Tim$^{\mathrm{ML}}$

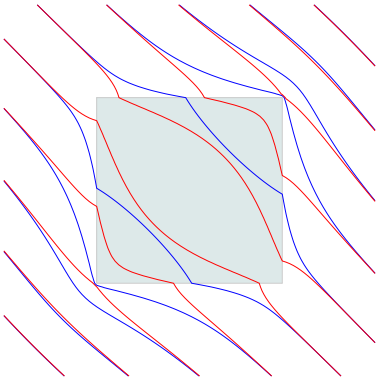
A Multiaquifer Analytic Element Model

Version 4.0

DRAFT

Mark Bakker
Water Resources Section, Civil Engineering and Geosciences
Delft University of Technology, Delft, The Netherlands
mark.bakker@tudelft.nl

August 20, 2015



ARTHUR: What manner of man are you that can
summon up fire without flint or tinder?
TIM: I... am an enchanter.
ARTHUR: By what name are you known?
TIM: There are some who call me ... Tim.
ARTHUR: Greetings, Tim the Enchanter.
Scene 32, Monty Python and the Holy Grail

Contents

1 Introduction

Tim$^{\mathrm{ML}}$ is a computer program for the simulation of steady-state multiaquifer flow with analytic elements and consists of a library of Python scripts and FORTRAN extensions. *Tim*$^{\mathrm{ML}}$ may be applied to an arbitrary number of aquifers and leaky layers. The head, flow, and leakage between aquifers may be computed analytically at any point in the aquifer system. The design of *Tim*$^{\mathrm{ML}}$ is object–oriented and has been kept simple and flexible. New analytic elements may be added to the code without making any changes in the existing part of the code. *Tim*$^{\mathrm{ML}}$ is coded in Python, a free and open-source programming language; occasional use is made of FORTRAN extensions to improve performance.

Tim$^{\mathrm{ML}}$ includes many multiaquifer analytic elements including (but not limited to): Constant, Uniform flow, Wells, Strength–specified linesinks, Head–specified linesinks, Resistance linesinks, Impermeable walls, Circular areasinks, Polygonal areasinks, Cylindrical inhomogeneities, Polygonal inhomogeneities, and Embedded multiaquifer domains. Most elements may be located or screened in multiple layers or aquifers. Utilities have been written for the creation of contour plots and path line plots using the matplotlib package. Graphical output includes many formats including pdf, eps and png. Alternatively, grids and layouts may be exported to Surfer[1] or MATLAB[2]. Three-dimensional path lines may be exported into files that can be imported into the same programs.

1.1 Approximations

Tim$^{\mathrm{ML}}$ is based on an analytic element formulation for multiaquifer flow with a number of restrictions. Aquifer parameters and leaky layer parameters must be piecewise constant. Flow is at steady-state. Unconfined flow in the top aquifer may be approximated by specifying an average transmissivity. Vertical flux between model layers is computed as the head difference divided by the vertial resistance. Hence, it is assumed that flow remains semi-confined in all layers, except for in the top layer (i.e., the head in layer n is higher than the top of layer n). The resistance to vertical flow is neglected within an aquifer, but flow is three–dimensional; the vertical component of flow in an aquifer is obtained from three–dimensional continuity of flow. Flow in leaky layers is approximated as vertical.

1.2 MIT License

Tim$^{\mathrm{ML}}$ is distributed under the MIT License.

[1] Surfer is sold by Golden Software, www.goldensoftware.com
[2] MATLAB is sold by The MathWorks, www.mathworks.com

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.3 Introduction for MLAEM users

MLAEM[3] is currently the only other available multiaquifer analytic element model. The main difference between *Tim*$^{\mathrm{ML}}$ and MLAEM is that in *Tim*$^{\mathrm{ML}}$ leakage between aquifers is simulated automatically and exactly without the need for specifying area-elements for leakage. One practical difference is that *Tim*$^{\mathrm{ML}}$ allows for the specification of one reference point in one aquifer only, while MLAEM requires the specification of one reference point in every aquifer.

1.4 Introduction for MATLAB users

There are a lot of similarities between Python and MATLAB. Both may be run from a command prompt and are interactive. Three major differences are important when running *Tim*$^{\mathrm{ML}}$. (1) The object–oriented programming capabilities of Python are much larger and more elegant than MATLAB. As a practical consequence, a Python method that has been stored in a file can only be called if the file is first imported (MATLAB often calls methods by looking up file names). (2) Variables are not automatically matrices in Python. The numpy package offers the ability to define matrices (called arrays). Array multiplication is by default term by term. Commands are often similar or identical to corresponding commands in MATLAB. For example, the function eig(A) returns the eigenvalues of A. (3) Python is free and open–source. For more details on the differences between numpy and MATLAB, see http://www.scipy.org/NumPy_for_Matlab_Users.

1.5 Introduction for MODFLOW users

A comprehensive introduction into multiaquifer modeling with analytic elements (and especially *Tim*$^{\mathrm{ML}}$) for MODFLOW users is planned here. For now it is limited to explaining the relationship between the resistance c of an aquitard (used in *Tim*$^{\mathrm{ML}}$) and the variable Vcont used in MODFLOW models. The resistance c is the reciprocal of Vcont: $c = 1/\mathrm{Vcont}$.

