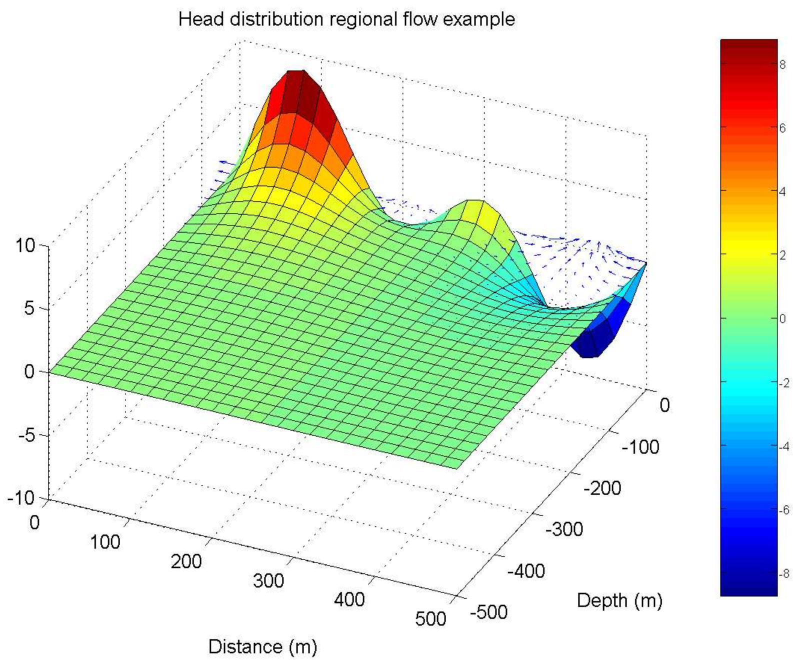
Groundwater flow modelling (450008)

Computer laboratories



dr. V.E.A. Post

Contents

Part I: Creating finite difference models in Python

Introduction

Nowadays, a wide selection of very powerful groundwater flow models is available. For almost every problem there is a code that suits your needs. But sometimes, the code at hand is not just what you want. It may be that it doesn't handle a specific boundary condition or you want the output to be in a slightly different format to make the post-processing easier. Wouldn't it be convenient then if you have the skills to modify the original code a bit, or even to create your own model?

Moreover, if you have made (simple) models yourself, you also better understand how existing codes work. This may be helpful in a situation where the model crashes or behaves unexpectedly in a different way.

Fewer and fewer hydrologists possess the skills to modify or write computer codes. So, learning to write your own models will give you an advantage. In order to do so, you also need to learn a programming language. A programming language is a set of functions and statements that allow you to pass commands on to the computer. There are dozens of different programming languages available. Examples include Visual Basic, Pascal, C, FORTRAN, Python and many, many more. In this course we will use Python. Python is a so-called command-line interpreter: you type in the commands that are subsequently executed. These commands can be combined into a program which we call a script. The advantage of using Python is that lots of the things you would normally have to worry about as a programmer have already been done for you. For example, creating plots and charts is extremely easy because the statements to send graphics to the screen have already been programmed. You can simply use the commands included in Python and its libraries like matplotlib instead of figuring out all this complicated stuff yourself.

Another advantage of Python is that it is becoming widely used in various parts of the scientific community. Popular geographic information systems like ArcGIS and GRASS also have Python functionality. In the groundwater industry, MATLAB, another interpreted programming language, is still the standard, although that also slowly starts to change. The reason that we do not use MATLAB in this course is that (i) Python is just as good and sometimes even better than MATLAB and (ii) Python is open-source software. The latter means that there are no licensing issues and that you can install and work with Python on any computer, for example at your home or on a laptop during your fieldwork.

So, in short, being able to write your own modeling code helps you to better understand what you are doing if you are using an existing model. Also, it gives you the ability to modify and improve models or create custom-made models. In the learning process, you will become familiar with the basics of programming languages, which you can apply to other areas outside groundwater flow modeling.

It is assumed that you are familiar with the basics of Python. If not, or if you want to refresh your memory, then first familiarize yourself by going through the Python tutorial, which is available from your instructor.

Chapter 1

Calculating heads

1.1 Solution methods

During the lectures we have looked at the solution of unknown heads at the interior nodes of a grid for which the heads at the boundary nodes are known (Dirichlet boundary condition). We have seen that by ‘moving’ the so-called five-star operator through the grid, the head at each node could be calculated as a function of the heads in the neighboring nodes. Calculating the heads once, however, was not enough to obtain the final solution. Instead, we started with an initial guess and then repeated the calculations until the heads no longer changed significantly (iteration).

1.1.1 Gauss-Seidel iteration using a spreadsheet

Such calculations are easily done by computers. Before creating a program in Python, let’s look at how we can use a spreadsheet to do these calculations. Suppose we have a grid as in figure 1.1 with the heads given for each cell on the model boundary.

