PYTOUGH: A PYTHON SCRIPTING LIBRARY FOR AUTOMATING TOUGH2 SIMULATIONS

Adrian Croucher[1]

[1]Department of Engineering Science, University of Auckland, New Zealand

a.croucher@auckland.ac.nz

Keywords: *Reservoir modelling, TOUGH2, Python*

ABSTRACT

Setting up TOUGH2 simulations of geothermal reservoirs is still often carried out manually, by editing input files. However, for complex simulations this process can be labour-intensive and error-prone. Partly in response to this problem, various graphical interfaces for TOUGH2 have been produced. While these can reduce the likelihood of input errors and make simple simulations easier, graphical interfaces often limit the full capability of the TOUGH2 simulator. More importantly, neither manual nor graphical methods lend themselves easily to automation, which is desirable for more complex simulations.

A third approach, so far relatively unexplored, is that of scripting. The PyTOUGH library has been developed to enable the user to control potentially all aspects of a TOUGH2 simulation (from grid generation and model setup through execution, post-processing and analysis) via simple Python scripts. This approach avoids the limitations of both manual and graphical methods, and makes possible complex simulations involving, for example, suites of many model runs with varying parameters (including permeabilities, heat inputs and levels of grid resolution).

Here the possibilities of the PyTOUGH scripting approach are demonstrated using both idealized and real-world TOUGH2 applications.

1. INTRODUCTION

The TOUGH2 simulator (Pruess, 2004) takes its input for a geothermal reservoir model from a model input data file, a fixed-format text file that is not designed primarily to be human-readable. The positions of the data on each line are important, so that the accidental insertion or omission of even a single extra space or other character can mean the difference between a correctly-running model and a fatal error- or perhaps worse, a subtle error that goes undetected.

Despite this, TOUGH2 data files are still often prepared manually via a text editor. More recently, graphical user interfaces (GUIs) for TOUGH2, e.g. MULGRAPH (O'Sullivan and Bullivant, 1995) and PetraSim (Yamamoto, 2008), have been written, partly to aid data file preparation, as well as to provide graphical post-processing capabilities. These can make TOUGH2 modelling significantly easier and less error-prone, particularly for new users. However, this convenience generally comes at a cost, as GUIs often limit the full potential and flexibility of the simulator- e.g. by not allowing graphical access to more advanced features (some of which may simply not be well suited to graphical control), or by limiting the types of model grids that can be used.

Additionally, more complex simulations- for example, numerous linked model runs carried out in batches, the input for one run possibly being modified depending on the results of a previous one- really require some degree of automation. Neither manual nor GUI approaches lend themselves readily to this.

There is, however, a third approach which has so far remained relatively unexplored: that of scripting. A script is a simple program written in a high-level scripting language, in which objects like model grids, data input files or model output are treated as variables to be manipulated by the script. Such a script can automate some or all of the preparation, running, graphical post-processing and analysis of a TOUGH2 model, eliminating the need for manual input editing, while retaining full control of all simulation features. Previously-prepared scripts can also be adapted and re-used for new models, saving considerable time.

This paper presents PyTOUGH, a new scripting tool for TOUGH2, and demonstrates some of its potential via example applications.

2. PYTOUGH

2.1 Introduction

PyTOUGH uses the Python scripting language. Python is versatile, powerful and easy to learn, and is freely available under an open-source license on all major computing platforms. Python scripts, unlike traditional programs written in e.g. Fortran or C/C++, do not need to be compiled, which speeds development.

Python extension libraries are available for very diverse tasks including science and engineering applications, for which Python has become increasingly popular (Langtangen, 2008). PyTOUGH makes use of the Numerical Python library (http://numpy.scipy.org/) to enable efficient computation even for larger models. PyTOUGH itself takes the form of an extension library, which can be called from any Python script.

No other scripting approaches for TOUGH2 appear to have been reported in the literature. Audigane et al. (2011) did present a set of Fortran routines for pre- and post-processing of TOUGH2 simulations, which has some similarity to parts of PyTOUGH's functionality. However, it is not based on a scripting language, and was not intended to provide complete control of the simulation, and so is not directly comparable.

PyTOUGH supports both standard TOUGH2 and AUTOUGH2 (the University of Auckland version), which uses slightly different input and output file formats. PyTOUGH contains several modules for controlling different aspects of TOUGH2 simulations, as described below.

