SingleLayer a Python class implementing a semi-analytic solution to the pressure disturbance due to injection into an aquifer with or without leaky layers Version 0.1

Karl W. Bandilla

December 11, 2009

1 Introduction

SingleLayer is a class programmed in the scripting language Python. The purpose of the class is to compute the pressure disturbance caused by injecting a fluid into an aquifer. The aquifer is either bound at the top and bottom by leaky layers or impermeable layers. The semi-analytic solutions implemented in this class are based on [1], [2], [3] and [4]. The units of the input variables are assumed to be consistent (e.g., all time in seconds, and all lengths in meters). The validity of input variables is not checked within this class, so bad input values (e.g., strings instead of numbers, negative values, ...) are going to lead to crashes.

2 License

SingleLayer is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free

3 Installation

In order to use *SingleLayer.py* Python 2.5 needs to be installed on the computer. Later versions of Python may be used, if they are supported by the *NumPy* and *SciPy* modules. Python and all needed third-party modules are freely available online. It is suggested that Python is installed from a package such as *PythonXY* which includes all third-party modules needed for *Zhouetal.py*. Alternatively, Python can be installed from www.python.org. In this case the packages *NumPy* and *SciPy* need to be installed separately.

4 Class structure

4.1 Global variables

The following variables store the main domain properties, and are available throughout the class (variable names in square brackets denote variable lists instead of single values):

- *[K]*: list of hydraulic conductivities [L/T] (float). The list has 1, 2 or 3 list entries: 1 if no leaky layers are present; 2 if only 1 leaky layer is present; 3 if 2 leaky layers are used. The first list entry is the conductivity of the aquifer; the other one(s) are for the leaky layer(s).

- *[H]*: list of layer thicknesses [L] (float). The list has 1, 2 or 3 list entries: 1 if no leaky layers are present; 2 if only 1 leaky layer is present; 3 if 2 leaky layers are used. The first list entry is the thickness of the aquifer; the other one(s) are for the leaky layer(s).

- *[S]*: list of storativities (volume of water that a unit decline of head releases from storage in a vertical prism of the aquifer of unit cross section) [-] (float). The list has 1, 2 or 3 list entries: 1 if no leaky layers are present; 2 if only 1 leaky layer is present; 3 if 2 leaky layers are used. The first list entry is the storativity of the aquifer; the other one(s) are for the leaky layer(s).

- Q: volumetric injection rate [L^3/T] (float). Positive into the aquifer.

- rw: well radius [L] (float). For an infinitesimal well radius rw is set to -1.

- rb: distance from injection well to lateral boundary [L] (float). For infinite aquifers rb has the value -1.

- $LLBC$: if leaky layers present, boundary condition at the top of the upper leaky layer and the bottom of the lower leaky layer (string). 'constant' for constant head boundary conditions and 'noflow' for no-flow boundary conditions.

- *LLstorage*: should storage in the leaky layer(s) be considered [-] (boolean). This variable is used to turn off leaky layer storage for comparisons to the Hantush-Jacob solution [2].

- *InjectionEnd*: time at which injection ceases [T] (float). It is assumed that injection started at t=0.0.

The following variables are dimensionless variables derived from the variables above. These variables are available throughout the class, but are intended for internal use only.

- $dimenTime = \frac{K[0]}{S[0]H[0]}$. Factor to convert from regular time to dimensionless time.

- $HeadFactor = \frac{Q}{4\pi K[0]H[0]}$. Factor to convert from dimensionless head to regular head.

- $rDw = \frac{rw}{H[0]}$. Dimensionless well radius (if applicable).

- $rDb = \frac{rb}{H[0]}$. Dimensionless boundary radius (if applicable).

- *sigmat* $= \frac{S[1]}{S[0]}$. Storage factor with respect to first leaky layer (if applicable).

- *sigmab* $= \frac{S[2]}{S[0]}$. Storage factor with respect to second leaky layer (if applicable).

- *lambdat* $= \sqrt{\frac{K[1]H[0]}{K[0]H[1]}}$. Leakage factor with respect to first leaky layer (if applicable).