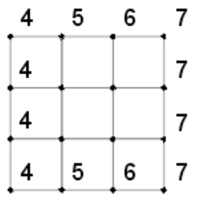
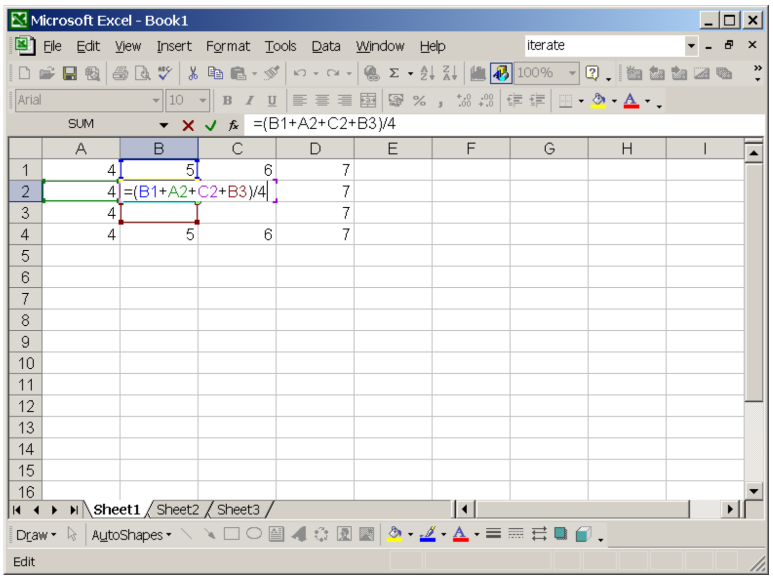


Figure 1.1: Simple mesh with heads fixed on the boundaries.

We can easily imagine the cells of a spreadsheet to coincide with the nodes of the model mesh. To do the calculations, proceed as follows:

- Open your spreadsheet program.

- Type in the fixed head values at the cells representing the model boundary.
- Type a formula in the upper left interior cells that calculates the average of the heads in the 4 neighboring cells (see figure 1.2).
- Copy this formula to the remaining interior cells.



SUM =(B1+A2+C2+B3)/4

	A	B	C	D
1	4	5	6	7
2	4	=(B1+A2+C2+B3)/4		7
3	4			7
4	4	5	6	7

Figure 1.2: Entering the formula for Gauss-Seidel iteration in Excel.

It is that easy! The only thing is that, depending on the spreadsheet you use, you may need to iterate manually. In Microsoft Excel, this is done automatically if you select Tools → Options and then on the Calculations tab enable the Iteration option.

1.1.2 Gauss-Seidel iteration using Python

The advantage of using a spreadsheet is that it is easier to envisage the model structure because, like a finite-difference model, a spreadsheet consists of cells. In the remainder of the exercises, however, we will use Python because of its much greater flexibility. As an example, take a look at the following Python-script, which does exactly the same as what we did before in our spreadsheet.

```
from numpy import array

h = array([[4., 5., 6., 7.],
           [4., 0., 0., 7.],
           [4., 0., 0., 7.],
           [4., 5., 6., 7.]])
```

```
dummy = h.shape
nrow = dummy[0]
ncol = dummy[1]

print 'Head matrix is a ', nrow, ' by ', ncol, ' matrix.'

ni = 1
conv_crit = 1e-3
converged = False

while (not converged):
    max_err = 0
    for r in range(1, nrow - 1):
        for c in range(1, ncol - 1):
            h_old = h[r, c]
            h[r, c] = (h[r - 1, c] + h[r + 1, c] + h[r, c - 1] + h[r, c + 1]) / 4.
            diff = h[r, c] - h_old
            if (diff > max_err):
                max_err = diff

    if (max_err < conv_crit):
        converged = True

    ni = ni + 1

print 'Number of iterations = ', ni - 1

print h
```

At first sight, this may look incomprehensibly complicated. It is certainly more complicated than entering the formulas in a spreadsheet. But if you get used to programming it will become much easier to 'read' such scripts. Note how indentation is used to structure the script. In fact, you have to indent in Python, otherwise it will complain! Try if you can understand what is going on here and then answer the following questions:

- What is the purpose of the variable converged?
- What values does the shape variable return? Can you think of a reason why it is better to use 'shape' to define the number of rows and columns rather than just assigning fixed values to these variables yourself?
- What is the name of the variable that is used to store the number of iterations?
- How does the program know that it only needs to calculate the interior nodes?

Exercise: The script is available on Blackboard. Download it to a local disk and open it in Python (simply by clicking File → Open and selecting it). The script is opened in the script editor and can be run by pressing F5. Investigate the effects of the convergence criterion and initial head guess on the number of iterations. Set the convergence criterion to different values and note the number of iterations. Do the same for the initial heads of the interior nodes.

1.1.3 Successive over relaxation

A way to speed up the convergence is to use Gauss-Seidel iteration in combination with successive over relaxation (SOR). In this method, the difference between the calculated value at the new iteration interval and that of the previous iteration interval $c = h_{i,j}^{m+1} - h_{i,j}^{m}$ is multiplied by a relaxation factor ω. The new value of $h_{i,j}^{m+1}$ becomes:

$$h_{i,j}^{m+1} = h_{i,j}^{m} + \omega c \tag{1.1}$$

This equation can be written as:

$$h_{i,j}^{m+1} = (1-\omega)h_{i,j}^{m} + \omega \frac{h_{i-1,j}^{m+1} + h_{i,j-1}^{m+1} + h_{i+1,j}^{m} h_{i,j+1}^{m}}{4} \tag{1.2}$$

Exercise: Open the example script of Gauss-Seidel iteration and modify it in such a way that it can incorporate SOR. What extra variable(s) are needed? Note that max_err can become negative with SOR and that you need to take care that you compare the absolute value of max_err to the convergence criterion. Use a value of $\omega = 1.1$.