2.2 Grid geometry

TOUGH2 uses a flexible finite-volume formulation, in which the model grid is described only as a set of block volumes and the connections between them. The blocks are not referenced to any particular coordinate system. But for grid generation and post-processing of model results, a coordinate system is usually needed.

Accordingly, AUTOUGH2 is generally used in conjunction with a separate 'MULGRAPH geometry file' which contains a geometric description of the grid (O'Sullivan and Bullivant, 1995). This can be used to generate a TOUGH2 finite-volume grid or for post-processing of results. It assumes a layered grid structure in the vertical, but allows arbitrary unstructured columns in the horizontal. The geometry file format was originally designed for use with the MULGRAPH GUI, but can be used equally well by other software.

Because Python is an object-oriented language, something as complex as a grid geometry can be represented as a single 'object', with its own 'properties' (variables) and 'methods' (functions it can carry out). Each object is an instance of a 'class' which specifies how objects of that class will work. PyTOUGH provides a **mulgrid** class for representing the contents of a geometry file as an object in a script. This class has methods for generating simple irregular rectangular 3-D grids, reading from and writing to disk, importing from or exporting to other formats, and for carrying out a range of grid operations including translation and rotation, local grid refinement, error checking, evaluating and optimizing grid quality and fitting topographic data.

For example, the section of Python script below creates a rectangular grid geometry from the grid size lists `dx`, `dy` and `dz`, refines a central area of interest, rotates it horizontally by 45° about its centre and fits surface topographic data from a file. As can be seen, in Python the methods (and properties) of an object are accessed using a dot after the object's name.

```
geo=mulgrid().rectangular(dx,dy,dz)
geo.refine(central_columns)
geo.rotate(45)
surface_data=np.loadtxt('surface.dat')
geo.fit_surface(surface_data)
```

These few lines, together with some simple code to specify the grid sizes and the location of the area to be refined, can be used to produce 3-D TOUGH2 grids such as the one shown in Figure 1.

It should be noted that PyTOUGH does not require the user to use MULGRAPH geometry files. Further modules could easily be included in PyTOUGH to represent other desired grid geometry formats and create TOUGH2 grids from them. PyTOUGH can also be used independently of any grid geometry pre-processing.

PyTOUGH also includes a separate **geometry** module which contains functions for carrying out various useful geometric calculations in 2-D, for example determining if a point is in a given polygon, and performing linear transformations (e.g. between different coordinate systems).

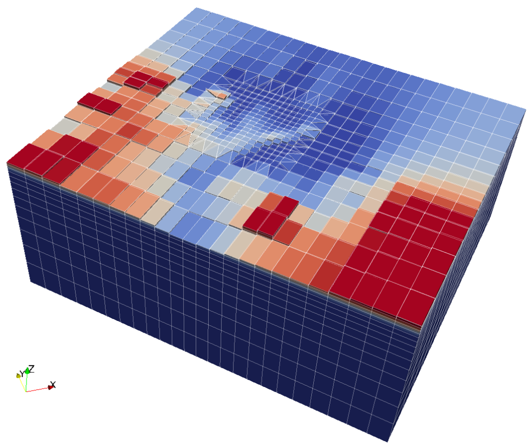

Figure 1: A rotated, locally refined rectangular TOUGH2 grid with fitted surface (shaded by elevation), generated using PyTOUGH

2.3 TOUGH2 input

The PyTOUGH **t2data** class contains a complete representation of the contents of a TOUGH2 input data file. This includes a 'grid' property (represented by a separate **t2grid** class) which holds the TOUGH2 finite-volume grid data and contains properties for accessing the individual grid blocks, connections and rock types. For convenience these properties can all be accessed either by index or by name. Further functionality includes error checking, transferring rock types and generators from one TOUGH2 model to another (e.g. when creating a refined model), and calling TOUGH2 to run the model.

For example, the following lines read an existing TOUGH2 data file, replace its grid with one created from the geometry defined above, write an updated data file and run the model:

```
dat=t2data('model.dat')
dat.grid=t2grid().fromgeo(geo)
dat.write('new_model.dat')
dat.run()
```

PyTOUGH also includes a **t2incon** class for handling initial conditions files, which have the same format as the final 'save' files produced at the end of a run.

