**Applied Hydrogeology Practical Weeks 10 and 11**

For this two week exercise you will build on the model you created during previous FloPy practicals. You will move from the simple single layer model, to a model of a layered aquifer, and simulate a well pumping from a confined aquifer.

You will then use this model to a) calculate source protection zones around the pumping well and b) consider the potential contamination of the well following a release of pollutants from a point source in the catchment.

Remember – there is some code development from the model you have already generated – this mostly involves relatively simple developments which test your understanding of indexing. Having a fully working code at the end of this exercise is the minimum you will require to score 50% in this assessment. Critically you also need to be able to demonstrate a hydrogeological understanding by fully answering the questions set (highlighted in red in this handout). Whilst it is possible to get a first class mark without answering the bonus questions these are potential extra marks you can gain.

Answers to questions in **BLUE** should be done as short comments to the code in the notebook, which will then be pushed to GitHub by 14.00 Wednesday 13th December.

Answers to questions in **RED** should be longer answers word processed in a separate document illustrated by screenshots. A hard copy of this should be handed to the School Office by 14.00 Wednesday 13th December.

You can use plt.savefig(……) – see online documentation - in place of screenshots to generate higher resolution figures if you would like to try this out.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Week 10 Practical Exercise**

Firstly, make a copy of the Unconfined Aquifer Practical from 2 weeks ago and place in a new directory of your choosing.

**PART 1: Setting up the three-layered 3D model**

**Add the following modules** (in addition to current ones – flopy and numpy);

import sys

import shutil

import matplotlib as mpl

import matplotlib.pyplot as plt #you can go ahead and remove the duplicate command underneath post processing of results – I’ve moved it here for asthetic reasons really

**Discretization**

**Start by changing the modelname to threelayer**

**Convert the model to;**

* 30 m thickness
* 3 layers of equal thickness

**At this stage both the time step and the Basic package parameters are fine, except that you should change the hydraulic head on the boundaries so that the hydraulic head at the site of the well before pumping is 5 m and the gradient across the model (h/l) is 2e-3. (This is done by changing the condeleft and conderight variables)**

**Layer-Property Flow package**

**For each of our 3 layers we want to define different aquifer properties.**

**Table 1 provides an initial estimate of rock properties for use in your modelling**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Layer 1 | Layer 2 | Layer 3 |
| Horizontal Hydraulic conductivity (m/d) | 50. | 0.1 | 10. |
| Vertical Hydraulic conductivity (m/d) | 50. | 0.01 | 1. |
| Specific Yield | 0.05 | 0.05 | 0.05 |
| Specific Storage | 1.e-4 | 1.e-4 | 1.e-4 |
| Laytyp | Unconfined | Confined | Confined |

Use a list structure to define each variable. Hint: this is the same way we define variables like perlen

**Transient General-Head Boundary Package**

According to Darcy’s Law

Q= -KA(h1-h0)/(X1-X0)

Conductance combines the K, A and X terms so that Darcy’s Law can be expressed as

Q=-C(h1-h0)

**Qu 1a.** Derive and explain the dimensions of C (conductance)

The flow package calculates the conductance between cells using an average hydraulic conductivity of the cells, the area of the interface between cells and the distance between the cell centres. Some of the head-dependent boundary conditions – including the General-Head Boundary - require the user to specify the conductance.

The General-Head Boundary package is the simplest of several head-dependent flux boundary packages.  In it, there is a linear relationship between the ground water flux into (or out of) the cell and the head in the cell. The user specifies a reference head and a conductance.  When the head in the cell equals the reference head, the flux is zero.  If the head in the cell is less than the reference head, water enters the groundwater system through the general-head boundary.  If the head in the cell is greater than the reference head, water leaves the groundwater system.

If you look in the first for loop you will see the hk variable referenced beside condleft and condright. Given we now have 3 different values for hydraulic conductivity we need to create the correct boundary condition for each cell depending on it’s hydraulic conductivity.

In a blank cell above or below run the following command;

hk[0]

Do you see how this indexing procedure works – change the 0 to a 1, 2 or a 3? What happens?– These are rhetorical questions – the goal is to firm up your ideas of indexing procedures.