The number of iterations is now .... Increase the value of ω and run the script again. Note that when $\omega > 1.25$ the number of iterations increases compared to the script without SOR! This is because for high values of ω the calculated heads during the first iterations overshoot the final 'true' values and it takes some time to converge back towards these values.

1.1.4 Direct solution

The previous exercise shows that the heads at the interior nodes are readily calculated using Gauss-Seidel iteration, either with or without SOR. Iterative solutions, however, are not the most efficient way to solve the system of finite difference equations. Direct solution methods are an alternative, more efficient way and are easily applied in Python. Remember that a system of linear equations (such as the finite-difference expressions for this problem) can be written in matrix form:

$$[A]\vec{h} = \vec{f} \tag{1.3}$$

where $[A]$ is a coefficient matrix, $\vec{h}$ a column vector of unknown heads and $\vec{f}$ a column vector containing all known values. $\vec{h}$ follows from:

$$\vec{h} = [A]^{-1}\vec{f} \tag{1.4}$$

where $[A]^{-1}$ is the so-called inverse of $[A]$.

For the problem presented above, write down a finite-difference expression for each unknown head at the interior nodes. The result is a system of 4 linear equations with 4 unknowns. Transfer all known values (the heads at the boundaries) to the right-hand side and express the system of equations in matrix format. Then use Numpy's linear algebra package to solve for $\vec{h}$ (see page 19 – 20 of the Python tutorial on how to do this). Did you get the same result as before?

Although direct solution methods are more efficient than iterative methods they require large amounts of computer memory for real-world numerical models. Therefore, most codes use solution schemes that combine the best of both worlds of iterative and direct solution techniques.

1.2 Regional flow example

The previous example was very basic and not of much use in real-world modeling. In the next example, we will use our numerical model to analyze flow systems in a topographically-driven groundwater system. In the course 'Groundwater hydraulics' you derived an analytical solution for this problem and did a numerical calculation with FlexPDE. We will now modify our script so we are able to solve the problem ourselves. We will combine the model with the powerful graphical capabilities of Python's Matplotlib package to visualize the output. In that way, we have already created a tool that starts to look like the industry-standard modeling packages!

Take the script for Gauss-Seidel iteration with SOR as your starting point. First save the script under a different name. Set the value of omega to $\omega = 1.8$ and set the convergence criterion to $1 \cdot 10^{-5}$. As a first step, we will increase the number of rows and columns of the model from 4 by 4 to 26 by 26. An easy way to so is to use the function zeros() to create the matrix h. In that way, all the starting heads will be set to 0 automatically. Look up how the function works and insert the appropriate statement in the script. Note that you have to import the function `zeros()` from numpy, just like the function `sin()` and `pi` (see below).

We use the same configuration as for the exercise in 'Groundwater hydraulics', so our model will measure 500 by 500 m. What will be the width of the grid cells (note: we have a mesh-centered grid!). Declare a variable called dx and set its value to the appropriate grid cell width. We will use square cells, so there is no need to define the height of the cells explicitly.

The heads at the top of the model can be defined by superposition of 2 sine functions with different wavelengths. This is expressed in the formula:

$$h = A_1 * sin(k_1 * x) + A_2 * sin(k_2 * x) \tag{1.5}$$

where A_1 and A_2 are the amplitudes of the respective waves, $k = 2 * \pi n / L$, n is the wave number and L is the width of the model. Modify the script to include the parameters L, A_1, A_2, n_1, n_2, k_1 and k_2. Set the values of $A_1 = A_2 = 5.0$ m and $n_1 = 1$ and $n_2 = 2$.

Equation 1.5 contains the variable x, so we need to know x for each column in the grid. Try to find the right statements to accomplish this and then implement equation 1.5 in the script. Once you have done this, you're almost ready to start the calculations. As a final step, include this statement at the beginning of the script:

```
from numpy import *
```

Include the following statements at the end of the script to produce graphical output:

```
close()
f = figure()
[X, Z] = meshgrid(linspace(0, L, ncol), linspace(0, -L, nrow))
contourf(X, Z, h)

colorbar()

[dhz, dhx] = gradient(h)
quiver(X, Z, -dhx, dhz, color = ’w’)
```

Look up what these functions do. Can you understand what they mean? If you do you’re now ready to start calculating your first real-world, self-made groundwater flow model!

1.2.1 No-flow boundaries

Note that we didn’t worry about the no-flow boundary at the bottom of the model in the previous exercise. How can you see from the contour lines of the hydraulic heads that the bottom boundary is not a no-flow model? In the next exercise we will implement the no-flow boundary. We only need to make a few adjustments.

Remember from the lectures that a way to incorporate no-flow boundaries is to add imaginary nodes outside the model domain. The type of grid determines how the no-flow boundary is implemented. For a mesh-centered grid, the gradient across the boundary becomes zero (the condition for no-flow) if $h_{i,j+1} = h_{i,j-1}$, where $h_{i,j+1}$ the head at the the imaginary node outside the model domain and $h_{i,j-1}$ is the head at first node inside the model boundary.

