TimML Documentation

Release 5.0.0

M. Bakker

Oct 30, 2017

CONTENTS

TimML is a computer program for the modeling of steady-state multi-layer flow with analytic elements TimML may be applied to an arbitrary number of layers and arbitrary sequence of aquifers and leaky layers. The Dupuit approximation is applied to aquifer layers, while flow in leaky layers is approximated as vertical. The head, flow, and leakage between aquifer layers may be computed analytically at any point in the aquifer system. The design of TimML 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. TimML is coded in Python. Behind the scenes, use is made of FORTRAN extensions to improve performance.

CHAPTER

ONE

INSTALLATION

TimML is written for Python 3.

pip install instructions will follwo soon.

CHAPTER

TWO

MAIN APPROXIMATIONS

CHAPTER

THREE

LIST OF AVAILABLE ELEMENTS

- Well
  - Discharge-specified well
  - Head-specified well
  - Multi-aquifer well. Well is screened in multiple layers and only total discharge is specified.
- Line-sink
  - Head-specified line-sink
  - String of head-specified line-sinks

3.1 Starting a Model

There are three ways to start a model:

1. *ModelMaq*, which is model consisting of a regular sequence of aquifer - leaky layer - aquifer - leaky layer, aquifer, etc. The top of the system can be either an aquifer or a leaky layer. The head is computed in all aquifer layers only.

```
ml = ModelMaq(kaq=[10, 30, 20], z=[0, -5, -10, -20, -25, -35], c=[2000, 5000])
```

2. *Model3D*, which is a model consisting of a stack of aquifer layers. The resistance between the aquifer layers is computed as the resistance from the middle of one layer to the middle of the next layer. Vertical anisotropy can be specified. The system may be bounded on top by a leaky layer.

```
Model3D(kaq=[10, 0.0025, 30, 0.001, 20], z=[0, -5, -10, -20, -25, -35], kzoverkh=0.1)
```

3. *Model*, which is a model consisting of an arbitrary sequence of leaky layers and aquifers. The resistance between all aquifers needs to be specified by the user. This is the most general option, but requires a bit more input.

```
Model(kaq=[10, 5, 30, 20], c=[2, 5, 2000], z=[0, -5, -10, -20, -25, -35],
      npor=[0.3, 0.3, 0.3, 0.3], ltype=['a', 'a', 'a', 'l', 'a'])
```

3.1.1 ModelMaq

class timml.model.**ModelMaq**(*kaq=1, z=[1, 0], c=[], npor=0.3, top='conf', hstar=None*)