2.4 TOUGH2 output

TOUGH2's main output is to a 'listing file', which is also a text file with a format particular to TOUGH2, and contains tables for results at blocks, connections and generators. PyTOUGH's **t2listing** class can represent a listing file as an object in a Python script. Listing files can be read from disk and (for transient simulations) navigated in time, and results within the various tables can be accessed individually or 'sliced', to give e.g. all results at a particular block, or an array of a particular quantity (such as temperature) over all blocks. Additionally, time histories of block, connection or generator quantities can be returned. Other useful functionality includes the ability to check convergence of steady-state runs by comparing results at different times.

Graphical post-processing of TOUGH2 grids and results can be carried out using PyTOUGH in two different ways.

Firstly, the **mulgrid** class provides methods for producing two-dimensional plots over grid layers or vertical slices, via the Matplotlib Python plotting library (http://matplotlib.sourceforge.net/). Arbitrary arrays can be plotted. These are typically slices from tables in a **t2listing** object, but could also be derived quantities calculated from them. Simple one-dimensional plots over lines or polylines (e.g. deviated well traces) in 3-D are also possible.

Secondly, PyTOUGH can also export TOUGH2 grids and results to Visualization Toolkit (VTK) XML files (http://www.vtk.org) for 3-D visualization using ParaView (http://www.paraview.org/) or similar software. (Figures 1, 2 and 5-8 in this paper were created this way.)

2.5 Thermodynamics

PyTOUGH contains thermodynamic 'steam table' functions for calculating the properties of water and steam within Python scripts. Two versions are included: the IFC-67 formulation used by TOUGH2, and the updated IAPWS-97 formulation used by the supercritical version of AUTOUGH2 (Croucher and O'Sullivan, 2008).

3. EXAMPLE APPLICATIONS

3.1 Convection in a 2-D confined aquifer

This example uses a relatively simple idealized model to demonstrate the potential of PyTOUGH for automating all stages of modelling, from grid generation and model setup through execution, post-processing and analysis. The model is a natural-state 2-D vertical slice of a confined aquifer with a localized heat source below, modelled with TOUGH2 using the EOS1 (pure water) equation of state.

3.1.1 Model setup, execution and post-processing

The model domain extends 15 km in the horizontal and is 1.5 km deep. The aquifer has isotropic permeability 10^{-13} m^2 and is confined above and below by low-permeability (10^{-16} m^2) layers with thickness 250 m. There is a Gaussian heat input distribution over the central 5 km of the base of the model, with peak value 0.3 W/m^2 and standard deviation 750 m. Boundary conditions are no-flow at the bottom and sides, and atmospheric temperature and pressure at the surface.

Using the PyTOUGH **mulgrid** class, an irregular rectangular grid for this problem can readily be set up, as was done in section 2.2. A refined grid is used for the central region above the heat source, outside which the grid sizes increase progressively towards the lateral boundaries. These grid sizes can easily be computed using functions available from the Numerical Python library, then passed to the `rectangular()` function as above to create the grid geometry and hence the TOUGH2 grid. For this model there are in total 221 columns and 60 layers. The script can then define two rock types, for the aquifer and confining layers, and loop over the grid blocks, assigning the appropriate rock type to each (in this case simply based on the elevation of each block).

The heat input distribution is represented in TOUGH2 by a heat generator in the bottom block of each column in the central region of the model, with heat flow rate determined by distance from the centre. First, a list of the central columns in the geometry `geo` is constructed:

```
cols=[col for col in geo.columnlist
      if side_width <= col.centre[0]
      <= side_width+centre_width]
```

and the bottom layer is identified by:

```
layer=geo.layerlist[-1]
```

(where in Python an index of -1 indicates the last element of a list or array). The Gaussian distribution of heat flow rates for these columns can be calculated from:

```
qmax,qsd=0.3,750.
x0=0.5*(cols[0].centre[0]+
      cols[-1].centre[0])
dxs=np.array([col.centre[0]-x0 for
      col in cols])/qsd
qcol=qmax*np.exp(-0.5*dxs*dxs)
```

The generators can now be created and added to the **t2data** object `dat` (representing the TOUGH2 data file) using:

```
for col,q in zip(cols,qcol):
    blkname=geo.block_name(layer.name,
      col.name)
    genname=' q'+col.name
    dat.add_generator(t2generator(name=
      genname,block=blkname,
      type='HEAT',gx=q*col.area))
```