We will expand our matrix at the bottom with one additional row, representing the imaginary nodes. To do so, change the declaration of matrix h. Then implement the no-flow boundary by adding the following line:

```
h[-1, :] = h[-3, :]
```

Specifying `-1` at the row index is short in Python for the last row; `-3` indicates the third-last row. The colon at the column index means ‘all columns’. So this statement assigns the heads of the third-last row to the heads of the last row, for all columns. Where would you insert this line into our script? If you add it your script is almost ready to handle the no-flow boundary. Before you start the calculation though, you need to make some minor changes to the statements that produce the graphics. These prevent the imaginary cells from being displayed. The correct syntax is:

```
close()
f = figure()
[X, Z] = meshgrid(linspace(0, L, ncol), linspace(0, -L, nrow - 1))
contourf(X, Z, h[:-1, :])

colorbar()

[dhz, dhx] = gradient(h[:-1, :])
quiver(X, Z, -dhx, dhz, color = ’w’)
```

Only 3 lines are different. Study the differences and make sure you know what they mean. Then start the calculations and pay particular attention to the head contours near the bottom boundary. What has changed?

Chapter 2

Calculating flows

2.1 Water budgets

In the previous examples the convergence criterion controls the accuracy of the solution. As a second check on the accuracy, a water balance can be set up. Remember from the lectures that for a 2D model with square grid cells:

$$Q = -k\frac{\Delta h}{\Delta x}\Delta x = -k\Delta h \tag{2.1}$$

Assume that the hydraulic conductivity (k) is 10 m/d. Extend the script of the regional flow example to calculate the water balance of the whole model domain. Calculate inflow and outflow of each model boundary except for the bottom boundary. Make sure to adopt a sign-convention for inflow (+) and outflow (-). Fill in the table below.

Table 2.1: Water budget for regional flow example

flow component	magnitude
q_{left}	$\cdots$
q_{right}	$\cdots$
q_{top}	$\cdots$
q_{bottom}	$\cdots$
q_{total}	$\cdots$

Compare the net inflow (inflow - outflow) to the magnitude of the inflow/outflow. Is the error in the water balance acceptable?

Note that the *net* flow over the top model boundary is basically zero but that the inflow and outflow components themselves over this boundary are not. What are the magnitudes of the inflow and outflow over the top boundary?

2.2 Abstraction well

In this exercise we calculate the drawdown due to a well that fully penetrates a confined aquifer. Flow is horizontal and the aquifer is isotropic. Under these conditions and when $\Delta x = \Delta y$, the formula for Gauss-Seidel iteration takes the following form (check your lecture notes):

$$h_{i,j} = \frac{h_{i-1,j} + h_{i,j-1} + h_{i+1,j} + h_{i,j+1} + \Delta x^2 R/T}{4} \tag{2.2}$$

where T is the transmissivity (m^2/day) and R is the volume of recharge/discharge per unit time per unit surface area. R and Q are related by (for square grid cells, so $\Delta x = \Delta y$):

$$R = \frac{-Q}{\Delta x^2} \tag{2.3}$$

Transmissivity is $T = 300$ m^2/day. The well is located at $x = 0$, $y = 0$ and has a discharge of $Q = 2000$ m^3/day. The value of R is for the infinitesimal volume around the well (figure 2.1). Outside this volume $R = 0$.

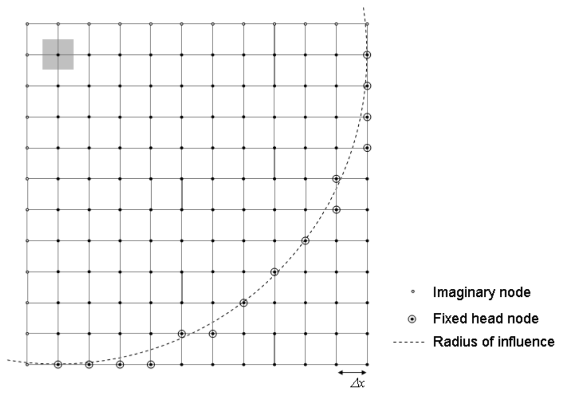


Figure 2.1: Finite difference grid for abstraction well exercise. Modified from Wang and Anderson (1982).

Because the equipotential lines will be cylindrical around the well, the problem is symmetric and we only need to consider one quarter of the problem domain. Let's model the lower-right quardrant so the left and upper boundary are the no-flow boundaries (figure 2.1). The analytical solution for this problem is given by the Thiem equation:

$$h = h_0 + \frac{Q}{2\pi T} \ln \frac{r}{r_{max}} \tag{2.4}$$

where h_0 is the head before pumping and r_{max} is the radius of influence of the pumping well. Assume for this exercise that $r_{max} = 2000$ m. Note that although time is not in this formula, this is actually not a steady-state problem: The

Thiem equation calculates the heads for a given r_{max}. Check your lecture notes of ‘Groundwater hydraulics’ to see how r_{max} varies with time.

As before, take the script for Gauss-Seidel iteration with SOR as your starting point. Set the value of omega to $\omega = 1.8$ and set the convergence criterion to $1 \cdot 10^{-3}$. Use a (mesh-centered) grid of 12 rows by 12 columns (of which the first row and column are imaginary nodes!). Use $\Delta x = \Delta y = 200$ m. Make the following modifications to the script (not all of them are straightforward, so ask for help if you don’t succeed yourself):

- Calculate the distance of each node to the well and find the nodes whose distance most closely match the radius of influence ($r_{max} = 2000$ m). Set the heads of these nodes and of those outside the radius to $h = 10$ m. Set the starting heads of all remaining nodes to $h = 5$ m.