1.6 Differences with Version 3 and before

Tim$^{\mathrm{ML}}$ version 4 is not compatible with previous versions. The most important changes are:

- *Tim*$^{\mathrm{ML}}$ is now imported as `timml` (so no more capitals).
- The `Model` class does not take the number of aquifer as input anymore. The number of aquifers is determined by *Tim*$^{\mathrm{ML}}$ from the length of the `zb` array.
- All layer numbering starts at zero at the top layer.

[3] MLAEM is sold by Strack Consulting, www.strackconsulting.com

1.7 Release History

Version 0.1. January, 2002. First beta release.

Version 1.0. May, 2002. First full release

Version 1.1. January 2003. Improvement of speed of aquifer system inhomogeneities (they are now 5-10 times faster), conversion to Python 2.2, and inclusion of a large number of hooks, including an experimental iterative solver, to prepare for inclusion of circular and elliptical inclusions in future versions.

Version 1.2. July 2003. Added Uflow element, resistance line-sinks, and Model3D command. Fixed inaccuracy with circular area-sinks regarding large radii. Build in more hooks for version 2 release.

Version 2.0. January 2004. Added ditches, polygonal area-sinks, cylindrical inhomogeneities, the possibility to model semi-confined aquifer systems, and three-dimensional path line tracing.

Version 2.1. August 2004. Improved speed, improved accuracy and efficiency of tracing algorithm, improved aquifer system inhomogeneities; not yet operational for semi-confined flow.

Version 2.1.1. January 2005. Bug fix for resistance line-sinks.

Version 3.0.alpha1. February 2006. Addition of polygonal inhomogeneities with the same number of aquifers inside and outside. Semi-confined flow has been disabled as it will be replaced by an entirely new formulation before the full 3.0 release. Aquifer system inhomogeneities may be added with the AquiferSystemInhomogeneity command; the old MakeInhomogeneity command will be phased out.

Version 3.0. February 2007. Switch to numpy and Python 2.4. Semi-confined flow option permanently removed; a better replacement is in the works. Number of fixes in trace routine. New webpage. Accuracy improvements FORTRAN extension. FORTRAN extension compiled with GNU compiler.

Version 3.1. July 2007. Percolating resistance line-sinks and some bug fixes.

Version 3.2. August 2008. Impermeable walls and some bug fixes. Simultaneous release of versions for Python 2.4 and 2.5.

Version 3.3.alpha. October 2008. Replaced MakeInhomSide by MakeInhomPolySide. Automatically checks for aqin and aqout. PolyInhomData checks for clockwise or counter-clockwise. New element Lake, including partially percolating bottom; needs installation of matplotlib and shapely. Interactive plotting with timcontour; needs installation of matplotlib.

Version 3.3. May 2009. The new Lake element had been tested and works well with all elements. Restriction is still that the entire outside of the lake needs to be the background aquifer. This restriction will be removed in the next release.

Version 3.4. March 2010. Added interactive graphical output using the matplotlib package. Updated the manual to use IPython as the preferred Python shell. Updated the new installation requirements.

Version 4.0.alpha. April 2015. Layers numbering starts at zero.

1.8 Acknowledgement

Development of *Tim*$^{\mathrm{ML}}$ was funded in part by:

- Intera, Inc., Austin, TxX
- Ecosystems Research Division, United States Environmental Protection Agency, Athens, GA
- WHPA, Bloomington, IN, a Layne Christensen company.
- Amsterdam Water Supply, Brabant Water, Waterbedrijf Limburg, in collaboration with Artesia.
- United States Bureau of Reclamation.

Dr. Stephen Kraemer of USEPA assisted in beta–testing and comparisons with MODFLOW.
Dr. Vic Kelson of WHPA co-developed the original design and has been providing feedback during the entire development process of *Tim*$^{\mathrm{ML}}$.
Dr. Philippe LeGrand contributed the script for generating windows installers.
Frans Schaars of Artesia extensively tested the trace routine through application to practical problems.

1.9 Plans

There is a long list of plans for the expansion of *Tim*$^{\mathrm{ML}}$. Implementation of the different ideas depends on available time, funds, and inspiration. If you want to contribute to the development of *Tim*$^{\mathrm{ML}}$, either through code development or through financial support, please contact Mark Bakker at markbak@gmail.com. Several of the listed elements are already in some stage of development. Plans for *Tim*$^{\mathrm{ML}}$ include:

- Lakes that can be connected or (partially) inside inhomogeneities.
- Leaky walls and leaky faults.
- Elliptical inhomogeneities with an arbitrary number of aquifers and leaky layers both inside and outside the inhomogeneity. They have been developed and implemented fully but have not been released as they require an additional Python extension in C++, which cannot be pickled. When interested, email the developer.
- Local percolating conditions (now only along ResLineSinks and Lakes).
- Nested inhomogeneities (one inhomogeneity or lake inside another inhomogeneity).
- Improvement of computational speed.
- Improvement of accuracy of short high-order elements.
- Better interfacing with existing GUIs and possible linkage with an existing open–source GUI.
- Addition of utilities to do contaminant transport.
- GUI for postprocessing using Matplotlib (prototype done in Tkinter).