Initial conditions for the natural-state simulation (in this case, just uniform atmospheric pressure and temperature in all blocks) can be created using the PyTOUGH **t2incon** class:

```
ptop,ttop=101.3e3,20.
inc=t2incon()
for blk in dat.grid.blocklist:
      inc[blk.name]=(ptop,ttop)
```

All other simulation parameters can also be set up via the script. (Often it is convenient to have the script open an existing TOUGH2 data file and modify only what needs changing for the new model.) The script can then execute the simulation, open the resulting TOUGH2 listing file and export the results to VTK format for visualization using:

```
dat.run()
lst=t2listing('model.listing')
lst.write_vtk(geo,'model.pvd',dat.grid)
```

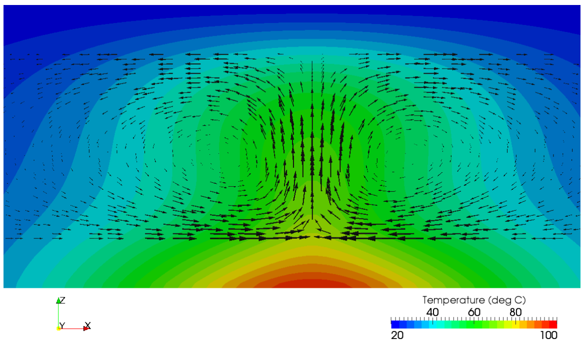

Figure 2: Temperature and mass flux in a confined aquifer heated from below, simulated using TOUGH2 via a PyTOUGH script

Figure 2 shows the results of the simulation, plotted using ParaView. The elevated temperatures and convection cells in the aquifer above the heated area can be seen.

Hence, using PyTOUGH, the entire modelling process for this problem, from grid generation to visualization of results, can be carried out using a single Python script.

3.1.2 Grid refinement study

Part of any TOUGH2 modelling problem is the determination of the appropriate model grid size. Generally the optimal grid size is taken to be the largest one that will resolve all the important features of the problem, giving the most economical solution that is still accurate. This can be determined using a grid refinement study, in which the grid is progressively refined until the solution ceases to change appreciably. In practice, grid refinement studies are often omitted because of the extra work involved in setting up multiple refined models using manual or GUI methods.

A grid refinement study is not difficult, however, if the model grid is constructed using the scripting approach. Generating results for a finer grid is simply a matter of changing a few variables in the script. In fact, the whole grid refinement study can itself be scripted as a loop.

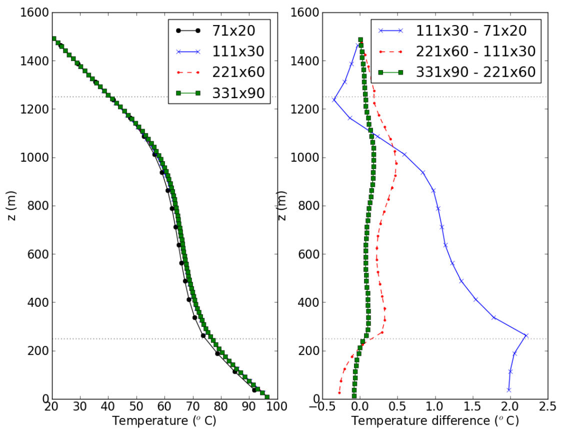

Figure 3: Effect of grid size on temperature profiles for confined aquifer model

Figure 3 shows the results of a grid refinement study carried out on the 2-D confined aquifer model. The plots (again produced using a Python script) show temperature profiles at the centre of the model for a range of different grid sizes, and the differences between them. It can be seen that refinement from 71x20 blocks (vertical grid size 75 m) through 111x30 blocks (50 m) to the original 221x60 grid (25 m) has a noticeable effect on the temperature profiles, but further refinement to 331x90 blocks (16.67 m) does not give appreciable improvement. Hence it can be concluded that 221x60 is an appropriate grid resolution for this problem.