- *lambdab* $= \sqrt{\frac{K[2]H[0]}{K[0]H[2]}}$. Leakage factor with respect to the second leaky layer (if applicable).

- *mt* $= \frac{\sqrt{sigmat}}{lambdat}$. Ratio of the square root of the storage factor to the leakage factor with respect to the first leaky layer (if applicable).

- *mb* $= \frac{\sqrt{sigmab}}{lambdab}$. Ratio of the square root of the storage factor to the leakage factor with respect to the second leaky layer (if applicable).

There are also several global variables that are required for the numerical Laplace inversion, which are also intended for internal use only:

- N: number of steps used in inversion (set to 400).

- *work*: array of complex numbers of length N+1.

- d: additional array of complex number of length N+1.

- *meth*: method to be used for inversion (set to 3).

- *merror*: accuracy measure (set to 10^{-10}).

4.2 Functions

The *SingleLayer* class consists of seven functions. The class also references four generic modules (*NumPy*, *SciPy.special*, *math*, and *cmath*), and one new module (*HoogPy*). The functions are:

- *SetRequiredProperties(K, H, S, Q, rw, rb, LLBC, LLstorage, InjectionEnd)*: takes the list of hydraulic conductivities (*[K]*), the list of layer thicknesses (*[H]*), the list of storativities (*[S]*), the injection rate (Q), the well radius (*rw*, optional parameter, default set to -1), the boundary radius (*rb*, optional parameter, default set to -1), leaky layer boundary condition (*LLBC*, optional parameter, default set to 'constant'), a flag if leaky layer storage should be used (*LLstorage*, optional parameter, default set to True), and the time at which injection ends (*InjectionEnd*, optional parameter, default set to -1). *SetRequiredProperties* needs to be run before any of the other functions in *SingleLayer.py*, because it sets the global variables on which the other functions rely. This function does not return any value upon completion.

- *SetProperty(proptype, value)*: sets individual model properties specified by *proptype* to the value or list of values given in *value*. The following properties can be changed using *SetProperty*: '*K*' list of hydraulic conductivities (list of floats), '*H*' list of layer thicknesses (list of floats), '*S*' list of storativities (list of floats), '*Q*' volumetric injection rate (float), '*rw*' well radius (float), '*rb*' boundary radius (float), '*LLBC*' leaky layer boundary condition (string), '*LLstorage*' leaky layer storage flag (boolean), and '*InjectionEnd*' end time of injection (float). It is the user's responsibility to ensure that the lists *K*, *H*, and *S* have the same length. After changing model prperties it is necessary to re-compute the dimensionless parameters using the function *SetDimensionlessParameters*. *SetProperty* does not return any value upon completion.

- *SetDimensionlessParameters()*: computes and stores the dimensionless variables (*dimenTime*, *HeadFactor*, *rDw*, *rDb*, *sigmab*, *sigmat*, *lambdat*, *lambdab*, *mt*, and *mb*).

- *GetHeadForRadiusTime(radius, time)*: gives the head for a given radius and time. Returns the value of head upon completion.

- *GetRadiusForHeadTime(head, time)*: gives the radius for a given value of head and time. This function takes longer than *GetHeadForRadiusTime* as finding the radius is an iterative process requiring multiple head computations. Returns the radius upon completion.

- *GetTimeForRadiusHead(radius, head)*: gives the time for a given value of head to be reached at a given radius. This function takes longer than *GetHeadForRadiusTime* as finding the time is an iterative process requiring multiple head computations. Returns the time upon completion.

- *LaplaceHead(p)*: evaluates the Laplace transformed head for a given value of the Laplace variable *p*. This function is intended for internal use only. It contains the equations derived in [3] and [4].

The new module is *HoogPy.py* which contains the numerical inversion of the Laplace transform. This module is intended for internal use only.