1.10 *Tim*$^{\mathbf{ML}}$ website

Tim$^{\mathrm{ML}}$ code is hosted on `github`

2 Installation

The following free, open-source software is required to run *Tim*$^{\mathrm{ML}}$: Python version 2.X, the packages numpy, scipy, and matplotlib. The shapely package is needed for the use of Lake elements (optional). *Tim*$^{\mathrm{ML}}$ uses a FORTRAN extension which means that *Tim*$^{\mathrm{ML}}$ is unfortunately both platform dependent and Python version dependent.

Although it is possible to download all required Python packages separately, it is much easier to install one of the Python installers that comes with a large selection of popular packages. Three main options are Anaconda, Enthought, and PythonXY. This manual is written for Anaconda users. Use of the other distributions only differs in how IPython is started.

3 Application

It may be beneficial (actually it is highly recommended) to spend some time going through an introductory Python tutorial if you are not familiar with Python; many free tutorials are available from the Python website. An introduction to Python for Exploratory computing can be found at: `http://mbakker7.github.io/exploratory_computing_with_python/`

Application of *Tim*$^{\mathrm{ML}}$ requires a certain intelligence from the user. Very few assertions are implemented in the code to verify that valid input variables are entered. (The only ones that are checked correspond to errors that the developer kept making or requests from users.) When *Tim*$^{\mathrm{ML}}$ returns an error, read the error carefully. If the last line of the error message starts with something like `AssertionError: TimML Input error:` then this is an assertion error indicating you are trying to enter invalid data and instructions are given on what you should do.

3.1 Starting a model

In this manual, *Tim*$^{\mathrm{ML}}$ is run from the IPython prompt. Start the Anaconda Launcher and launch the IPython-Qtconsole. At the Python command prompt, import `timml` ('`In []:`' is used for the Python prompt in this manual; IPython adds a input number between the square brackets):

```
In []:  from timml import *
```

It may take a couple of seconds to load everything. If you have not installed *Tim*$^{\mathrm{ML}}$ correctly, you will get the following error message:

```
In []: from timml import *
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-2-9b20be32f415> in <module>()
----> 1 from timml import *

ImportError: No module named timml
```

When you create an input file, the line `from timml import *` must be the first line of your file. The next command must be the creation of a model, which must be given a name, for example `ml`. All elements will be added to this model.

`In []: ml = Model(k,zb,zt,c,n=[],nll=[])` where

- `k` is a list[4] of permeabilities of the aquifers starting from the top down.
- `zb` is a list of bottom elevations of the aquifers from the top down (Note: this may be counter intuitive, so be careful.) The number of aquifers `Naquifers` is determined from the length of `zb`.
- `zt` is a list of top elevations of the aquifers from the top down
- `c` is a list of resistances (dimension: time) of separating layers between aquifers. The list starts with the bottom of aquifer 1 and is thus `Naquifers`-1 long.
- `n` is a list with the porosities of the aquifers. If it is not entered, it is set to 0.3.

[4] A list is a sequence separated by commas and between square brackets, such as: [1,2,3]

- `nll` is a list with the porosities of the leaky layers. If it is not entered, it is set to 0.3.

Note that aquifers are numbered from the top down in *Tim*$^{\mathrm{ML}}$, starting with number 0. Also note that any parameters in the argument list that are followed by an equal sign (such as `n=[]`) are optional parameters. Alternatively, when 3D flow is modeled, a model may be constructed with the command

`In []: ml = Model3D(z=[1,0.5,0],kh=1.0,kzoverkh=1.0)` where

- `z` is a list with the top elevations of all the layers (from the top down) that an aquifer is divided into; the last value is the bottom of the aquifer.
- `kh` is the horizontal hydraulic conductivity of the aquifer and may either be one value or a list/tuple/array with a value for each layer.
- `kzoverkh` is the vertical anisotropy ratio k_z/k_h and may either be one value or a list/tuple/array with a value for each layer.

3.2 Adding analytic elements

The following elements may be added to the model:

`In []: Constant(modelParent,xr,yr,head,layer,label=None)` where

- `modelParent` is the model to which the element is added
- `xr,yr` is the location of the reference point
- `head` is the head at the reference point
- `layer` is the number of the aquifer in which the reference point is applied
- `label` is an optional character string representing the label of the element (if you want to access this element later, it may be hard if you don't give it a label)

Only one `Constant` element may be added to a model[5]. It is not checked in *Tim*$^{\mathrm{ML}}$ whether more than one `Constant` is added to the model[6].

`In []: Uflow(modelParent,grad,angle,label=None)` where

- `modelParent` is the model to which the element is added
- `grad` is the gradient of the head (positive)
- `angle` is the direction of uniform flow, in degrees measured counter-clockwise with respect to positive x (East).
- `label` is an optional character string representing the label of the element.

`In []: Well(modelParent,xw,yw,Qw,rw,layers,label=None)` where

- `modelParent` is the model to which the element is added
- `xw,yw` is the location of the well

[5]This is different than in MLAEM, where a constant must be specified for every layer.
[6]This is one of those places where the intelligence of the user is expected.

- `Qw` is the discharge [L^3/T] of the well (positive for taking water out)
- `rw` is the radius of the well
- `layers` is the number of the model layer in which the well is screened or a list of layer numbers when the well is screened in multiple layers.
- `label` is an optional character string representing the label of the element.

When the well is screened in more than one layer (a multi–aquifer well), the discharge of the well is divided over the aquifers such that the heads in the aquifers are equal.

