taking place. When this problem occurs the block in question switches between single-phase and two-phase at each Newton iteration causing TOUGH2 to reduce the time step dramatically. The plots in Figure (6) show the total time and the time step size for two very similar simulations, one which experiences numerical non-convergence and the other which does not.



(a) Total time (-) simulation converges
(-) non-convergence occurs



(b) Time step size (-) simulation converges
(-) non-convergence occurs

Figure 6: Plots of total time and time step size for two simulations. One which converges the other which does not.

The solution of this numerical issue is an area of current research. Previously the most effective approach was to manually stop the simulation, inspect the block in question, intervene and restart the simulation. This approach is extremely inefficient especially as the block affected may change throughout the simulation.

By using PyTOUGH scripts this process has been automated so that no manual intervention is required to implement the workaround. The algorithm for the process is shown in Figure (7).

The parameters *m* and *n* can be changed to enforce less or more rigorous checking. The script also tracks and records which blocks have been changed. This checking process improves the efficiency of simulations particularly in dynamic situations where the steam zone may be moving or evolving.

POST-PROCESSING SIMULATIONS

Once simulations have been completed post processing the results can be a challenging task. As discussed in O'Sullivan et.al [2011] the results from sequential simulations must be gathered together in the correct sequence and presented in a meaningful way. For TOUGH2 simulations this task involves extracting time-dependent information from listing files for each block, generator or connection and storing it in a data file. The same information is

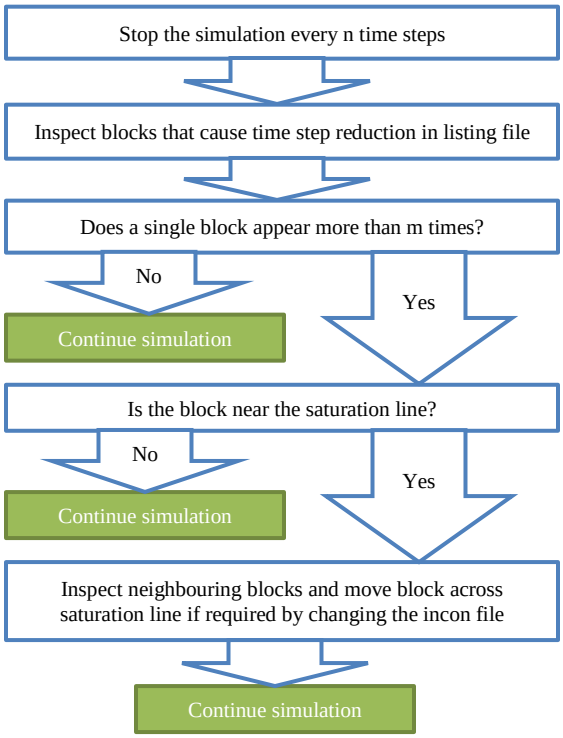
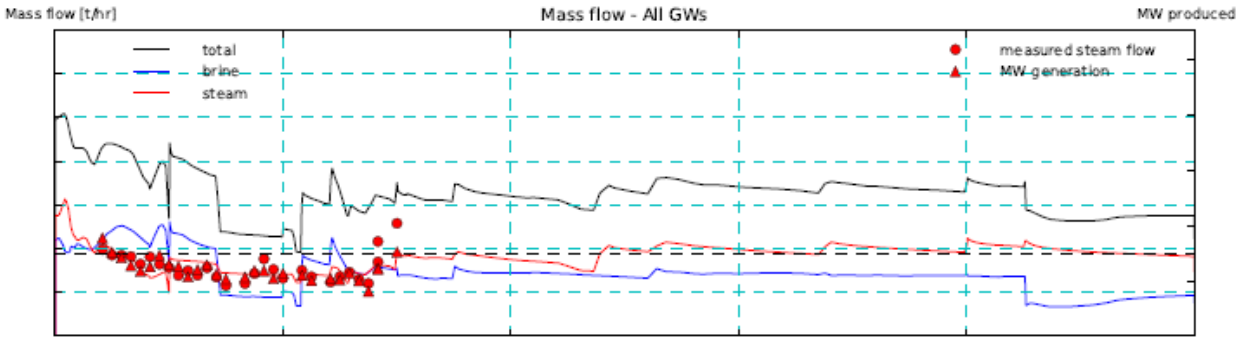


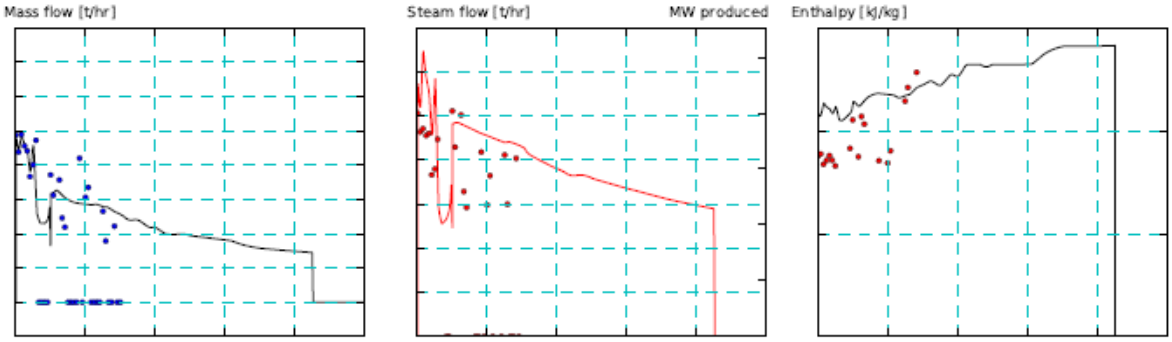
Figure 7: Algorithm for correcting numerical non-convergence.

extracted from each subsequent listing file and is appended to the data file.