5 Example

This example shows how the *SingleLayer* class is used to compute the pressure disturbance for several scenarios. This script is given in *Example.py*, and the modules *SingleLayer.py*, *Utilities.py*, and *HoogPy.py* should either be in same directory as *Example.py* or be located within the Python search path.

```
import SingleLayer as sl

# model properties
Ha = 500.0 #m, aquifer thickness
Ht = 200.0 #m, thickness of top leaky layer
Hb = 100.0 #m, thickness of bottom leaky layer
Q = 0.1 #m^3/s, injection rate
rw = 1.0 #m, well radius
rb = 60000.0 #m, boundary radius
Ka = 1e-5 #m/s, aquifer hydraulic conductivity
Kt = 4e-12 #m2, hydraulic conductivity of top leaky layer
Kb = 4e-11 #m2, hydraulic conductivity of bottom leaky layer
Sa = 2e-3 #storativity of aquifer
St = 8e-4 #storativity of top leaky layer
Sb = 4e-4 #storativity of bottom leaky layer

#Specify when to generate results (time from injection start [s])
times = [1e2, 1e4, 1e6, 1e8, 1e9]
#Specify what pressure head [m] to generate radii for
contours = [0.1, 1.0, 10.0, 25.0]
#Specify distance from well [m]
radius = [10.0, 50.0, 100.0, 1000.0, 5000.0, 10000.0, 50000.0]
```

```
#initialize the model
model = sl.SingleLayer()
model.SetRequiredProperties([Ka, Kt, Kb], [Ha, Ht, Hb],  \
                            [Sa, St, Sb],Q, rw, rb,  \
                            'constant', True, -1)
model.SetDimensionlessParameters()

#Generate pressures for specified locations and times
print 'time [s], distance [m], head [m] in bounded domain'
for tim in times:
    for dis in radius:
        print tim, dis, model.GetHeadForRadiusTime(dis, tim)

#Generate pressure contours for specified pressures and times
print 'time [s], distance [m], head [m] in bounded domain'
for tim in times:
    for con in contours:
        print tim, model.GetRadiusForHeadTime(con, tim), con

#Generate times to reach pressure at specified distance
print 'time [s], distance [m], head [m] in bounded domain'
for rad in radius:
    for con in contours:
        print model.GetTimeForRadiusHead(rad, con), rad, con

#change model to infinite domain
model.SetProperty('rb', -1)
model.SetDimensionlessParameters()
#Generate pressures for specified locations and times
print 'time [s], distance [m], head [m] in an infinite domain'
for tim in times:
    for dis in radius:
        print tim, dis, model.GetHeadForRadiusTime(dis, tim)

#change change leaky layer boundary conditions
model.SetProperty('rb', rb)
model.SetProperty('LLBC', 'noflow')
model.SetDimensionlessParameters()
```

```
#Generate pressures for specified locations and times
print 'time [s], distance [m], head [m] in a bounded domain'
for tim in times:
    for dis in radius:
        print tim, dis, model.GetHeadForRadiusTime(dis, tim)

#end injection after t=1e7
model.SetProperty('LLBC', 'constant')
model.SetProperty('InjectionEnd', 1e7)
model.SetDimensionlessParameters()
#Generate pressures for specified locations and times
print 'time [s], distance [m], head [m] in an infinite domain'
for tim in times:
    for dis in radius:
        print tim, dis, model.GetHeadForRadiusTime(dis, tim)
```