`In []: LineSink(modelParent,x1,y1,x2,y2,sigma,layers,label=None)` where

- `modelParent` is the model to which the element is added
- `x1,y1,x2,y2` are the left and right end points of the line-sink
- `sigma` is the strength [L^2/T] of the line-sink per unit length of line-sink (positive for taking water out); the strength is constant along the element.
- `layers` is the number of the model layer in which the line-sink is screened or a list of layer numbers when the line-sink is screened in multiple layers.
- `label` is an optional character string representing the label of the element.

When the line-sink is screened in more than one aquifer, the strength of the line-sink is divided over the aquifers such that the heads in the aquifers are equal. Note that if the strength is zero, this means that you are modeling a highly conductive, vertical fault zone.

```
In []: HeadLineSink(modelParent,x1,y1,x2,y2,head,layers,label=None)
```

- `modelParent` is the model to which the element is added
- `x1,y1,x2,y2` are the left and right end points of the line-sink
- `head` is the head at the center of the line-sink
- `layers` is the number of the model layer in which the line-sink is screened or a list of layer numbers when the line-sink is screened in multiple layers.
- `label` is an optional character string representing the label of the element.

```
In []: ResLineSink(modelParent,x1,y1,x2,y2,head,res,width,layers=[1],label=None,bottomelev=None)
```

- `modelParent` is the model to which the element is added
- `x1,y1,x2,y2` are the left and right end points of the line-sink
- `head` is the head at the center of the line-sink
- `res` is the resistance of the bottom of the line-sink against in/outflow.
- `width` is the width of the line-sink
- `layers` is the number of the model layer in which the line-sink is screened or a list of layer numbers when the line-sink is screened in multiple layers.
- `label` is an optional character string representing the label of the element.

- `bottomelev` is the elevation of the bottom of the (leaky) resistance layer at the bottom of the line-sink. This value is only used to check whether there is a hydraulic connection between the computed head and the bottom of the leaky layer. When such conditions may occur, the solution needs to be computed iteratively using the command `ml.solve(doIterations=True)`.

`In []: LineSinkDitch(modelParent,xylist,Q,res,width,layers=[1],label=None)`

- `modelParent` is the model to which the element is added
- `xylist` is a list of (x, y) pairs of the nodes of the line-sink ditch, such as [(0,0),(10,0),(10,10),(0,10)].
- `Q` is the total discharge [L^3/T] of the ditch. The head in the ditch will be constant, but is *a priori* unknown.
- `res` is the resistance of the ditch against in/outflow.
- `width` is the width of the ditch
- `layers` is the number of the layer in which the ditch is screened.
- `label` is an optional character string representing the label of the element.

`In []: LineDoubletImp(modelParent,x1,y1,x2,y2,order=0,layers=[1],label=None)`

- `modelParent` is the model to which the element is added
- `x1,y1,x2,y2` are the left and right end points of the section of impermeable wall.
- `order` is the order of the solution. The no-flow condition is applied at order+1 points along the wall segment. *Order cannot be higher than 8* for the current implementation. Note: these elements are not magic. To obtain an accurate solution it may be necessary to discretize the wall in multiple sections.
- `layers` is the number of the model layer in which the impermeable wall is screened or a list of layer numbers when the impermeable wall is screened in multiple layers.
- `label` is an optional character string representing the label of the element.

`In []: CircAreaSink(modelParent,xp,yp,Rp,infil,layer,label=None)` where

- `modelParent` is the model to which the element is added
- `xp,yp` is the center of the circular area-sink
- `Rp` is the radius of the area-sink
- `infil` is the areal infiltration rate [L/T] of the area-sink (positive for putting water into the aquifer)
- `layer` is the number of the model layer to which the area-sink is added.
- `label` is an optional character string representing the label of the element.

`In []: PolyAreaSink(modelParent,xylist,infil,label=None)` where

- `modelParent` is the model to which the element is added

- `xylist` is a list of (x, y) pairs of the nodes of the area-sink boundary given in counter clockwise direction (this is not checked by the model!), such as [(0,0),(10,0),(10,10),(0,10)]; the model will close the polygon automatically (so don't give the begin node again at the end)
- `infil` is the areal infiltration rate [L/T] of the area-sink (positive for putting water into the aquifer). The infiltration rate is constant over the area-sink.
- `label` is an optional character string representing the label of the element.

A polygonal area-sink is automatically added to layer 0. The sides of a polygonal area-sink are exact (and hence, the lengths of the segments that make up the boundary don't effect the accuracy of the solution as they do for PolygonInhom or Lake elements, for example).

`In []: Lake(ml,xylist,hlake,cbot,Hbot,nbot=0.3,Lmax=None,areatol=0.2)` where

- **Lake** elements are currently not included in TimML
- `ml` is the model to which the lake is added
- `xylist` is a list of (x, y) pairs of the corners of the polygon that bounds the lake. The first point does not have to be repeated to close the polygon; it is closed automatically.
- `hlake` is the uniform water level in the lake.
- `cbot` is the resistance of the bottom of the lake
- `Hbot` is the thickness of the bottom of the lake, i.e., the vertical hydraulic conductivity of the lake sediment is Hbot/cbot.
- `nbot` is the porosity of the lake bottom; it is set to 0.3 by default.
- `Lmax` is the maximum length of a side. When it is not None, the provided list of (x, y) values is expanded such that each segment of the boundary is at most `Lmax` long.
- `areatol` is the tolerance used to determine whether convergence is achieved when iterating for the size of the percolating area. The default value is 0.2 and it rarely if ever needs to be smaller.