**Use hk[XXX] in place of hk where XXX takes the value of the iteration variable.**

Insert **print ()** into various stages of for loops to get an idea of intermediary variables of hk at different stages of the for look.

**Divide zbot by 3 – Why? Answer as an adjacent comment.**

**Under the section** *#Make list for stress period 2***, delete the lines containing condleft and condright.**

**Copy the condleft and condright lines you just edited from the first ‘for il in range(nlay):’ into a similar place for the second ‘for il in range(nlay):’ loop.**

**Transient well package**

Currently the well is intersecting the 0’th layer (i.e. the 1st model layer – remember indexing in Python indexing starts at 0 – you just convinced yourself of this by looking at how hk[1] works)

**Change this so that it pumps out of the 3rd layer at a rate of 1.157 L/s (remember to use consistent units in your modelling – HINT the current value is 100 m3/day).**

**Output Control, Preconditioned Conjugate Gradient Packages, Writing the MODFLOW Data Files, Running the Modelling – Leave the same**

**Post-Processing the results**

You can delete import matplotlib.pyplot as plt as we already included it above in the initial set of imports. It does no harm here, but the advantage of moving this to the top of the script is that we will shortly be using it in the MODPATH simulations.

Leave import flopy.utils.binaryfile as is. We will only use it in this cell. That’s the asthetic rule of thumb; use something once - import close to the function, use it many times - import at the start.

At some point you may decide to alter the length of time we allow the simulation to run to (or you may not!). At present;

mytimes = [1.0, 101.0, 201.0]

This references the final timesteps at each of the three stress periods. Print out the variable times and see for yourself that the final timestep is 201.0

If we choose to change the length of the simulation we will need to change 201.0. **Using the times variable and indexing feed the last timestep (for a simulation of any length) into the mytimes variable in the place of XXX;**

mytimes = [1.0, 101.0, XXX]

**In the lines;**

plt.imshow(head[0, :, :], extent=extent, cmap='BrBG', vmin=0., vmax=10.)

plt.colorbar() # plot colourbar

CS = plt.contour(np.flipud(head[0, :, :]), levels=levels, extent=extent,    zorder=10)

head[0, : , :] refers to all rows and all columns of the 0th (i.e. 1st) layer.

**You can change the value of 0 to 2 to see the values of heads in the bottom layer**. This may come in handy when you want to view the effect of pumping on the bottom aquifer.

**Move all the code under # Plot the head versus time (i.e. from idx = (0, nrow/2 - 1, ncol/2 - 1) to plt.show()) to a new cell bellow.** At present this code only plots the data for the first layer. We are going to change this so that it plots data for the same row/column but for each layer.

**Add 2 new idx variables which reference layers 2 and 3 (hint in python this would be the 1st and 2nd)**

**Add 2 new ts variables using the new idx variables**

**Plot those two new variables on top of the existing head**

**Use the plt.plot(… label=’something’) argument to give each line a name**

**Use plt.legend() to display those labels on the graph.**

You should now see three lines and a legend on the plot.

**Qu 1b. Using the figure you generate- briefly (3-4 sentances) comment on the differences between these lines (HINT: Try changing vka in the middle (aquiclude) layer to get a clearer idea of whats going on and the reasons for the differences).**

**Run the function get\_kstpkper() on the headobj variable**. **What does this function do? Comment adjacent to code.** Go to the flopy documentation and type ‘get\_kstpkper’ into search. If you put the () in the search you won’t find it.

We wish to see the head data for the last time step of the last stress period. This is because we want to see the steady state head situation. In fact actually because we specified the second stress period to run to a steady state situation (steady=True) the very first time step of the second stress period is at steady state. See for yourself how changing the third value for the steady variable from True to False affects the model. **Paste the following into a cell;**

**hds = headobj.get\_data(kstpkper=(99, 2))**

Check in the documentation to see how this get\_data function works. Print out the hds variable and cross reference with the perlen and nstp variables so you can assure yourself you understand how everything works.