- Include lines to make the left and upper boundaries no-flow boundaries.

- Modify the statements that perform the Gauss-Seidel iteration in such a way that they can take into account the abstraction well (equation 2.2).

- Make sure that the heads are only calculated for the nodes that are part of the area of influence of the well. In other words, don’t calculate the heads of the nodes that you have set to $h = 10$ m. You will have to think of a smart trick to accomplish this.

- Include the options for graphical output at the end.

- Finally, add a program line to calculate the analytical solution with the Thiem equation so you can compare it to the numerical result.

You will find that the analytical and numerical results agree very well. You could include a water balance calculation but that involves quite a bit of administration since the inflow here is not simply through the right and bottom grid boundaries but across the circumference formed by the sides of the cells that are mark the radius of influence.

Chapter 3

Transient simulations

3.1 Abstraction well (transient)

The previous exercises all assumed steady-state conditions. In this exercise the well-drawdown problem will be expanded to a transient simulation. As before, only the lower-left quadrant, measuring 2000 by 2000 m, is considered but now all the boundaries are no-flow boundaries and the cell size is $\Delta x = \Delta y = 100$ m.

The numerical results will be compared to the analytical solution by Theis that calculates the drawdown at a radius (r) from the well:

$$h_0 - h = \frac{Q}{4\pi T} W(u) \tag{3.1}$$

where

$$W(u) = \int_u^\infty \frac{e^{-\psi}}{\psi} d\psi \tag{3.2}$$

and

$$u = \frac{r^2 S}{4Tt} \tag{3.3}$$

$W(u)$ is called the well functions and is usually tabulated in textbooks. It is also implemented in Python so there is no need to look it up: you simply calculate it with Python! The input file for this exercise has been prepared for you (it is available via Blackboard). Study it and answer the following questions:

1. What is the function of the parameters alpha and n_steps?

2. What is the size of the time steps?

3. See if you can find the lines that calculate the analytical solution. What is the Python equivalent of the well function $W(u)$?

4. At what distance are the analytical solution and the numerical solution compared?

After you have studied the script, run it and observe the graph that is produced. It compares the fit between the analytical and numerical results.

1. At what time do they start to deviate and why is this?
2. What causes the smaller deviations before this time?
3. Modify the script to have it perform fully-implicit calculations. What is the effect on the number of iterations?
4. Then change to a fully-explicit formulation. What happens? To prevent this, only one line of the code needs to be changed. Which one? And how would you change it? Make the change and discuss the implications.

To finish, uncomment the lines that produce the 3D graph of the heads during the calculations. A 'movie' will be displayed on the screen that shows the change in heads over time.

In a relatively short time you have mastered Python, learned to program your own modelling codes and became a movie producer. Not bad, don't you think?!

Part II: Creating finite difference models in MODFLOW

Chapter 4

Well field in a river valley

This exercise is a modified version of tuturial 3 from Chiang (2005). It serves to (1) become familiar with the input file structure of MODFLOW, (2) compare 3D and quasi-3D vertical discretization methods, (3) illustrate the use of the MODFLOW Well and River packages and (4) practice the interpretation of the water balance. The files that are needed for this exercise are available on Blackboard.

Problem description

A river flows through a valley, which is bounded to the north and south by impermeable granite intrusions (figure 4.1). The hydraulic heads in the valley at the upstream and downstream model boundaries are known. The river forms part of a phreatic aquifer, which overlies a confined aquifer of a variable thickness. A silty layer with a thickness of 0.5m separates the two aquifers. A well field consisting of 3 pumping wells is to be installed, which will be abstracting groundwater at a proposed rate of $Q = 500$ m^3/day from the confined aquifer. The question is how that well field will affect the discharge of the river, which provides water to a valuable ecosystem downstream.

The relevant hydraulic parameters of the aquifer system are listed in table 4.1.

Model setup

Several files are available on Blackboard with information on the geometry and the heads of the aquifer system. Start by downloading these files and put them in your working directory.

We will initially simulate the system with a quasi-3D model (steady-state). That means that the silt layer is not explicitly included in the model. Instead, we enter an appropriate value of the vertical leakance. Hence, the head in the silt layer is not calculated but the exchange of water between the upper and the lower aquifer through the silt layer is. To set up the model, proceed as follows:

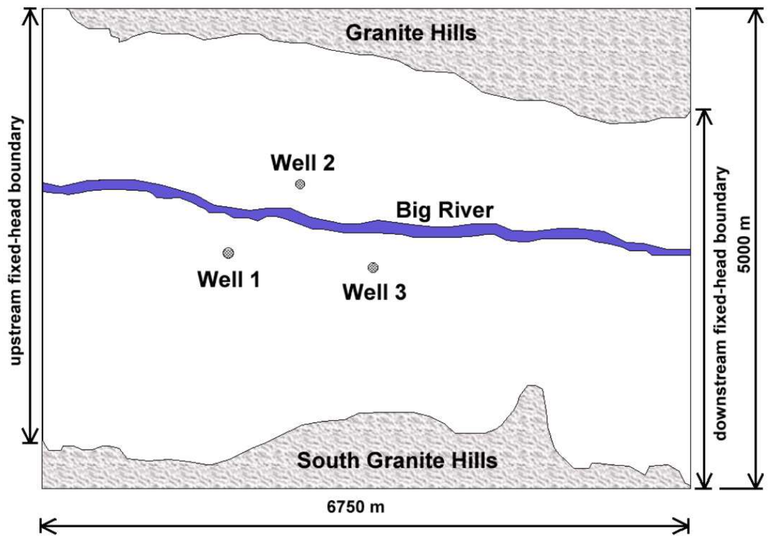


Figure 4.1: Configuration of the model

- Create a new model in PMWIN and lay out the grid. Use 2 model layers (each representing a single aquifer), 20 rows and 27 columns. You can find the model extent from figure 4.1.
- Load the basemap for this exercise by going to Options → Map in the grid editor. Right click on the DXF File field and open the file basemap.dxf (which you just copied from Blackboard). Make sure you check the box in front of the filename.
- The map will be loaded but it is shifted relative to the grid. You can move the grid by selecting Options → Environment and entering Xo = 200 and Yo = 6000 in the Coordinate System tab.
- You can already refine the grid at the location of the well field. To do so, leave the Grid Editor and enter it again. Then halve the widths of the columns 8 through 14 and rows 7 through 12 by repeatedly right-clicking each of them and specifying the appropriate refinement factor (= 2). Your grid will look like the one in figure 4.2.
- Set the appropriate layer properties (confined or unconfined) under Grid → Layer property. Select 'User defined' for the Transmissivity and Leakance.
- Set the boundary conditions. The granite hills will be represented by inactive grid cells. Use the Polygon feature of the PMWIN editor to accurately delineate the hills. You can copy the inactive cells to the second model layer by activating the 'Layer copy' button on the toolbar. The boundary condtition

Table 4.1: Aquifer system parameters for river valley exercise

Parameter	magnitude	units
Aquifer 1 (phreatic)		
Horizontal hydraulic conductivity (k_h)	5	m/day
Vertical hydraulic conductivity (k_v)	0.5	m/day
Porosity (n)	0.2	
Silt layer (confining unit)		
Horizontal hydraulic conductivity (k_h)	0.5	m/day
Vertical hydraulic conductivity (k_v)	0.05	m/day
Porosity (n)	0.25	
Aquifer 2 (confined)		
Horizontal hydraulic conductivity (k_h)	2	m/day
Vertical hydraulic conductivity (k_v)	1	m/day
Porosity (n)	0.25	
River		
Stage (upstream/downstream)	19.4/17	m
Bottom elevation (upstream/downstream)	17.4/15	m
Width	100	m
River bed hydr. conductivity	2	m/day
River bed thickness	1	m

at the upstream and downstream valley boundaries will be fixed heads. The values of the fixed heads (which you need to enter later under Grid → Initial and prescribed heads) are in the file fixed_heads.dat.

- Specify the layer top elevations by going to Grid → Top of Layers and loading the files top1.dat and top3.dat. Specify the layer bottom elevations by going to Grid → Bottom of layers. Do *not* accept the option for determining the bottom elevations from the top elevations of the underlying layers. Set the bottom elevation of layer 1 to the elevation of the top of the silt layer, which is stored in the file top2.dat. Set the bottom of layer 2 to a constant value of 0 m.

- Select Parameters → Time and set the time units to days and specify that this will be a steady-state simulation.

- Specify the transmissivity, vertical leakance, horizontal hydraulic conductivity and effective porosity. You can calculate these number with the data from table 4.1 and the thicknesses of the layers (look these up in the model). Take care of the following:

  1. Note that the thickness of the confined aquifer is not constant. Therefore, the transmissivity of this layer varies. You can calculate it au-

tomatically by using a trick. First, set the hydraulic conductivity to 2 m/day in each cell (use the Reset matrix function). Then load the file top3.dat which contains the aquifer top elevation. Since the bottom elevation is 0 everywhere, the numbers in this file equal the thickness of the aquifer. Before pressing the Ok button to load the file, select the option multiply. The numbers currently in the grid will be multiplied with the numbers in the file.

2. The vertical leakance is only defined for the first model layer. It is not defined for the bottom model layer (you can check this in the MODFLOW input file later). It is dependent on the thicknesses of all 3 hydrostratigraphic units. It is a bit harder to calculate than the transmissivity so the file has been prepared for you. It is called vcont1.dat. Check if you understand how these values have been calculated. Do this by calculating the value for a particular cell by hand.

- Add the river. MODFLOW requires that the river data (i.e. stage, bottom elevation, and riverbed conductance) are specified for each grid cell that the river intersects. Go to Models → MODFLOW → Flow packages → River. Use the Polyline input method to accurately trace the river. Vertices are added by left-clicking; defining the polyline is cancelled by right-cliking. To complete the polyline, left-click the last vertex you specified again. Then right-click the leftmost (upstream) vertex and enter the data for this river node (the values are in table 4.1). Do the same for the rightmost (downstream) vertex. You do not need to enter the values for the other vertices: PMWIN interpolates between the two outer vertices, which makes life a bit easier.

After entering all the data make sure that you did not forget anything. Checking the input is an important step in groundwater modelling, especially if you are running models that take a couple of hours to finish. Then run MODFLOW.