Lakes have a fixed and uniform water level, a resistance at the bottom, and are bounded by a polygon. Lakes are added automatically to model layer 0. Lakes are modeled using the approach described in Bakker (2007). Lakes may not share boundaries with other lakes and may not be inside or overlap inhomogeneities. The boundary of the lake is made-up of second-order line elements. Boundary segments must be made sufficiently short to obtain an accurate solution. The accuracy of the solution may be verified visually: both the head contours must be continuous across the boundary and the tangent to the head contours must be continuous across the boundary (since the aquifer properties don't change from outside the lake to below the lake). An example of an inaccurate solution is given in the Examples section, so you know what to look for.

3.3 Inhomogeneous aquifer properties

Three kinds of inhomogeneities may be added: polygonal inhomogeneities, cylindrical inhomogeneities, and aquifer system inhomogeneities.

3.3.1 Multi-aquifer polygonal inhomogeneities

Multi-aquifer polygonal inhomogeneities need to have the same number of aquifers inside and outside the polygon. Model layer 0 on the inside is connected to model layer 0 on the outside, etc. Properties of all aquifers and leaky layers may differ between the inside or the outside, but they don't have to: if you want to change only the transmissivity of aquifer 2 and leave the rest the same, that is fine too, but the modeling process will be the same. Inhomogeneities may share boundaries. Inhomogeneities may be nested inside other inhomogeneities, but they can only be modeled if you enter the smaller inhomogeneity *before* the larger inhomogeneity.

When modeling an inhomogeneity, aquifer data for each inhomogeneity must be entered first:

`In []: PolygonInhom(modelParent,Naquifers,k,zb,zt,c,xylist,n=[],nll=[])` where

- `modelParent` is the model to which the aquifer data is added
- `k` is a list of permeabilities of the aquifers starting from the top down.
- `zb` is a list of bottom elevations of the aquifers from the top down (Note: this may be counter intuitive, so be careful.)
- `zt` is a list of top elevations of the aquifers from the top down
- `c` is a list of resistances (dimension: time) of separating layers between aquifers, starting with the bottom of aquifer 0. Hence, the length of this list is `Naquifers`-1.
- `xylist` is a list of (x, y) pairs of the nodes of the inhomogeneity boundary (can be either clockwise or counter clockwise); the model will close the polygon automatically (so don't give the begin node again at the end)
- `n` is a list with the porosities of the aquifers. If it is not entered, it is set to 0.3.
- `nll` is a list with the porosities of the leaky layers. If it is not entered, it is set to 0.3.

Sides of an inhomogeneity are added as polylines. Each polyline must have one aquifer system on the left side and one aquifer system on the right side. Line-elements that are put along the polyline will have the same order. When the aquifer on one of the sides changes, or you want a different order of the line elements, you must start a new inhomogeneity side. The command for adding elements along an inhomogeneity side is

`In []: MakeInhomPolySide(ml,xylist,order,closed=False):` where

- `modelParent` is the model to which the element is added
- `xylist` is a list of (x, y) pairs of the nodes of the polyline that is the (part of the) boundary of an inhomogeneity. The aquifer on the left side of the polyline (going from point 1 to 2 to 3 etc.) must be the same for all segments of the polyline. Similarly, the aquifer on the right side of the polyline (going from point 1 to 2 to 3 etc.) must be the same for all segments of the polyline. This boundary must coincide with the boundary of an inhomogeneity specified with PolygonInhom, although it may have more nodes along each side, i.e., one straight segment of PolygonInhom may be subdivided into multiple segments with MakeInhomPolySide, as long as they all lie on the same straight line.

- `order` is the order of the line elements that are put along the polyline. Maximum order is 8.
- `closed` is a flag that is `False` by default. When it is `True` (or 1), the polyline will be closed (so don't repeat the first point of the polyline again, just set the `closed` to `True`

It is essential that sides are added along the entire inhomogeneity boundary. When inhomogeneities share boundaries, don't repeat the shared boundary. Note that this function replaces the deprecated `MakeInhomSide`, which was included in versions prior to 3.3. It worked the same except that the aquifers on the left and right sides had to be provided as arguments[7].

3.3.2 Cylindrical inhomogeneities

Cylindrical inhomogeneities intersect all aquifers. the number of aquifers and leaky layers must be the same inside and outside the cylinder. Both the transmissivity of the aquifers and the resistance of leaky layers may be different between the inside and the outside of the cylinders. *May* is the operative word here as they don't have to be different (just like for polygonal inhomogeneities). For example to model a sandy hole in a leaky clay layer you can change the resistance of one leaky layer only and leave all other resistances and transmissivities the same. To enter a cylindrical inhomogeneity, you have to define the aquifer data on the inside of the cylinder first

`In []: inhom1 = CircleInhomData(modelParent,k,zb,zt,c,xc,yc,R,n=[],nll=[])` where

- `modelParent` is the model to which the aquifer data is added
- `k` is a list of permeabilities of the aquifers starting from the top down.
- `zb` is a list of bottom elevations of the aquifers from the top down (Note: this may be counter intuitive, so be careful.)
- `zt` is a list of top elevations of the aquifers from the top down
- `c` is a list of resistances (dimension: time) of separating layers between aquifers, starting with the bottom of aquifer 1. Hence, the length of this list is `Naquifers`-1.
- `xc,yc` is the center of the cylinder
- `R` is the radius of the cylinder
- `n` is a list with the porosities of the aquifers. If it is not entered, it is set to 0.3.
- `nll` is a list with the porosities of the leaky layers. If it is not entered, it is set to 0.3.