If you print hds[:,:,:] you will see that that it’s a numpy array. It’s a 3D numpy array – see for yourself by running; np.shape(hds)

It has a 3x10x10 shape at this stage. The command hds[:,:,:] calls for python to read out **all** (i.e. using a **colon**  **:** ) the layers (0th **:** ), rows (1st **:** ) and columns (2nd **:** ). Thus it generates a massive array of values.

So if we were to call hds[1,3,5] this would give us a single value corresponding to a cell in the 2nd layer, 4th row and 6th column.

**Adapt the following code to**;

* Generate cross-sections showing variation of head in both x and y plane – change the 2nd and 3rd indexing values (HINT – Only 1 index value needed)
* View distribution of head in the different layers – change the 1st indexing value

**plt.imshow(hds[x, x, x])**

**plt.colorbar();**

**Adapt the following code to**;

* Generate cross-sections of head values in various layers (HINT – You need 2 indexes)

**plt.plot(hds[x, x, x])**

The easiest way to copy and paste the data for a cross section plot you made above into something like excel is to print it out without the commas generated when you print a pure numpy array. e.g. to print out the values in the 3rd layer, 7th column we can loop through the array and print to the screen.

**for i in hds[2,7,:]:**

**print (i)**

**PART 2: Running tests using the layered 3D MODFLOW model**

Start by increasing the spatial resolution of the model – **change nrow and ncol to 30**. This should help us see the effects of the tests we are about to run better.

**BONUS POINTS (FOR THE COMPUTATIONALLY CURIOUS – answer in word document as BONUS 1):** What happens to the speed of the code, specifically the running of the executable, when we increase the resolution of the model? Plot your results. Talk to demonstrator if interested at looking at this problem.

**Change the pumping rate in stages from 1.157 L/s to 34.7 L/s**

**Q2a -** Can you see a problem evaluating the effect of pumping at the higher rate using this model? Write an explanation of this problem in the word document and include suitable plots as supporting evidence**.**

**Given your observations and interpretations in Q2a change the model dimensions. Remember to maintain a hydraulic gradient of 2e-3.**

**Q2b –** Explain with the aid of appropriate diagrams how changing the volume of the aquifer simulated has improved the model.

**Model validation**

K values for Layers 1 and 3 given in Table 1 are well constrained, derived from the geometric mean of a large number of pumping tests in these aquifers. K values for the aquiclude (Layer 2) given in Table 1 are the geometric mean of those derived from core plug laboratory tests as no pumping test values are available.

**Q3a.** Evaluate your model by attempting to simulate head measured in piezometers within Layers 1 and 3 (data in Table 2) at steady state when the well is pumped at **34.72 L/s**. How well do the simulated heads in the model using parameter values in Table 1 match the observed values? Quantify the degree of fit using the root mean squared error (RMSE)

where n is the number of observations, the predicted value predicted by the model and *yj* the observed value.

**Q3b.** Adjust values of hydraulic conductivites (kh and kva) for the aquiclude (layer 2) in your model to generate the best match. Aim for an RMSE of 0.1m between your predicted heads and the observed heads reported in Table 3. Discuss sensitivity of the predicted cone of depression in both aquifers to these changes and present your best fit model. Suggest reasons the core plug measurements of hydraulic conductivity for the aquiclude may have provided a poor fit to the observed drawdown.

Table 2 –Head measured in piezometers within the upper (layer 1) and lower (layer 3) aquifers at steady state pumping of 34.7 L/s. Positions of observation wells are given as distances in m relative to the well. Positive values are distances to the west and north of the well, and negative values are distances to the east and south of the well.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Piezometer | X (m) | Y (m) | Head (m) |
| 1 | A | 834 | 0 | 5.80 |
| 1 | B | 0 | -667 | 3.82 |
| 1 | C | 0 | 0 | 1.60 |
| 1 | D | -167 | 0 | 2.38 |
| 1 | E | 0 | 333 | 3.28 |
| 3 | F | 0 | 0 | 1.01 |
| 3 | G | 0 | 167 | 2.73 |
| 3 | H | -500 | 0 | 2.59 |
| 3 | I | 0 | 333 | 3.28 |