Running the *Example.py* script should lead to the following output:

```
time [s], distance [m], head [m] in bounded domain
100.0 10.0 2.91414034404
100.0 50.0 0.040420593118
100.0 100.0 6.95250958598e-006
100.0 1000.0 0.0
100.0 5000.0 0.0
100.0 10000.0 0.0
100.0 50000.0 0.0
10000.0 10.0 10.0757609169
10000.0 50.0 4.99074970171
10000.0 100.0 2.90027672417
10000.0 1000.0 6.60479246298e-006
10000.0 5000.0 1.32549191535e-027
10000.0 10000.0 0.0
10000.0 50000.0 0.0
1000000.0 10.0 17.3898492206
1000000.0 50.0 12.2672484958
1000000.0 100.0 10.0621317644
1000000.0 1000.0 2.88938376457
1000000.0 5000.0 0.0390984120527
```

```
1000000.0 10000.0 6.44056440906e-006
1000000.0 50000.0 1.1760221485e-027
100000000.0 10.0 24.5898744968
100000000.0 50.0 19.4668809204
100000000.0 100.0 17.2605440205
100000000.0 1000.0 9.93339945647
100000000.0 5000.0 4.85778825333
100000000.0 10000.0 2.7866388005
100000000.0 50000.0 0.0367031105509
1000000000.0 10.0 28.8615415939
1000000000.0 50.0 23.7385439733
1000000000.0 100.0 21.5321948975
1000000000.0 1000.0 14.2035977629
1000000000.0 5000.0 9.09601917938
1000000000.0 10000.0 6.93305193048
1000000000.0 50000.0 2.9534891292
time [s], distance [m], head [m] in bounded domain
100.0 42.8019022599 0.1
100.0 21.2427652424 1.0
100.0 1.03138644018 10.0
100.0 no contour 25.0
10000.0 426.631851822 0.1
10000.0 211.577223894 1.0
10000.0 10.2411166601 10.0
10000.0 no contour 25.0
1000000.0 4256.97660283 0.1
1000000.0 2109.20336965 1.0
1000000.0 101.973240507 10.0
1000000.0 no contour 25.0
100000000.0 41731.007084 0.1
100000000.0 20472.5743292 1.0
100000000.0 979.267213543 10.0
100000000.0 8.79110822437 25.0
1000000000.0 no contour 0.1
1000000000.0 no contour 1.0
1000000000.0 3755.88184481 10.0
1000000000.0 33.6403058319 25.0
time [s], distance [m], head [m] in bounded domain
```

```
5.14417 10.0 0.1
21.721 10.0 1.0
9534.3 10.0 10.0
131030000.0 10.0 25.0
136.65 50.0 0.1
557.32 50.0 1.0
239270.0 50.0 10.0
1407000000.0 50.0 25.0
548.47 100.0 0.1
2232.2 100.0 1.0
961500.0 100.0 10.0
2213800000.0 100.0 25.0
54980.0 1000.0 0.1
223980.0 1000.0 1.0
104480000.0 1000.0 10.0
6473200000.0 1000.0 25.0
1380600.0 5000.0 0.1
5672100.0 5000.0 1.0
1288100000.0 5000.0 10.0
15420000000.0 5000.0 25.0
5553700.0 10000.0 0.1
23058000.0 10000.0 1.0
2057400000.0 10000.0 10.0
pressure not reached 10000.0 25.0
138250000.0 50000.0 0.1
426350000.0 50000.0 1.0
3883900000.0 50000.0 10.0
pressure not reached 50000.0 25.0
time [s], distance [m], head [m] in an infinite domain
100.0 10.0 2.91414034404
100.0 50.0 0.040420593118
100.0 100.0 6.95250958598e-006
100.0 1000.0 0.0
100.0 5000.0 0.0
100.0 10000.0 0.0
100.0 50000.0 0.0
10000.0 10.0 10.0757609169
10000.0 50.0 4.99074970171
```

```
10000.0 100.0 2.90027672417
10000.0 1000.0 6.60479246298e-006
10000.0 5000.0 1.32549191535e-027
10000.0 10000.0 0.0
10000.0 50000.0 0.0
1000000.0 10.0 17.3898492206
1000000.0 50.0 12.2672484958
1000000.0 100.0 10.0621317644
1000000.0 1000.0 2.88938376457
1000000.0 5000.0 0.0390984120527
1000000.0 10000.0 6.44056440906e-006
1000000.0 50000.0 1.17602214849e-027
100000000.0 10.0 24.5898738815
100000000.0 50.0 19.4668803051
100000000.0 100.0 17.2605434052
100000000.0 1000.0 9.93339883178
100000000.0 5000.0 4.85778738352
100000000.0 10000.0 2.78663689515