CircleInhomData should be given a name (`inhom1` in the input line above) so that it can be used to define the inside of an inhomogeneity.

Then a cylindrical inhomogeneity may be added:

`In []: CircleInhom(modelParent,order,aqin,aqout,label=None)` where

- `modelParent` is the model to which the element is added
- `order` is the number of terms that is used in the series expansion (the more the more accurate and the slower the solution).

[7] This lead to too many input errors, so it is now figured out by TimML

- `aqin` is the `CircleInhomData` on the inside of the inhomogeneity (that would be `inhom1` in the example above).
- `aqout` is the aquifer data on the outside of the inhomogeneity. This is the background aquifer data defined with `Model`. For example, if the model is called `ml` then the aquifer data on the outside is `ml.aq` (`aq` is an attribute of `Model`).
- `label` is an optional character string representing the label of the element.

Warning: cylindrical inhomogeneities may become inaccurate when the radius is very large compared to the leakage factor. This can be fixed, but has not been implemented.

3.3.3 Aquifer system inhomogeneities

Aquifer system inhomogeneities are not included in the current version of TimML. Aquifer system inhomogeneities have either *one* aquifer outside (defined with `Model`) and an arbitrary number of aquifers and leaky layers inside or vice versa. Aquifer system inhomogeneities may not share boundaries. To enter an aquifer system inhomogeneity, first enter aquifer data for the inhomogeneity:

`In []: inhom1 = PolygonInhom(modelParent,Naquifers,k,zb,zt,c,xylist,n=[],nll=[])` where

- `modelParent` is the model to which the aquifer data is added
- `Naquifers` is the number of aquifers
- `k` is a list of permeabilities of the aquifers starting from the top down.
- `zb` is a list of bottom elevations of the aquifers from the top down (Note: this may be counter intuitive, so be careful.)
- `zt` is a list of top elevations of the aquifers from the top down
- `c` is a list of resistances (dimension: time) of separating layers between aquifers, starting with the bottom of aquifer 1. Hence, the length of this list is `Naquifers`-1.
- `xylist` is a list of (x, y) pairs of the nodes of the inhomogeneity boundary given in counter clockwise direction (this is not checked by the model!), such as [(0,0),(10,0),(10,10),(0,10)]; the model will close the polygon automatically (so don't give the begin node again at the end)
- `n` is a list with the porosities of the aquifers. If it is not entered, it is set to 0.3.
- `nll` is a list with the porosities of the leaky layers. If it is not entered, it is set to 0.3.

PolygonInhom should be given a name (`inhom1` in the input line above) so that it can be used to define the inside of an inhomogeneity. Then an inhomogeneity may be added:

`In []: AquiferSystemInhomogeneity(modelParent,xylist,aqin,aqout)` where

- `modelParent` is the model to which the element is added
- `xylist` is a list of (x, y) pairs of the nodes of the inhomogeneity boundary given in counter clockwise direction (this is not checked by the model!), such as [(0,0),(10,0),(10,10),(0,10)]; the model will close the polygon automatically (so don't give the begin node again at the end). This boundary must coincide with the boundary of `aqin`, although it may have more (or less) nodes along each side. For aquifer system inhomogeneities, the order of the line elements is fixed to order two.

- `aqin` is the `PolygonInhom` on the inside of the inhomogeneity (that would be `inhom1` in the example above).
- `aqout` is the aquifer data on the outside of the inhomogeneity. This is the background aquifer data defined with `Model`. For example, if the model is called `ml` then the aquifer data on the outside is `ml.aq` (`aq` is an attribute of `Model`).

3.4 Running the model

In this section it is assumed that the model is called `ml`. Once all analytic elements are entered, a solution is obtained by typing

```
In []:  ml.solve()
```

Don't forget the `()`, since `solve` is a method of the `Model` class. To verify whether the solution fulfills the boundary conditions at all control points, type

```
In []:  ml.check()
```

The head at a point (x, y) in layer ilayer may be computed by typing

```
In []:  ml.head(ilayer,x,y)
```

or better yet, to get the heads in all layers at point (x, y) (this takes the same amount of computation time as one layer) type

```
In []:  ml.headVector(x,y)
```

When modeling 3D flow, and you are not really keeping track of layers, you can evaluate the head at any 3D point with

```
In []:  ml.head3D(x,y,z)
```

Note that `head3D` does not give a head value inside a leaky layer.