ModelMaq Class to create a multi-aquifer model object

Parameters

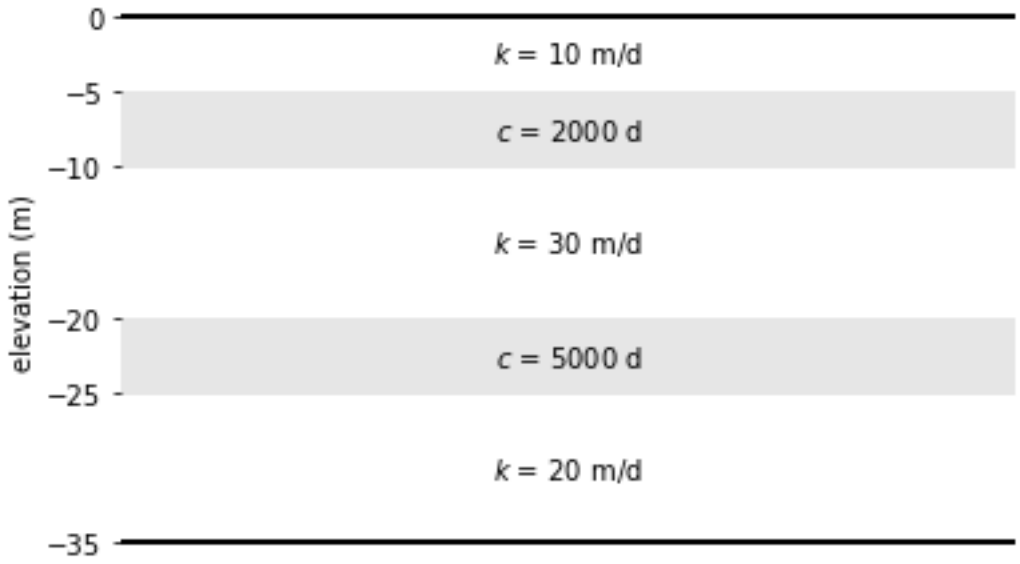

Fig. 3.1: A *ModelMaq* example with three aquifers and two leaky layers

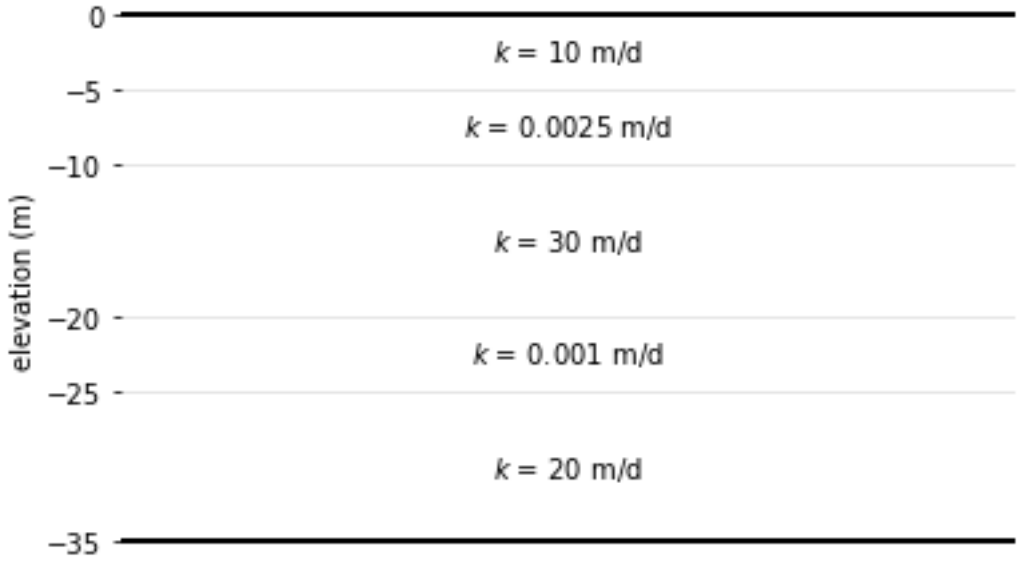

Fig. 3.2: A *Model3D* example consisting of five layers all treated as aquifers and a vertical anisotropy of 0.1

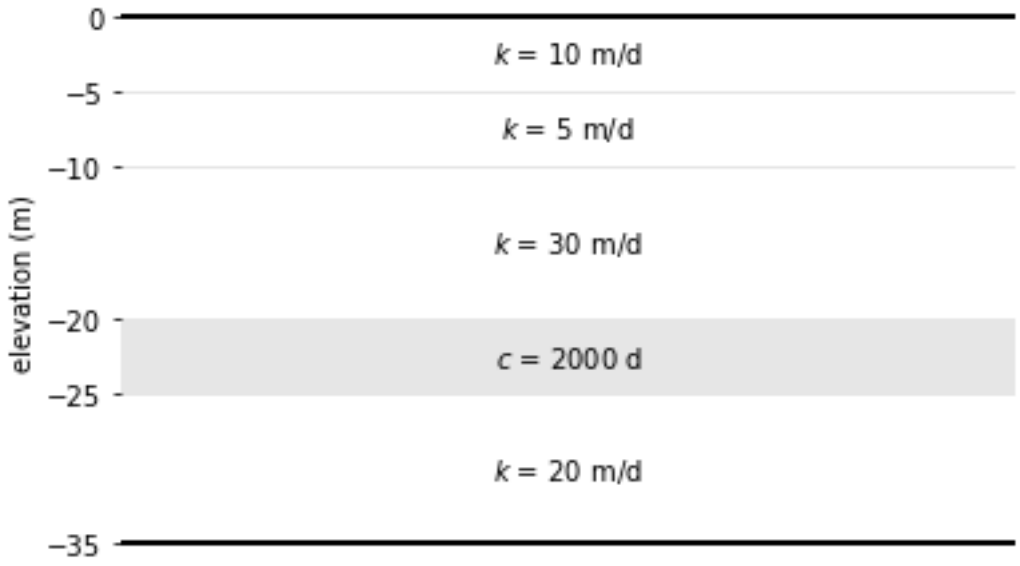

Fig. 3.3: A *Model* example with four aquifers and one leaky layer

- **kaq** (*float, array or list*) – hydraulic conductivity of each aquifer from the top down if float, hydraulic conductivity is the same in all aquifers
- **z** (*array or list*) – elevation tops and bottoms of the aquifers from the top down leaky layers may have zero thickness if top='conf': length is 2 * number of aquifers if top='semi': length is 2 * number of aquifers + 1 as top of leaky layer on top of systems needs to be specified
- **c** (*float, array or list*) – resistance of leaky layers from the top down if float, resistance is the same for all leaky layers if top='conf': length is number of aquifers - 1 if top='semi': length is number of aquifers
- **npor** (*float, array or list*) – porosity of all aquifers and leaky layers from the top down if float, porosity is the same for all layers if top='conf': length is 2 * number of aquifers - 1 if top='semi': length is 2 * number of aquifers
- **top** (*string, 'conf' or 'semi' (default is 'conf')*) – indicating whether the top is confined ('conf') or semi-confined ('semi')
- **hstar** (*float or None (default is None)*) – head value above semi-confining top, only read if top='semi'

Examples

```
>>> ml = ModelMaq(kaq=[10, 20], z=[20, 12, 10, 0], c=1000)
```

contour(*win*, *ngr=20*, *layers=0*, *levels=20*, *layout=True*, *labels=False*, *decimals=0*, *color=None*, *newfig=True*, *figsize=None*, *legend=True*)

Contour plot

Parameters

- **win** (*list or tuple*) – [x1, x2, y1, y2]

- **ngr** (*scalar, tuple or list*) – if scalar: number of grid points in x and y direction if tuple or list: nx, ny, number of grid points in x and y direction

disvec(*x*, *y*, *aq=None*)
Discharge vector at *x*, *y*

Returns **qxqy** – first row is Qx in each aquifer layer, second row is Qy

Return type array size (2, naq)

head(*x*, *y*, *layers=None*, *aq=None*)
Head at *x*, *y*

Returns **h** – head in all *layers* (if not *None*), or all layers of aquifer (otherwise)