3.1.3 Effect of aquifer permeability on peak heat flux

Using the scripting approach, it is also straightforward to iterate over a range of values for a given model parameter and investigate the effect on the model results. For example, we may wish to know how the aquifer permeability affects the peak heat flux at the ground surface. The following script iterates over a logarithmic range of aquifer permeabilities, runs the model for each case, calculates the peak surface heat flux at the end of each simulation and stores the results in a list:

```
kvalues=np.logspace(-12,-15,num=20)
icol=geo.num_columns/2
area=geo.columnlist[icol].area
qtop=[]
for k in kvalues:
    dat.grid.rocktype['formn'].
        permeability=[k]*3
    dat.write('model.dat')
    dat.run()
    lst=t2listing('model.listing')
    lst.last()
    qm=lst.connection['Heat flow'][icol]
    qtop.append(-qm/area)
    t2incon('model.save').
        write('model.incon')
```

The last line copies each set of final model results to the initial conditions for the next run, to save simulation time (so only the first run needs to start from 'cold' conditions).

The results are shown in Figure 4. For very low permeabilities, heat flow in the system is mainly by conduction and the peak surface heat flux is low. As the permeability is increased, the convection cells form, increasing the heat flow to the surface. The convection cells increase in size as the permeability is increased, but are constrained vertically by the confining layers, causing them to spread out horizontally, thus causing the peak surface heat flux to decrease again at high permeabilities.

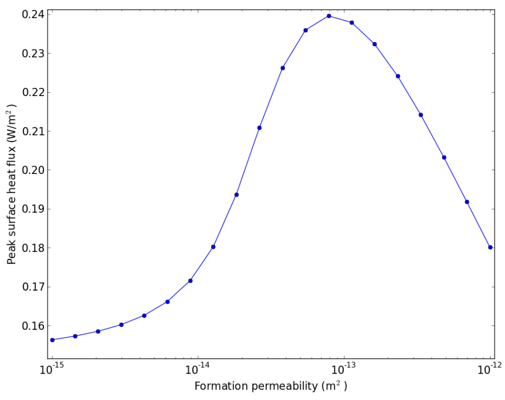

Figure 4: Effect of aquifer permeability on peak surface heat flux

We can also make use of the many extension libraries available for Python to carry out further analysis of the model results. For example, the exact aquifer permeability that gives the greatest peak surface heat flux can easily be found using one of the non-linear optimization algorithms from the Scientific Python library (http://www.scipy.org/). All that is required is to write a small Python function (by rearranging some of the code given above) that takes the permeability and returns the peak surface heat flux, and pass that function into the optimization routine. Doing this shows that the peak surface heat flux is maximized when the aquifer permeability is 0.83×10^{-13} m^2.

3.2 Lihir gold mine model

The PyTOUGH library was recently used to model the geothermal system in the Lihir gold mine, Papua New Guinea. More details of the Lihir geothermal system and the model can be found in O'Sullivan et al. (2011). The Lihir system presents particular modelling challenges, because the ground surface at the mine pit changes with time as the pit is excavated.

TOUGH2 has no explicit allowance for model domains or grids that change with time, so the long-term evolution of the system has to be simulated as a series of short periods over which the model geometry is held constant. For this study, the system was modelled from 1999 to 2023 in a sequence of one-year simulations, with a subset of the results of each yearly model run serving as initial conditions for the next.

A complex model of this sort cannot easily be managed using manual or GUI approaches. It requires at least some of the modelling process to be automated, and hence is a very suitable application for PyTOUGH.

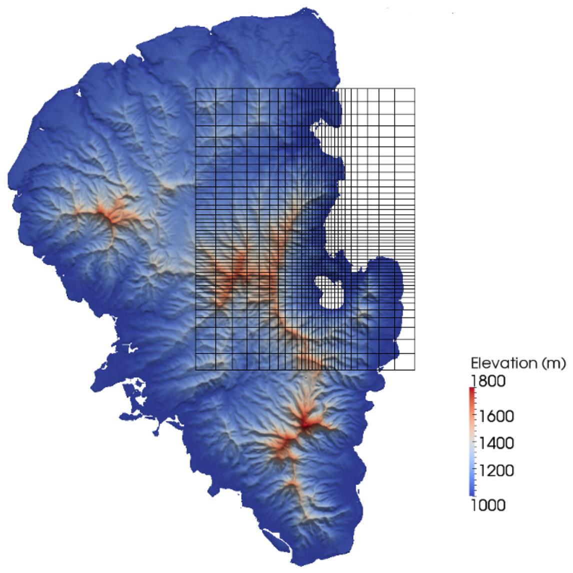

Figure 5: Lihir island, showing surface elevations, extent of model grid and location of mine pit