It is important to know the characteristic lengths of the aquifer system, called the leakage factors, which have dimensions of length. In general, the leakage between aquifers (or layers) approaches zero three times the largest leakage factor away from any analytic element (in the absence of areal recharge). The leakage factors are a function of the aquifer and leaky layer parameters only. To find out what they are for your model, type

```
In []:  ml.aq.lab
```

or if you want to know what they are inside an inhomogeneity with aquifer data `pinhom` type

```
In []:  pinhom.lab
```

Tim$^{\mathrm{ML}}$ allows you to have several models open at the same time in the same Python window. For example one model could be called `mlone` and the other model `mltwo`. The difference in head at a point between the two models may then be computed by typing[8]

```
In []:  mlone.headVector(x,y)-mltwo.headVector(x,y).
```

3.4.1 Three-dimensional path line tracing

To trace a pathline, use the command

```
In []:  [xyz,t,end,layerp]=traceline(ml,xstart,ystart,zstart,stepin,tmax=1e30,maxsteps=10,
tstart=0.0,window=[-1e30,-1e30,1e30,1e30],labfrac=2.0, Hfrac=2.0))
```

which returns a 3D array `xyz` with the (x, y, z) locations of the points along the trace line, a 1D array `t` with the time at each point, a list `end` with three items: reason why traceline ended, label of element at which element ended (if it ended at an element; if element doesn't have label this contains None), and total time it took to reach final point, and finally a list with layer numbers of all the points along the traceline (handy for visualization). Specify a reasonable (space) stepsize with the `stepin` variable. When close to an element, the stepsize is automatically reduced if it is larger than λ_{min}/`labfrac`, such that an accurate pathline is computed; this is especially important when doing a 3D model where leakage factors may be very small. In

[8] Admit it, you are falling in love with Python

addition, the stepsize is automatically reduced when the vertical step is larger than H/`Hfrac`.

3.5 Graphical output

In the IPython console, the command `%matplotlib` must be given to allow interactive plotting with matplotlib. For inline figures use `%matplotlib inline`, otherwise, specify the backed of your choice, for example `%matplotlib qt`.
The following commands are available for interactive graphical output in *Tim*$^{\mathrm{ML}}$:

```
In []:  timcontour( ml, xmin, xmax, nx, ymin, ymax, ny, layers=0, levels = 10, color = None,
width = 0.5, style = '-', separate = 0, layout = 1, newfig = 1, labels = 0, labelfmt = '%1.2f',
xsec = 0, returnheads = 0, returncontours = 0, fillcontour = 0, size=None, pathline = False)
```

The required arguments are

- `ml` is the model name
- `xmin,xmax` are the minimum and maximum x values of the contour window
- `nx` is the number of points in the x direction
- `ymin,ymax` are the minimum and maximum y values of the contour window
- `ny` is the number of grid points in the y direction

Optional keyword arguments are

- `layers` is used to specify the aquifers for which contours are drawn. `layers` may be specified as
  - `'all'` contours all aquifers
  - A number, contour all model layers up to this number (i.e., `layers=3` contours the first three layers, which are numbers 0, 1, and 2.)
  - A list with numbers. Contour only the model layer numbers in the list
- `levels` is used to specify head values to contour. There are four options:
  - `'ask'`. Ask *Tim*$^{\mathrm{ML}}$ to print the minimum and maximum head values to the screen after which the minimum, maximum, and step needs to be entered.
  - `[hmin,hmax,step]` A list with the minimum, maximum and step size of the head contours you want to see
  - `number`. Show `Number` contours and let *Tim*$^{\mathrm{ML}}$ figure out which values to contour.
  - `array`. Show contours for the head values in the array.
- `color` is the color of the contours. There are three options:
  - A string, either 'b','g','r','c','m','y', or 'k', for blue, green, red, cyan, magenta, yellow or black. All contours have the same color.
  - A list with these colors. Use the fist item in the list for the first aquifer that is contoured, the second item for the second aquifer, etc.

  - `None`. If nothing is specified, the colors of contours in the first 5 aquifers are 'b', 'r', 'g', 'm', and 'c', and this sequence is repeated for any aquifers after that.

- `width` the line width of all contours.
- `style` the line style (`'-'` is a solid line).
- `separate` If this keyword is true, all aquifers are plotted in separate figures, otherwise they are all shown in the same figure.
- `layout` A layout of all elements is shown if this keyword is true.
- `newfig` A new figure is created when this keyword is true. Otherwise, the contours are added to the active figure.
- `labels` Labels are shown on the contours if this keyword is true.
- `labelfmt` Is the format of the numbers of the labels. The default `'%1.2f'` means two numbers behind the decimal.
- `xsec` A cross-sectional view is added below the contour plot if this keyword is true.

```
In []:  timlayout(ml)
```
where `ml` is the model name. This commands plots a layout of all the elements to a new figure and chooses the limits of the axes such that all elements fit in the figure. After this, you can use the zoom button to select the view window you are interested in. You may zoom in by drawing a box (using the left mouse button, left-click on one of the corners and keep the button pressed, drag to a different part of the figure until the box has the shape you like). Zooming out works as follows. On Window, draw a box using the *right* mouse button (as for zooming in). When you are finished drawing the box, the current figure will be zoomed-out until it fits in the box you just drew. On a Mac, do the same but hold down the control key.

```
In []:  timtracelines(ml,xlist,ylist,zlist,step,twoD=1,tmax=1e30,Nmax=200,labfrac=2.0,
Hfrac=5.0,window=[-1e30,-1e30,1e30,1e30],color=None,width=0.5,xsec=0)
```
Plot tracelines on the current figure, where

- `xlist,ylist,zlist` are lists with starting values. All lists must have the same lengths. Even if you want to plot one trace line, you need to specify them as lists with one number each.
- `step` is the maximum step size in space. The step size may be reduced if *Tim*$^{\mathrm{ML}}$ expects that results will be inaccurate (see keywords `labfrac, Hfrac`).
- `tmax` is the maximum travel time.
- `Nmax` is the maximum number of steps.
- `labfrac`. Step size will be reduced to $\lambda_t extmax/$`labfrac` within a distance of $3\lambda_{\max}$ from an element.
- `Hfrac`. The step size will be reduced if the maximum vertical step is larger than $H/$`Hfrac`.