Return type array length *naq* or *len(layers)*

headalongline(*x*, *y*, *layers=None*)
Returns head[Nlayers,len(x)] Assumes same number of layers for each x and y layers may be None or list of layers for which head is computed

headgrid(*xg*, *yg*, *layers=None*, *printrow=False*)
Grid of heads

Parameters

- **xg** (*array*) – x values of grid
- **yg** (*array*) – y values of grid
- **layers** (*integer, list or array, optional*) – layers for which grid is returned
- **printrow** (*boolean, optional*) – prints dot to screen for each row of grid if set to *True*

Returns **h**

Return type array size *nlayers, ny, ny*

See also:

headgrid2()

headgrid2(*x1*, *x2*, *nx*, *y1*, *y2*, *ny*, *layers=None*, *printrow=False*)
Grid of heads

Parameters

- **x2, nx** (*x1,*) – x values of grid
- **y2, ny** (*y1,*) – y values of grid
- **layers** (*integer, list or array, optional*) – layers for which grid is returned
- **printrow** (*boolean, optional*) – prints dot to screen for each row of grid if set to *True*

Returns **h**

Return type array size *nlayers, ny, ny*

plot(*win=None*, *newfig=True*, *figsize=None*, *orientation='hor'*, *topfigfrac=0.8*)
Plot layout

Parameters **win** (*list or tuple*) – [x1, x2, y1, y2]

remove_element(*e*)
Remove element *e* from model

solve(*printmat=0*, *sendback=0*, *silent=False*)
Compute solution

tracelines(*xstart, ystart, zstart, hstepmax, vstepfrac=0.2, tmax=1000000000000.0, nstepmax=100, silent='.', color=None, orientation='hor', win=[-1e+30, 1e+30, -1e+30, 1e+30], newfig=False, figsize=None*)
Draw trace lines

vcontour(*win*, *n*, *levels*, *labels=False*, *decimals=0*, *color=None*, *vinterp=True*, *nudge=1e-06*, *newfig=True*, *figsize=None*, *layout=True*)
Vertical contour

3.1.2 Model3D

class timml.model.**Model3D**(*kaq=1, z=[1, 0], kzoverkh=1, npor=0.3, top='conf', topres=0, topthick=0, hstar=0*)
Model3D Class to create a multi-layer model object consisting of many aquifer layers. The resistance between the layers is computed from the vertical hydraulic conductivity of the layers.

Parameters

- **kaq** (*float, array or list*) – hydraulic conductivity of each layer from the top down if float, hydraulic conductivity is the same in all aquifers
- **z** (*array or list*) – elevation of top of system followed by bottoms of all layers from the top down bottom of layer is automatically equal to top of layer below it if top='conf': length is number of layers + 1 if top='semi': length is number of layers + 2 as top of leaky layer on top of systems needs to be specified
- **kzoverkh** (*float*) – vertical anisotropy ratio vertical k divided by horizontal k if float, value is the same for all layers length is number of layers
- **npor** (*float, array or list*) – porosity of all aquifer layers from the top down if float, porosity is the same for all layers if top='conf': length is number of layers if top='semi': length is number of layers + 1
- **top** (*string, 'conf' or 'semi' (default is 'conf')*) – indicating whether the top is confined ('conf') or semi-confined ('semi')
- **topres** (*float*) – resistance of top semi-confining layer, only read if top='semi'
- **topthick** (*float*) – thickness of top semi-confining layer, only read if top='semi'
- **hstar** (*float or None (default is None)*) – head value above semi-confining top, only read if top='semi'

Examples

```
>>> ml = Model3D(kaq=10, z=np.arange(20, -1, -2), kzoverkh=0.1)
```

contour(*win*, *ngr=20*, *layers=0*, *levels=20*, *layout=True*, *labels=False*, *decimals=0*, *color=None*, *newfig=True*, *figsize=None*, *legend=True*)
Contour plot

Parameters

- **win** (*list or tuple*) – [x1, x2, y1, y2]

- **ngr** (*scalar, tuple or list*) – if scalar: number of grid points in x and y direction if tuple or list: nx, ny, number of grid points in x and y direction

disvec(*x*, *y*, *aq=None*)
Discharge vector at *x*, *y*

Returns **qxqy** – first row is Qx in each aquifer layer, second row is Qy

Return type array size (2, naq)

head(*x*, *y*, *layers=None*, *aq=None*)
Head at *x*, *y*

Returns **h** – head in all *layers* (if not *None*), or all layers of aquifer (otherwise)

Return type array length *naq* or *len(layers)*

headalongline(*x*, *y*, *layers=None*)
Returns head[Nlayers,len(x)] Assumes same number of layers for each x and y layers may be None or list of layers for which head is computed

headgrid(*xg*, *yg*, *layers=None*, *printrow=False*)
Grid of heads

Parameters

- **xg** (*array*) – x values of grid
- **yg** (*array*) – y values of grid
- **layers** (*integer, list or array, optional*) – layers for which grid is returned
- **printrow** (*boolean, optional*) – prints dot to screen for each row of grid if set to *True*