100000000.0 50000.0 0.034483544231
1000000000.0 10.0 27.9180526856
1000000000.0 50.0 22.7950544995
1000000000.0 100.0 20.5887036545
1000000000.0 1000.0 13.2598729376
1000000000.0 5000.0 8.1466285517
1000000000.0 10000.0 5.96591640338
1000000000.0 50000.0 1.38813497408
time [s], distance [m], head [m] in a bounded domain
100.0 10.0 2.91414034404
100.0 50.0 0.040420593118
100.0 100.0 6.95250958598e-006
100.0 1000.0 0.0
100.0 5000.0 0.0
100.0 10000.0 0.0
100.0 50000.0 0.0
10000.0 10.0 10.0757609169
10000.0 50.0 4.99074970171
10000.0 100.0 2.90027672417
10000.0 1000.0 6.60479246298e-006
```

```
10000.0 5000.0 1.32549191535e-027
10000.0 10000.0 0.0
10000.0 50000.0 0.0
1000000.0 10.0 17.3934175046
1000000.0 50.0 12.2708124986
1000000.0 100.0 10.065684674
1000000.0 1000.0 2.8922319237
1000000.0 5000.0 0.0392309610606
1000000.0 10000.0 6.48229729372e-006
1000000.0 50000.0 1.21400417448e-027
100000000.0 10.0 24.6235845052
100000000.0 50.0 19.5005903069
100000000.0 100.0 17.2942516917
100000000.0 1000.0 9.96695518677
100000000.0 5000.0 4.8890987087
100000000.0 10000.0 2.81333505809
100000000.0 50000.0 0.0379749318883
1000000000.0 10.0 29.1622162047
1000000000.0 50.0 24.0392174865
1000000000.0 100.0 21.8328653876
1000000000.0 1000.0 14.5040022728
1000000000.0 5000.0 9.39251922531
1000000000.0 10000.0 7.22143032755
1000000000.0 50000.0 3.18051786965
time [s], distance [m], head [m] in an infinite domain
100.0 10.0 2.91414034404
100.0 50.0 0.040420593118
100.0 100.0 6.95250958598e-006
100.0 1000.0 0.0
100.0 5000.0 0.0
100.0 10000.0 0.0
100.0 50000.0 0.0
10000.0 10.0 10.0757609169
10000.0 50.0 4.99074970171
10000.0 100.0 2.90027672417
10000.0 1000.0 6.60479246298e-006
10000.0 5000.0 1.32549191535e-027
10000.0 10000.0 0.0
```

```
10000.0 50000.0 0.0
1000000.0 10.0 17.3898492206
1000000.0 50.0 12.2672484958
1000000.0 100.0 10.0621317644
1000000.0 1000.0 2.88938376457
1000000.0 5000.0 0.0390984120527
1000000.0 10000.0 6.44056440906e-006
1000000.0 50000.0 1.1760221485e-027
100000000.0 10.0 0.160534392407
100000000.0 50.0 0.160533847488
100000000.0 100.0 0.16053219412
100000000.0 1000.0 0.16033078392
100000000.0 5000.0 0.155831853374
100000000.0 10000.0 0.143006669273
100000000.0 50000.0 0.0115158191307
1000000000.0 10.0 0.0324142696959
1000000000.0 50.0 0.032414268567
1000000000.0 100.0 0.0324142653172
1000000000.0 1000.0 0.0324139558651
1000000000.0 5000.0 0.0324087658349
1000000000.0 10000.0 0.0323964157624
1000000000.0 50000.0 0.0322635331933
```

References

[1] C. V. Theis, The relation between lowering of the piezometric surface and the rate and duration of discharge of a well using ground water storage, Trans. Am. Geophys. Union 16th anual meeting (Pt. 2) (1935) 519–524.

[2] M. S. Hantush, C. E. Jacob, Nonsteady radial flow in an infinite leaky aquifer, Trans. Am. Geophys. Union 36th anual meeting (Pt. 1) (1955) 95–100.

[3] A. F. Moench, Transient flow to a large-diameter well in an aquifer with storative semicofining layers, Water Resourses Research 21 (8) (1985) 1121–1131.

[4] Q. Zhou, J. T. Birkholzer, C.-F. Tsang, A semi-analytical solution for large-scale injection-induced pressure perturbation and leakage in a laterally bounded aquiferaquitard system, Transport in Porous Media 78 (1) (2009) 127–148, doi: 10.1007/s11242-008-9290-0.