The first stage of the modelling process was to prepare a 3-D model grid geometry for each yearly model. An initial irregular rectangular 3-D grid geometry was prepared for the natural-state model, and this was used as the basis for each yearly grid geometry. The horizontal grid structure (shown in Figure 5) was kept constant, but at each year, the surface elevations over the grid columns in the pit were fitted anew to projected annual pit profiles provided by Newcrest Mining Ltd. This can all be done straight-forwardly in a few lines of Python script using a loop over the simulation years. Surface fitting at each year was carried out using the `fit_surface()` method of PyTOUGH's **mulgrid** class. This can fit scattered elevation data to an arbitrary unstructured horizontal grid, made up of triangular and/or quadrilateral cells, using least-squares finite element fitting. The method includes the option of applying Sobolev smoothing (Terzopoulos, 1986), which penalizes large gradients or curvature in the fitted surface, preventing the fitting process from either failing or producing spurious results for clumped data sets with areas of low data density.

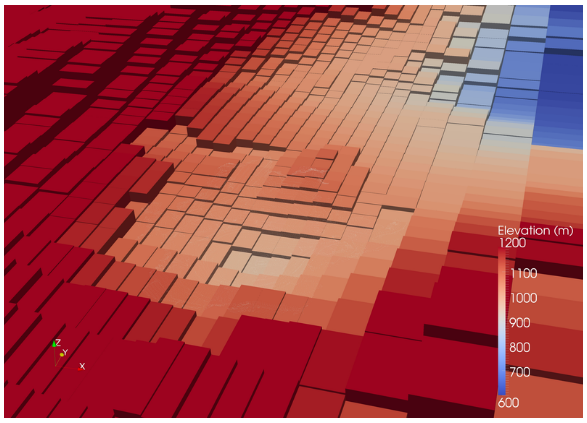

Figure 6: Modelled Lihir mine pit, natural state

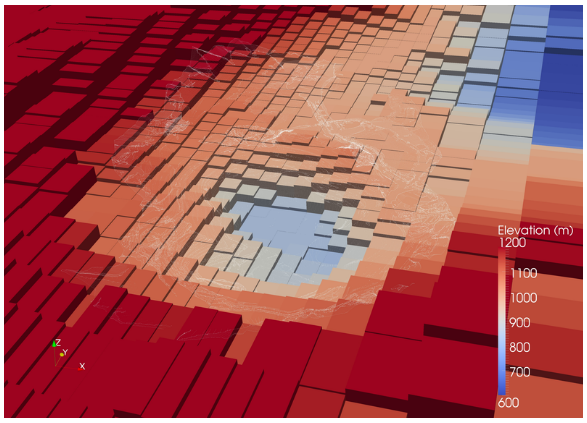

Figure 7: Modelled Lihir mine pit, 2011

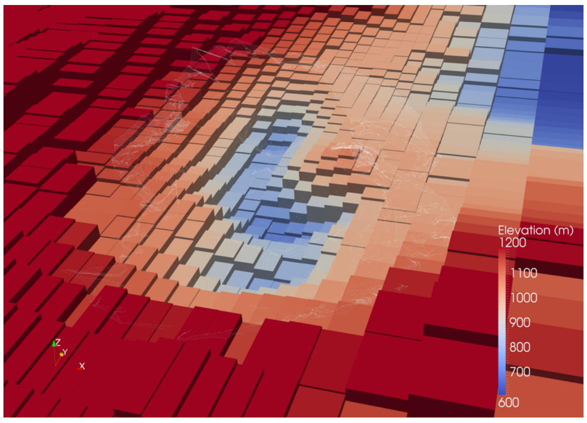

Figure 8: Modelled Lihir mine pit, 2023

Assembling the system of least-squares finite element equations requires finding which grid column each surface data point lies in. For large data sets and grids this geometric search is the most time-consuming part of the fitting process and has to be done as efficiently as possible. PyTOUGH includes an accelerated search algorithm based on a quadtree data structure, similar to that proposed by Krause and Rank (1996). In addition, the `fit_surface()` method allows the user to fit subsets of the whole grid, so for the Lihir model the process can be sped up further by only fitting columns within the mine pit.