Returns **h**

Return type array size *nlayers, ny, ny*

See also:

headgrid2()

headgrid2(*x1*, *x2*, *nx*, *y1*, *y2*, *ny*, *layers=None*, *printrow=False*)
Grid of heads

Parameters

- **x2, nx** (*x1,*) – x values of grid
- **y2, ny** (*y1,*) – y values of grid
- **layers** (*integer, list or array, optional*) – layers for which grid is returned
- **printrow** (*boolean, optional*) – prints dot to screen for each row of grid if set to *True*

Returns **h**

Return type array size *nlayers, ny, ny*

plot(*win=None*, *newfig=True*, *figsize=None*, *orientation='hor'*, *topfigfrac=0.8*)
Plot layout

Parameters **win** (*list or tuple*) – [x1, x2, y1, y2]

remove_element(*e*)
Remove element *e* from model

solve(*printmat=0*, *sendback=0*, *silent=False*)
Compute solution

tracelines(*xstart, ystart, zstart, hstepmax, vstepfrac=0.2, tmax=1000000000000.0, nstepmax=100, silent='.', color=None, orientation='hor', win=[-1e+30, 1e+30, -1e+30, 1e+30], newfig=False, figsize=None*)
Draw trace lines

vcontour(*win*, *n*, *levels*, *labels=False*, *decimals=0*, *color=None*, *vinterp=True*, *nudge=1e-06*, *newfig=True*, *figsize=None*, *layout=True*)
Vertical contour

3.1.3 Model

class timml.model.**Model**(*kaq*, *c*, *z*, *npor*, *ltype*)
Model Class to create a model object consisting of an arbitrary sequence of aquifer layers and leaky layers. Use ModelMaq for regular sequence of aquifer and leaky layers. Use Model3D for multi-layer model of single aquifer

Parameters

- **kaq** (*array*) – hydraulic conductivity of each aquifer from the top down
- **z** (*array*) – elevation tops and bottoms of all layers layers may have zero thickness
- **c** (*array*) – resistance between two consecutive aquifers if ltype[0]='a': length is number of aquifers - 1 if ltype[0]='l': length is number of aquifers
- **npor** (*array*) – porosity of all layers from the top down
- **ltype** (*array of characters*) – array indicating for each layer whether it is 'a' aquifer layer 'l' leaky layer

remove_element(*e*)
Remove element *e* from model

disvec(*x*, *y*, *aq=None*)
Discharge vector at *x*, *y*

Returns **qxqy** – first row is Qx in each aquifer layer, second row is Qy

Return type array size (2, naq)

head(*x*, *y*, *layers=None*, *aq=None*)
Head at *x*, *y*

Returns **h** – head in all *layers* (if not *None*), or all layers of aquifer (otherwise)

Return type array length *naq* or *len(layers)*

headgrid(*xg*, *yg*, *layers=None*, *printrow=False*)
Grid of heads

Parameters

- **xg** (*array*) – x values of grid
- **yg** (*array*) – y values of grid
- **layers** (*integer, list or array, optional*) – layers for which grid is returned

- **printrow** (*boolean, optional*) – prints dot to screen for each row of grid if set to *True*

Returns h

Return type array size *nlayers, ny, ny*

See also:

headgrid2()

headgrid2 (*x1*, *x2*, *nx*, *y1*, *y2*, *ny*, *layers=None*, *printrow=False*)
Grid of heads

Parameters

- **x2, nx** (*x1,*) – x values of grid
- **y2, ny** (*y1,*) – y values of grid
- **layers** (*integer, list or array, optional*) – layers for which grid is returned
- **printrow** (*boolean, optional*) – prints dot to screen for each row of grid if set to *True*

Returns h

Return type array size *nlayers, ny, ny*

headalongline (*x*, *y*, *layers=None*)
Returns head[Nlayers,len(x)] Assumes same number of layers for each x and y layers may be None or list of layers for which head is computed

solve (*printmat=0*, *sendback=0*, *silent=False*)
Compute solution

contour (*win*, *ngr=20*, *layers=0*, *levels=20*, *layout=True*, *labels=False*, *decimals=0*, *color=None*, *newfig=True*, *figsize=None*, *legend=True*)
Contour plot

Parameters

- **win** (*list or tuple*) – [x1, x2, y1, y2]
- **ngr** (*scalar, tuple or list*) – if scalar: number of grid points in x and y direction if tuple or list: nx, ny, number of grid points in x and y direction

plot (*win=None*, *newfig=True*, *figsize=None*, *orientation='hor'*, *topfigfrac=0.8*)
Plot layout

Parameters **win** (*list or tuple*) – [x1, x2, y1, y2]

tracelines (*xstart, ystart, zstart, hstepmax, vstepfrac=0.2, tmax=1000000000000.0, nstepmax=100, silent='.', color=None, orientation='hor', win=[-1e+30, 1e+30, -1e+30, 1e+30], newfig=False, figsize=None*)
Draw trace lines

vcontour (*win*, *n*, *levels*, *labels=False*, *decimals=0*, *color=None*, *vinterp=True*, *nudge=1e-06*, *newfig=True*, *figsize=None*, *layout=True*)
Vertical contour

3.2 Inhomogeneities

Polygonal inhomogeneities can be added in three ways:

1. *PolygonInhomMaq*, which is an inhomogeneity consisting of a regular sequence of aquifer - leaky layer - aquifer - leaky layer, aquifer, etc. The top of the system can be either an aquifer or a leaky layer.
2. Model3D to be added

3.2.1 ModelMaq

class timml.inhomogeneity.**PolygonInhomMaq**(*model, xy, kaq=1, z=[1, 0], c=[], npor=0.3, top='conf', hstar=None, order=3, ndeg=3*)

Create a polygonal inhomogeneity

Parameters

- **model** (*Model object*) – model to which the element is added
- **xy** (*array or list*) – list or array of (x,y) pairs of coordinates of corners of the inhomogeneity polygonal boundary is automatically closed (so first point is not repeated)
- **kaq** (*float, array or list*) – hydraulic conductivity of each aquifer from the top down if float, hydraulic conductivity is the same in all aquifers
- **z** (*array or list*) – elevation tops and bottoms of the aquifers from the top down leaky layers may have zero thickness if top='conf': length is 2 * number of aquifers if top='semi': length is 2 * number of aquifers + 1 as top of leaky layer on top of systems needs to be specified
- **c** (*float, array or list*) – resistance of leaky layers from the top down if float, resistance is the same for all leaky layers if top='conf': length is number of aquifers - 1 if top='semi': length is number of aquifers
- **npor** (*float, array or list*) – porosity of all aquifers and leaky layers from the top down if float, porosity is the same for all layers if top='conf': length is 2 * number of aquifers - 1 if top='semi': length is 2 * number of aquifers
- **top** (*string, 'conf' or 'semi' (default is 'conf')*) – indicating whether the top is confined ('conf') or semi-confined ('semi')
- **hstar** (*float or None (default is None)*) – head value above semi-confining top, only read if top='semi'
- **order** (*int*) – polynomial order of flux along each segment
- **ndeg** (*int*) – number of points used between two segments to numerically integrate normal discharge

3.3 Elements

3.3.1 Wells

There are two types of wells: wells for which the total discharge is specified and wells for which the head inside the well is specified. Both types of wells may have an entry resistance (skin effect) defined by the resistance c (dimension: time). The discharge Q_i in layer i is a function of the head h_i in layer i just outside the well and the head h_w inside the well:

$$Q_i = 2\pi r_w (h_i - h_w)/c$$

1. *Well* is a well for which the total discharge is specified. The total discharge is distributed across the layers in which the well is screened such that the head inside the well is the same in each screened layer.

2. *HeadWell* is a well for which the head inside the well is specified. The discharge in each layer is computed such that the head in all screened layers is equal to the specified head.

Well with specified discharge

class timml.well.**Well**(*model*, *xw=0*, *yw=0*, *Qw=100.0*, *rw=0.1*, *res=0.0*, *layers=0*, *label=None*)

Well Class to create a well with a specified discharge. The well may be screened in multiple layers. The resistance of the screen may be specified. The head is computed such that the discharge Q_i in layer i is computed as

$$Q_i = 2\pi r_w (h_i - h_w)/c$$

where c is the resistance of the well screen and h_w is the head inside the well. The total discharge is distributed over the screens such that h_w is the same in each screened layer.

Parameters

- **model** (*Model object*) – model to which the element is added
- **xw** (*float*) – x-coordinate of the well
- **yw** (*float*) – y-coordinate of the well
- **Qw** (*float*) – total discharge of the well
- **rw** (*float*) – radius of the well
- **res** (*float*) – resistance of the well screen
- **layers** (*int, array or list*) – layer (int) or layers (list or array) where well is screened
- **label** (*string or None (default: None)*) – label of the well

Examples

```
>>> ml = Model3D(kaq=10, z=np.arange(20, -1, -2), kzoverkh=0.1)
>>> Well(ml, 100, 200, 1000, layers=[0, 1, 2, 3])
```

capzone(*nt=10*, *zstart=None*, *hstepmax=10*, *vstepfrac=0.2*, *tmax=None*, *nstepmax=100*, *silent='.'*)

Compute a capture zone

Parameters

- **nt** (*int*) – number of path lines
- **zstart** (*scalar*) – starting elevation of the path lines
- **hstepmax** (*scalar*) – maximum step in horizontal space
- **vstepfrac** (*float*) – maximum fraction of aquifer layer thickness during one step
- **tmax** (*scalar*) – maximum time

- **nstepmax** (*scalar (int)*) – maximum number of steps
- **silent** (*boolean or string*) – True (no messages), False (all messages), or '.' (print dot for each path line)

Returns xyzt

Return type list of arrays of x, y, z, and t values

discharge()
The discharge in each layer

Returns Discharge in each screen with zeros for layers that are not screened

Return type array (length number of layers)

headinside()
The head inside the well

Returns Head inside the well for each screen

Return type array (length number of screens)

plotcapzone(*nt=10, zstart=None, hstepmax=20, vstepfrac=0.2, tmax=365, nstepmax=100, silent='.', color=None, orientation='hor', win=[-1e+30, 1e+30, -1e+30, 1e+30], newfig=False, figsize=None*)
Plot a capture zone

Parameters

- **nt** (*int*) – number of path lines
- **zstart** (*scalar*) – starting elevation of the path lines
- **hstepmax** (*scalar*) – maximum step in horizontal space
- **vstepfrac** (*float*) – maximum fraction of aquifer layer thickness during one step
- **tmax** (*scalar*) – maximum time
- **nstepmax** (*scalar (int)*) – maximum number of steps
- **silent** (*boolean or string*) – True (no messages), False (all messages), or '.' (print dot for each path line)
- **color** (*color*) –
- **orientation** (*string*) – 'hor' for horizontal, 'ver' for vertical, or 'both' for both
- **win** (*array_like (length 4)*) – [xmin, xmax, ymin, ymax]
- **newfig** (*boolean (default False)*) – boolean indicating if new figure should be created
- **figsize** (*tuple of integers, optional, default: None*) – width, height in inches.