- `window`. Size of the window where the traceline is stopped.
- `color`. Color of the tracelines. If `None`, colors will change automatically between layers.
- `width`. Width of the tracelines.
- x̂sec. Set to True if traclines need to be added to a cross-sectional plot as well (provided the current figure already has a cross-sectional figure added to it).

`In []: capturezone( ml, w, N, z, step, tmax, xsec=False )`
Draw `N` tracelines from well `w` starting at elevations(s) `z` using step size `step` (this needs to be a negative value, as you want to go against the flow), stop at time `tmax`, and add the traclines to the cross-sectional window if `xsec=True`; `z` may be a number or a list of numbers.

`In []: timvertcontour( ml, x1, y1, x2, y2, nx, zmin, zmax, nz, levels = 10, color = None, width = 0.5, style = '-', newfig = 1, labels = 0, labelfmt = 'returnheads = 0, returncontours = 0, fill = 0, size=None)` Create a contour plot of a vertical cross-section, especially useful when making a 3D model with `Model3D`. All arguments are the same as for `timcontour` except that

- `x1,y1,x2,y2` are the horizontal coordinates of the end points of the cross-section.
- `zmin,zmax` are the vertical elevations of the bottom and top of the vertical cross-section.

Tracelines cannot be started interactively in a cross-section as flow is not confined to this cross-section except for in very special cases.

3.5.1 Export to Surfer

To create a grid file to look at contour plots of heads in Surfer type
`In []: surfgrid(ml,xmin,xmax,nx,ymin,ymax,ny,filename='/temp/dump',Naquifers=1)` where

- `modelParent` is the model name
- `xmin,xmax` are the minimum and maximum x values of the gridding window
- `nx` is the number of grid points minus 1 in the x direction (a grid of 1 gives 2 grid points, get it?)
- `ymin,ymax` are the minimum and maximum y values of the gridding window
- `ny` is the number of grid points minus 1 in the y direction
- `filename` is a filename between quotes and with forward slashes for the directory (Python is smart enough to figure out when you are on Windows, Mac, Linux, or Unix). Don't add an extension, this is added, with the layer number, for you.
- `Naquifers` has three different options. Either it is the string `'all'` in which case grids are produced for all aquifers. Or it can be a number, then grids are produced starting at the top and ending (but including) this aquifer number. And finally, it can be a list with numbers of the aquifers for which you want a grid; e.g. `[1,5,6]` will produce grids for aquifers 1, 5, and 6.

A bln file (compatible with Surfer) containing the layout of all analytic elements may be created by typing
`In []: surferlayout(ml,filename='/temp/dump.bln')`
A Surfer grid in a vertical cross-section (handy when you do a quasi 3D model) may be obtained with the command
`In []: surfvertgrid(ml,xmin,ymin,xmax,ymax,nh,zmin,zmax,nz,filename='/temp/dump')`
where `nh` is the number of horizontal grid points. This will create a horizontal axis starting at zero. Note that when you look at a vertical grid in Surfer you may have to reset the vertical scale, as Surfer automatically makes the horizontal and vertical scales equal (and thus you may get a very squooshed plot).

The command
`In []: surfertrace(ml,xstart,ystart,zstart,stepin,tmax=1e30,maxsteps=10,`
`tstart=0.0,filename='/temp/dump.bln')`
writes one trace line to a Surfer bln-file. This will be a 2D projection on the horizontal plane, as Surfer cannot do 3D plotting (at least as far as I know).

3.5.2 Export to Matlab

There is also a routine that spits out contouring data that can be read into MATLAB:
`In []: matgrid(ml,xmin,xmax,nx,ymin,ymax,ny,filename='/temp/dump.m',Naquifers=1)`
which will create a MATLAB m-file with variables xg, yg, h1, h2, ... depending on the number of layers. `Naquifers` may be specified the same way as for `surfgrid`. A m-file with the layout is obtained by typing
`In []: matlablayout(ml,'/temp/dump.m',color='k')`
To get the layout in MATLAB, open a figure and type 'hold', then type the name of the m-file containing the layout. A matlab grid in a vertical cross-section parallel to the x-axis may be created with the command
`In []: matlabvertgrid(ml,xmin,xstep,xmax,zmin,zstep,zmax,ycrosssection,filename='/temp/dump.m')`

Multiple trace lines can be generated and written to a MATLAB m-file with the command
`In []: matlabtracelines(ml,xrange,yrange,zrange,step,filename,twoD=1,tmax=1e30,Nmax=20)`
where `xrange,yrange,zrange` are lists or arrays with starting locations of the trace lines. Both arrays of the x, y, z locations of points along the trace line and a plotting statement are written to the m-file. The plotting statement depends on the setting of `twoD`. `twoD=1` plots 2D horizontal projections of pathlines, `twoD=2` plots 2D vertical projections of pathlines in the x, z plane, and `twoD=3` plots 3D pathlines.