The modelled mine pits for natural state, 2011 and 2023 are shown in Figures 6-8. The remaining stages of the modelling process consisted of:

- preparing a TOUGH2 grid and data input file for each yearly model, from its respective grid geometry- determining which blocks have been removed by mining (or added to mining stockpiles) and adjusting the heights of the uppermost blocks to follow the surface profile
- running the sequence of linked yearly models, with a subset of the results of each model being used as initial conditions for the next
- combining the results of all the yearly models into long-term results over the entire simulation period.

All of these stages were also carried out using PyTOUGH. Details of them can be found in O'Sullivan et al. (2011).

4. CONCLUSION

The examples above demonstrate some of the potential of a scripting approach to TOUGH2 simulations using the PyTOUGH library, as an alternative to, or in conjunction with, traditional manual or GUI approaches. Scripting can make model setup simpler and less error-prone in many cases (although it is of course still possible to make errors in a script), without limiting the potential of the TOUGH2 simulator. PyTOUGH also enables the user to perform post-processing and analysis of TOUGH2 results via a range of other Python extension libraries.

The real power of the scripting approach, however, lies in its ability to enable more complex modelling tasks that would be difficult or impossible to carry out manually or via a GUI- particularly running suites of models that have different sets of parameters, or that are linked to each other, with the input of one being determined by the output of another.

Rather than presenting the user with a pre-determined set of options, a scripting library like PyTOUGH provides a 'toolbox' of methods which can be combined, according to the needs of the modelling problem at hand, in an almost unlimited number of ways.

Some future applications of PyTOUGH that are envisaged include using it to populate rock types in a TOUGH2 data file by interfacing with geological modelling software. Simple inverse modelling, similar to what can be done with iTOUGH2 (Finsterle, 1999) could be done by coupling PyTOUGH with available Python libraries for non-linear optimization. Ensembles of models with stochastically generated parameters (e.g. rock properties), sampled from a distribution, could be run to perform sensitivity studies. Creating similar scripting libraries for other simulators would open up further possibilities, such as hybrid models involving more than one simulator- e.g. combining TOUGH2 with a rock mechanics code to model subsidence, similar to what was done (via other means) by Yeh and O'Sullivan (2007).

ACKNOWLEDGEMENTS

Some of the development of PyTOUGH was carried out to assist with work funded by Contact Energy, Newcrest Mining Ltd and the New Zealand Foundation for Research, Science and Technology.

REFERENCES

Audigane, P., Chiaberge, C., Mathurin, F., Lions, J., Picot-Colbeaux, G.: A workflow for handling heterogeneous 3D models with the TOUGH2 family of codes: applications to numerical modeling of CO2 geological storage. *Comp. Geosci.* 37 (4), 610-620 (2011).

Croucher, A.E., O'Sullivan, M.J.: Application of the computer code TOUGH2 to the simulation of supercritical conditions in geothermal systems. *Geothermics* 37, 622-634 (2008).

Finsterle, S. : *iTOUGH2 user's guide.* Report LBNL-40040, Lawrence Berkeley National Laboratory, Berkeley, California (1999).

Krause, R., Rank, E.: A fast algorithm for point-location in a finite element mesh. *Computing* 57, 49-62 (1996).

Langtangen, P.H.: *Python scripting for computational science.* Springer Verlag, New York (2008).

O'Sullivan, J.P, Croucher, A.E., O'Sullivan, M.J.: Modelling the evolution of a mine pit in a geothermal field at Lihir Island, Papua New Guinea. *Proc. 33rd NZ Geothermal Workshop*, Auckland, NZ. (2011).

O'Sullivan, M.J., Bullivant, D.: A graphical interface for the TOUGH2 family of simulators. In: *Proc. TOUGH Workshop 1995,* Berkeley, Calif., 90-95 (1995).

Pruess, K. : The TOUGH2 codes- a family of simulation tools for multiphase flow and transport processes in permeable media. *Vadose Zone Journal* 3(3), 738 (2004).

Terzopoulos, D.: Regularization of inverse visual problems involving discontinuities. *IEEE Trans. Pattern Analysis & Machine Intelligence* PAMI-8 (4), 413-424 (1986).

Yamamoto, H. : PetraSim: a graphical user interface for the TOUGH2 family of multiphase flow and transport codes. *Ground Water*, 46(4), 525-528 (2008).

Yeh, A., O'Sullivan, M.J. : Computer modelling of subsidence in geothermal fields. *Proc. 29th NZ Geothermal Workshop*, Auckland, NZ (2007).