Well with specified head

class timml.well.**HeadWell**(*model*, *xw=0*, *yw=0*, *hw=10*, *rw=0.1*, *res=0*, *layers=0*, *label=None*)
HeadWell Class to create a well with a specified head inside the well. The well may be screened in multiple layers. The resistance of the screen may be specified. The head is computed such that the discharge Q_i in layer i is computed as

$$Q_i = 2\pi r_w (h_i - h_w)/c$$

where c is the resistance of the well screen and h_w is the head inside the well.

Parameters

- **model** (*Model object*) – model to which the element is added
- **xw** (*float*) – x-coordinate of the well
- **yw** (*float*) – y-coordinate of the well
- **hw** (*float*) – head inside the well
- **rw** (*float*) – radius of the well
- **res** (*float*) – resistance of the well screen
- **layers** (*int, array or list*) – layer (int) or layers (list or array) where well is screened
- **label** (*string (default: None)*) – label of the well

capzone(*nt=10*, *zstart=None*, *hstepmax=10*, *vstepfrac=0.2*, *tmax=None*, *nstepmax=100*, *silent='.'*)
Compute a capture zone

Parameters

- **nt** (*int*) – number of path lines
- **zstart** (*scalar*) – starting elevation of the path lines
- **hstepmax** (*scalar*) – maximum step in horizontal space
- **vstepfrac** (*float*) – maximum fraction of aquifer layer thickness during one step
- **tmax** (*scalar*) – maximum time
- **nstepmax** (*scalar (int)*) – maximum number of steps
- **silent** (*boolean or string*) – True (no messages), False (all messages), or '.' (print dot for each path line)

Returns xyzt

Return type list of arrays of x, y, z, and t values

discharge()
The discharge in each layer

Returns Discharge in each screen with zeros for layers that are not screened

Return type array (length number of layers)

headinside()
The head inside the well

Returns Head inside the well for each screen

Return type array (length number of screens)

plotcapzone(*nt=10, zstart=None, hstepmax=20, vstepfrac=0.2, tmax=365, nstepmax=100, silent='.', color=None, orientation='hor', win=[-1e+30, 1e+30, -1e+30, 1e+30], newfig=False, figsize=None*)
Plot a capture zone

Parameters

- **nt** (*int*) – number of path lines
- **zstart** (*scalar*) – starting elevation of the path lines

- **hstepmax** (*scalar*) – maximum step in horizontal space
- **vstepfrac** (*float*) – maximum fraction of aquifer layer thickness during one step
- **tmax** (*scalar*) – maximum time
- **nstepmax** (*scalar (int)*) – maximum number of steps
- **silent** (*boolean or string*) – True (no messages), False (all messages), or '.' (print dot for each path line)
- **color** (*color*) –
- **orientation** (*string*) – 'hor' for horizontal, 'ver' for vertical, or 'both' for both
- **win** (*array_like (length 4)*) – [xmin, xmax, ymin, ymax]
- **newfig** (*boolean (default False)*) – boolean indicating if new figure should be created
- **figsize** (*tuple of integers, optional, default: None*) – width, height in inches.

3.3.2 Line-sinks

Line-sinks are lines along which water is taken out of or put into the aquifer. The inflow/outflow along the line-sink varies as a polynomial (the order of the polynomial may be specified). There are two types of line-sinks: line-sinks for which the head is specified along the line-sink, and line-sinks for which the total discharge is specified and the head along the line-sink is unknown but uniform. Both types of line-sinks may have an entry resistance defined by the resistance c (dimension: time). The inflow σ_i into the line-sink in layer i is a function of the head h_i just outside the line-sink in layer i and the head h_{ls} inside the line-sink:

$$\sigma_i = w(h_i - h_{ls})/c$$

This equation is applied along each control point of the line-sink. w is the distance over which water infiltrates into the line-sink. The distanc may be the width of the stream, for example, in case of a partially penetrating stream. In case the stream penetrates the aquifer layer fully, the distance may equal the thickness of the aquifer layer (if water enters primarily from one side), or twice the aquifer thickness (if water enters from both sides).

The two types of line-sinks may be entered one by one or as a string. For a string of head-specified line-sinks, the head needs only be specified at the beginning and end of the string and the head is interpolated between these values. Alternatively, the head may be specified at the beginning and end of some or all line-sinks in the string For a string of line-sink ditch elements, the head is uniform along the entire string while the total discharge of all the line-sinks in the string is equal to the specified value.