Graphical user interfaces like PMWIN are great in that they tremendously facilitate data entry. The drawback is that a lot of the ‘action’ takes place behind the scenes and remains hidden from the user. For example, PMWIN generates all the input files that MODFLOW needs. Usually, there is no need to edit these input files yourself but as an academic you are naturally curious of how they look. Moreover, sometimes a graphical user interface does not support all the options of a code and you may need to modify the input files directly. Therefore, before proceeding, go to your working directory that contains all the files of this model. Look up the file that has the extension ‘.nam’. It contains a list of the filenames that are used by MODFLOW. Look up these files and try to find out what their purpose is and what data they contain.

Then plot the hydraulic head distribution using Tools → 2D Visualization. Also check the water balance. You can use either the water budget option of PMWIN or open the file ‘output.dat’ in a text editor. How much water enters the model domain

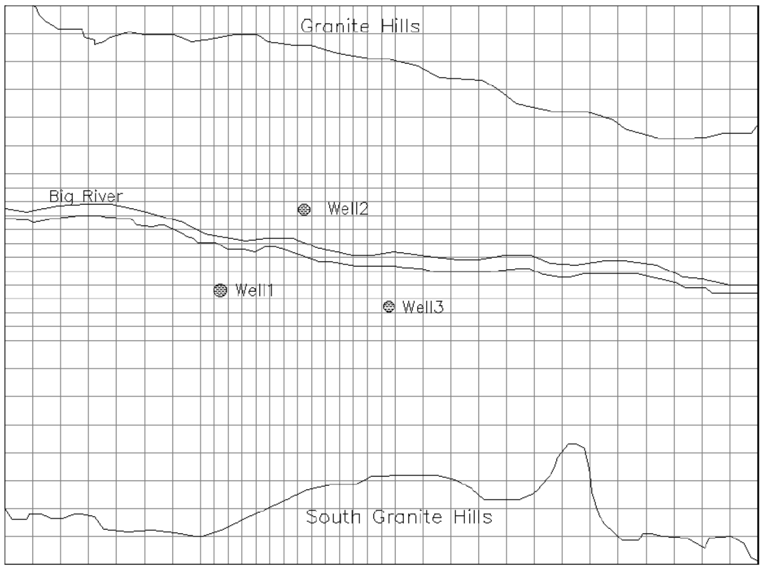


Figure 4.2: Model grid

through the upstream valley boundary? And how much water leaves through the downstream boundary? The river is both a source and a sink of water to the aquifer. But on the whole, is this a losing or a gaining river? Note that the water balance error is basically zero. Write down the numbers as we will compare them to the model with groundwater abstraction later.

Then run the model again but with the 3 wells in place in the confined aquifer. Each withdraws water at a rate of 500 m^3/day from layer 3, which is entered under Models → MODFLOW → Flow packages → Well.

Again, observe the head distribution. This time, also use PMPATH to draw flow lines. Then check the water budget again. What percentage of the pumped water derives from groundwater and how much is infiltrated river water? By how much has the river discharge decreased at the downstream model boundary compared to the natural situation?

Then change the model from a quasi-3D to a full 3D model. The easiest way to do this is to:

- go to Grid → Mesh size and subdivide the second model layer into 2 layers. Do this by pressing PgDn to go to the second model layer. Then you can right-click on any cell and type 2 in the Layer refinement field.

- set the appropriate layer properties under Grid → Layer property. Select 'Calculated' for the Transmissivity and Leakance. This means that the values of these parameters will be calculated by MODFLOW using the hydraulic conductivities and layer top and bottom elevations.

- set the boundary conditions. The values for the two aquifers remain the same as before. The inactive cells of the second model layer (the silt layer) are the same as for the aquifers, but the boundaries on the upstream and downstream valley boundaries are no-flow boundaries (we assume that flow is purely vertical in the silt layer so there is no flow over the boundary).

- set the layer top and bottom elevations using the values from the files you used earlier.

- set the horizontal and vertical hydraulic conductivity and effective porosity to the values from table 4.1.

If you run the model you will get basically the same results as with the quasi-3D model. Check this by comparing the head distribution and the water balance.

As a final exercise, do a sensitivity analysis for the hydraulic conductivity of the river bed. As you can imagine, this parameter is extremely difficult to quantify and it will also probably be quite variable along the streambed. We have used a value of 1 m/day but why could they not range between 0.5 to 2 m/day? Re-run the model using these extremes and investigate the effect on the hydraulic head distribution and the water budget. Don't let these result make you lose faith in models but consider them a lesson in critical thinking.

Part III: Creating finite element models in MicroFEM

Chapter 5

Well in a semi-confined aquifer

This exercise serves to illustrate the basic capabilities of MicroFEM. A well abstracts groundwater at a constant rate of 50 m^3/hour from a confined aquifer. Under natural conditions uniform flow occurs in a WSW direction, parallel to the long sides of a rectangular model. We will model the head distribution in the aquifer and calculate a water balance.

1. Create a network using the FemGrid generator: Choose Files → New grid and in the window that appears select 'Create new grid' and press OK. In the subsequent window, enter a descriptive model name. This is required before you can continue. By pressing OK you enter the grid editor.

   In the upper righthand corner you see 3 tabsheets in which you enter the required information to create the grid. You will have 5 fixed nodes: four at the corners of the grid and 1 for the well. Only 3 fixed nodes are default, however. To change this, type 5 in the edit field and click the button 'Change number to'. Now there will be 5 rows for which you can enter the x- and y-coordinates. The coordinates are (1000, 0), (13000, 3000), (12000, 7000) and (0, 4000) for the model boundary corner nodes and (4500, 3000) for the well. Define the segments and regions as explained during the lecture. Set the distance between nodes to 400 m on the model edges and to 250 m in the model interior.