For TOUGH-MP the process is significantly more complicated. TOUGH-MP stores time-dependent information about individual blocks in data files known as FOFT files. At present the standard version of TOUGH-MP does not properly record time-dependent information for generators or connections. We have corrected this problem so that valuable time-dependent generator information is stored in GOFT files during the simulation. However, separate FOFT and GOFT files are created for each processor during the simulation. This means the PyTOUGH script must reconcile many of these files to correctly collate the time-dependent data for wells and blocks. This is further complicated by the algorithm preventing numerical non-convergence as this effectively requires that each sub-simulation is broken up into a third tier of sub-sub-simulations each with a FOFT and GOFT file for each processor. The end result can be many thousands of files that must be reconciled. Python scripts and in particular the PyTOUGH libraries make this difficult task relatively simple and easily repeatable.



(a) Plots for totals calculated for a simulation spanning 10 years



(b) Plots for an individual well calculated for a simulation spanning 10 years

Figure 8: Results from a TOUGH-MP simulation

The plots in Figure (8) show examples of the types of plots produced from large, parallel, sequential TOUGH-MP simulations. Both field-wide totals and individual well totals are shown.

CONCLUSIONS

A number of novel approaches have been presented that use PyTOUGH libraries to control complex simulations of geothermal systems. Methods that can be used to automatically generate nested grids, calculate and apply boundary conditions and prepare wells have been shown to be important tools for maintaining accuracy and efficiency as simulations are carried out. They show particular promise as inverse modelling techniques are applied to real-world scale geothermal systems. PyTOUGH tools for controlling sequential simulations and post-processing data from large, parallel simulations have also been discussed and examples of their use given. As numerical models of geothermal systems become increasingly complex and address wider ranges of problems the capability of the PyTOUGH libraries to interact with, control and organise TOUGH2 and TOUGH-MP simulations will become increasingly important.

REFERENCES

Burnell, J., Clearwater, E., Croucher, A., Kissling, W., O'Sullivan, J.P., O'Sullivan, M.J. & Yeh, A. (2012), "Future directions in geothermal modelling," *Proceedings (electronic) 34rd New Zealand Geothermal Workshop*, University of Auckland, Auckland, New Zealand, 19-21 November, 2012.

Croucher, A. E. (2012), "PyTOUGH User's Guide," University of Auckland, Auckland, New Zealand, github.com/acroucher/PyTOUGH.

Croucher, A. E. (2011), "PyTOUGH: a Python scripting library for automating TOUGH2 simulations," *Proceedings (electronic) 33rd New Zealand Geothermal Workshop*, University of Auckland, Auckland, New Zealand, 21-23 November, 2011.

Cui, T., Fox, C., & O'Sullivan, M.J. (2011), "Bayesian calibration of a large-scale geothermal reservoir model by a new adaptive delayed acceptance Metropolis Hastings algorithm," *Water Resources Research*, **47** (10), W10521, doi:10.1029/2010WR010352.

Doherty, J. (2010), "PEST: model-independent parameter estimation. User manual, 5th edition." Watermark Numerical Computing, Corinda, Australia, http://www.sspa.com/pest.

Omagbon, J.B. & O'Sullivan, M.J. (2011), "Use of an heuristic method and PEST for calibration of geothermal models," *Proceedings (electronic) 33rd New Zealand Geothermal Workshop*, University of Auckland, Auckland, New Zealand, 21-23 November, 2011.

O'Sullivan, J.P., Croucher, A.E., O'Sullivan, M.J. Stevens, L. & Esberto, M. (2011), "Modelling the evolution of a mine pit in a geothermal field at Lihir Island, Papua, New Guinea," *Proceedings (electronic) 33rd New Zealand Geothermal Workshop*, University of Auckland, Auckland, New Zealand, 21-23 November, 2011.

O'Sullivan, M.J. & Bullivant, D. (1995), "A graphical interface for the TOUGH2 family of flow simulators," *Proceedings of the TOUGH workshop*, Lawrence Berkeley National Laboratory, University of California, Berkeley, 1995.

O'Sullivan, M.J. & Yeh, A. (2010). "Wairakei-Tauhara modeling report." *Report to Contact Energy*, Uniservices and Department of Engineering Science, University of Auckland, Auckland, New Zealand, 1-280, www.contactenergy.co.nz/web/pdf/environmental/P4ReservoirModellingReport.pdf.

Pruess, K., Oldenburg, K. & Moridis, G. (1999), "TOUGH2 user's guide, version 2.0," Lawrence Berkeley National Laboratory, University of California, Berkeley.

Simmons, S. & O'Sullivan, M.J. (2010), "Numerical Model of the Changes in Geothermal Activity in the Rotomahana-Waimangu System Due to the 1886 Eruption of Mt Tarawera," *Proceedings of the World Geothermal Congress*, Bali, Indonesia, 25-30 April, 2010.

Wellmann, J.F., Croucher, A., & Regenauer-Lieb, K. (2012), "Python scripting libraries for subsurface fluid and heat flow simulations with TOUGH2 and SHEMAT," *Computers and Geosciences*, **43**, 197-206, doi:10.1016/j.cageo.2011.10.011.

Wikipedia contributors (2013), "Atmospheric pressure." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, Web.

Zhang, K., Wu, Y.-S. & Pruess, K. (2008), "User's guide for TOUGH2-MP - a massively parallel version of the TOUGH2 code," Lawrence Berkeley National Laboratory, University of California, Berkeley.