1. *HeadLineSink* is a line-sink for which the head is specified along the line-sink. Example Notebook.
2. *HeadLineSinkString* is a string of head-specified line-sinks
3. *LineSinkDitch* is a line-sink for which the head is unknown and uniform and the total discharge is specified
4. *LineSinkDitchString* is a string of line-sink ditch elements

Line-sink with specified head

class timml.linesink.**HeadLineSink**(*model*, *x1=-1*, *y1=0*, *x2=1*, *y2=0*, *hls=1.0*, *res=0*, *wh=1*, *order=0*, *layers=0*, *label=None*, *name='HeadLineSink'*, *addtomodel=True*)

HeadLineSink Class to create a head-specified line-sink which may optionally have a width and resistance

Parameters

- **model** (*Model object*) – Model to which the element is added
- **x1** (*scalar*) – x-coordinate of fist point of line-sink
- **y1** (*scalar*) – y-coordinate of fist point of line-sink
- **x2** (*scalar*) – x-coordinate of second point of line-sink
- **y2** (*scalar*) – y-coordinate of second point of line-sink
- **hls** (*scalar, array or list*) – head along line-sink if scalar: head is the same everywhere along line-sink if list or array of length 2: head at beginning and end of line-sink if list or array with length order + 1: heads at control points
- **res** (*scalar (default is 0)*) – resistance of line-sink
- **wh** (*scalar or str*) – distance over which water enters line-sink
- **order** (*int (default is 0)*) – polynomial order or inflow along line-sink
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **label** (*str or None*) – label of element

See also:

HeadLineSinkString

String of head-specified line-sinks

class timml.linesink.**HeadLineSinkString**(*model, xy=[(-1, 0), (1, 0)], hls=0, res=0, wh=1, order=0, layers=0, label=None, name='HeadLineSinkString'*)

Class to create a string of head-specified line-sinks which may optionally have a width and resistance

Parameters

- **model** (*Model object*) – Model to which the element is added
- **xy** (*array or list*) – list or array of (x,y) pairs of coordinates of end-points of line-sinks in string
- **hls** (*scalar, array or list*) – head along string if scalar: head is the same everywhere along the string if list or array of length 2: head at beginning and end of string if list or array with same length as xy: heads at nodes, which may contain nans, except for first and last point
- **res** (*scalar (default is 0)*) – resistance of line-sink
- **wh** (*scalar or str*) – distance over which water enters line-sink
- **order** (*int (default is 0)*) – order of all line-sinks in string
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **label** (*str or None*) –

See also:

HeadLineSink

discharge()
Discharge of the element in each layer

Line-sink ditch

Specified total discharge with unknown but uniform head

class timml.linesink.**LineSinkDitch**(*model*, *x1=-1*, *y1=0*, *x2=1*, *y2=0*, *Qls=1*, *res=0*, *wh=1*, *order=0*, *layers=0*, *label=None*, *addtomodel=True*)
Class to create a line-sink for which the total discharge is specified, and for which the head along the line-sink is uniform but unknown.

Parameters

- **model** (*Model object*) – Model to which the element is added
- **x1** (*scalar*) – x-coordinate of fist point of line-sink
- **y1** (*scalar*) – y-coordinate of fist point of line-sink
- **x2** (*scalar*) – x-coordinate of second point of line-sink
- **y2** (*scalar*) – y-coordinate of second point of line-sink
- **Qls** (*scalar*) – total discharge of the line-sink
- **res** (*scalar (default is 0)*) – resistance of line-sink
- **wh** (*scalar or str*) – distance over which water enters line-sink
- **order** (*int (default is 0)*) – polynomial order or inflow along line-sink
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **label** (*str or None*) – label of element

See also:

LineSinkDitchString

Line-sink ditch string

Specified total discharge with unknown but uniform head

class timml.linesink.**LineSinkDitchString**(*model, xy=[(-1, 0), (1, 0)], Qls=1, res=0, wh=1, order=0, layers=0, label=None*)
Class to create a string of LineSinkDitch elements for which the total discharge of the string is specified, and for which the head along the entire string is uniform but unknown.

Parameters

- **model** (*Model object*) – Model to which the element is added
- **xy** (*array or list*) – list or array of (x,y) pairs of coordinates of end-points of line-sinks in string
- **Qls** (*scalar*) – total discharge of the string
- **res** (*scalar (default is 0)*) – resistance of line-sinks in string
- **wh** (*scalar or str*) – distance over which water enters the string

- **order** (*int (default is 0)*) – polynomial order or inflow along each line-sink in string
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **label** (*str or None*) – label of element

See also:

LineSinkDitch

3.3.3 Line-doublets

Line-doublets are used to simulate leaky and impermeable walls. The flux normal to an impermeable wall is zero. The flux through a leaky wall (the normal component q_n) is defined as

$$q_n = (h^- - h^+)/c$$

where h^- and h^- are the heads on the minus and plus sides of the wall, and c is the resistance against flow through the walll. An impermeable wall is equivalent to a leaky wall with a resistance that is equal to infinity.

1. *ImpLineDoublet* is used to simulate one straight impermeable wall
2. *ImpLineDoubletString* is an impermeable wall represented by a poly line of straight segments
3. *LeakyLineDoublet* is used to simulate one straight leaky wall
4. *LeakyLineDoubletString* is a leaky wall represented by a poly line of straight segments

Impermeable wall

class timml.linedoublet.**ImpLineDoublet**(*model*, *x1=-1*, *y1=0*, *x2=1*, *y2=0*, *order=0*, *layers=0*, *label=None*, *addtomodel=True*)

Create a segment of an impermeable wall, which is simulated with a line-doublet

Parameters

- **model** (*Model object*) – Model to which the element is added
- **x1** (*scalar*) – x-coordinate of fist point of line-doublet
- **y1** (*scalar*) – y-coordinate of fist point of line-doublet
- **x2** (*scalar*) – x-coordinate of second point of line-doublet
- **y2** (*scalar*) – y-coordinate of second point of line-doublet
- **order** (*int (default is 0)*) – polynomial order of potential jump along line-doublet (head jump if transmissivity is equal on each side of wall)
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **label** (*str or None*) – label of element

See also:

ImpLineDoubletString

String of impermeable wall segments

class timml.linedoublet.**ImpLineDoubletString**(*model, xy=[(-1, 0), (1, 0)], layers=0, order=0, label=None*)

Create a string of impermeable wall segements consisting of line-doublets

Parameters

- **model** (*Model object*) – Model to which the element is added
- **xy** (*array or list*) – list or array of (x,y) pairs of coordinates of end-points of the segements in the string
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **order** (*int (default is 0)*) – polynomial order of potential jump along line-doublet (head jump if transmissivity is equal on each side of wall)
- **label** (*str or None*) – label of element

See also:

ImpLineDoublet

Leaky wall

class timml.linedoublet.**LeakyLineDoublet**(*model, x1=-1, y1=0, x2=1, y2=0, res=0, order=0, layers=0, label=None, addtomodel=True*)

Create a segment of a leaky wall, which is simulated with a line-doublet. The specific discharge through the wall is equal to the head difference across the wall divided by the resistance of the wall.

Parameters

- **model** (*Model object*) – Model to which the element is added
- **x1** (*scalar*) – x-coordinate of fist point of line-doublet
- **y1** (*scalar*) – y-coordinate of fist point of line-doublet
- **x2** (*scalar*) – x-coordinate of second point of line-doublet
- **y2** (*scalar*) – y-coordinate of second point of line-doublet
- **res** (*scalar*) – resistance of leaky wall
- **order** (*int (default is 0)*) – polynomial order of potential jump along line-doublet (head jump if transmissivity is equal on each side of wall)
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **label** (*str or None*) – label of element

See also:

LeakyLineDoubletString

String of leaky wall segments

class timml.linedoublet.**LeakyLineDoubletString**(*model, xy=[(-1, 0), (1, 0)], res=inf, layers=0, order=0, label=None*)

Class to create a string of leaky wall segements consisting of line-doublets

Parameters

- **model** (*Model object*) – Model to which the element is added
- **xy** (*array or list*) – list or array of (x,y) pairs of coordinates of end-points of the segements in the string
- **res** (*scalar*) – resistance of leaky wall
- **layers** (*scalar, list or array*) – layer(s) in which element is placed if scalar: element is placed in this layer if list or array: element is placed in all these layers
- **order** (*int (default is 0)*) – polynomial order of potential jump along line-doublet (head jump if transmissivity is equal on each side of wall)
- **label** (*str or None*) – label of element

See also:

ImpLineDoublet

3.3.4 Constant

Every model needs at least one head-specified condition. In addition, the head may be specified at one point in one layer, provided this is in an area where the aquifer is confined. This point is historically called the *reference point*. The *reference* point is useful to control the behavior in the far field, especially in models where the far field is not modeled explicitly.

class timml.constant.**Constant**(*model*, *xr=0*, *yr=0*, *hr=0.0*, *layer=0*, *label=None*)

Specify the head at one point in the model in one layer. The head may only be specified in an area of the model where the aquifer system is confined.

Parameters

- **model** (*Model object*) – model to which the element is added
- **xr** (*float*) – x-coordinate of the point where the head is specified
- **yr** (*float*) – y-coordinate of the point where the head is specified
- **hr** (*float*) – specified head
- **rw** (*float*) – radius of the well
- **layer** (*int*) – layer where the head is specified
- **label** (*string or None (default: None)*) – label of the element

3.3.5 Uniform Flow

class timml.uflow.**Uflow**(*model*, *slope*, *angle*, *label=None*)

Add uniform flow to the model. Uniform flow may only be added to a model of which the background aquifer system is confined.

Parameters

- **model** (*Model object*) – model to which the uniform flow is added
- **slope** (*float*) – slope of the head (head drop divided by the distance) in the direction of flow
- **angle** (*float*) – direction of flow in degerees (0 degrees is straight East, counter clock-wise is positive)

- **label** (*string or None (default: None)*) – label of the element

3.3.6 Area-sinks

There is a circular area-sink. In the future there will be a strip area-sink and a polygonal area-sink.

Circular Area-Sink

class timml.circareasink.**CircAreaSink**(*model*, *xc=0*, *yc=0*, *R=1*, *N=0.001*, *layer=0*, *name='CircAreasink'*, *label=None*)

3.4 Utilities

Utilities include contouring in both the horizontal and vertical plane, and tracing path lines in both horizontal and vertical cross-section. All these utilities are functions of the Model class. .. |project| replace:: TimML

INDEX

W