   Then press the button 'Fixed nodes' in the toolbar at the bottom of your screen. The fixed nodes are created and displayed on your screen. The caption of the button has changed to 'Nodes on segments'. Press it again and observe what happens. Press it again and again and each time make sure you understand what is going on. Your network will have 866 nodes (figure 5.1). Then you can either save the network (recommended) or choose the number of aquifers (which is 1) and start entering the hydraulic parameters.

2. Find the dimensions of the rectangular model domain, i.e. the exact model width and length. Do this by placing the cursor in one of the 4 corner nodes and assigning the variable *r* to all nodes for a parameter (e.g. h0) in the input

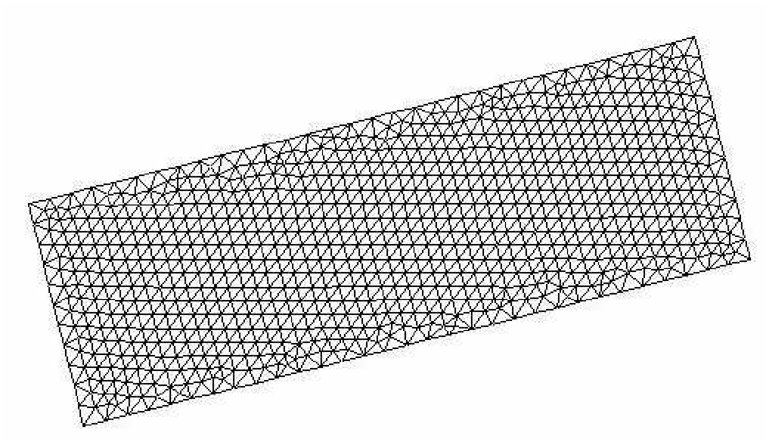


Figure 5.1: Rectangular model area with short east- and western and long northern and southern boundaries.

mode. Look up the meaning of this parameter in the help file! What are the model dimensions?

3. We start with a confined aquifer so the vertical resistance of the first model layer should be set to 0 (which is interpreted as infinite by MicroFEM). The transmissivity is $T = kD = 800$ m^2/day. These parameters are entered in the input mode.

4. Groundwater flow is in a WSW direction parallel to the long sides of the model. The head at the eastern boundary is 20 m and the hydraulic gradient is 0.001. Assign head values to all model nodes by marking the east model boundary in the walking mode and using a formula that calculates the head as a function of the head on the eastern boundary, the gradient and the variable *d* (which represents the distance to the nearest marked node).

5. Make a contour map in the drawing mode to check that the heads were entered correctly. Double-check by going to the western boundary: the head should be 7.631 m here.

6. Go to the fixed node that contains the well (coordinates 4500, 3000). Note that it has the label 'fixed node 5' so it is easily found by using the 'Jump to node' function in the walking mode. Press F12 or the 'Jump to node' button, select 'Next label' and choose 'fixed node 5' from the pull-down list. Make sure that you ended up in the right node (check the coordinates as well as the label in the lower-right corner of the screen) and then assign the well discharge to this node in the input mode. Well discharge has a positive value in MicroFEM, contrary to MODFLOW! Also change the name (= the value of the parameter label1) of this node from 'fixed node 5' to 'Well'.

7. Run the model by going to Calculate → ‘Go calculate’. Why is there no convergence?
8. Restore the natural hydraulic head situation by repeating step 4. Then fix the heads on the eastern and western model boundaries. Mark the nodes on these boundaries and make sure that the head of the first aquifer is selected in the parameter list. Then use the ‘Toggle heads for marked nodes’ function to fix the heads: The word ‘fixed’ will appear in the parameter list for the marked nodes.
9. Run the model again by going to Calculate → ‘Go calculate’. This time, a solution is found. Why?
10. Draw the hydraulic head contours in the drawing mode. You can overlay a set of flow vectors to visualize the flow. Explore the options of the drawing mode by clicking around a bit.
11. Go to the node that contains the well to draw the flowlines to delineate the capture zone of the well. Specify a number of 12 flowlines, a timestep of 100 years, layer thickness of 40 m and a porosity of 30 %. What is the width of the capture zone? Why are the isochrons curved?
12. Set up a water balance for the entire model. You can find the water balance options in the walking mode. What is the inflow through the eastern model boundary? And what is the outflow over the western boundary?
13. We will now change the model from a confined aquifer with a well to a semi-confined aquifer without a well. Moreover, we set the transmissivity downstream of the well to 300 m^2/day. The *groundwater table* (h0) is set to the original *hydraulic heads* under natural conditions. Make sure to make the following changes:
    - Assign a hydraulic resistance (= parameter c1) of 2500 days to the confining unit.
    - Set the groundwater table by marking the eastern boundary and entering the formula (as in step 4 but now for h0 instead of h1): 20-0.001*d.
    - Change the transmissivity downstream of the well from 800 m^2/day to 300 m^2/day. Do this by marking the appropriate nodes and assigning the value of 300 m^2/day to the marked nodes only in the input mode.
    - Remove the well from the model (i.e. set the discharge to 0).
14. Calculate the heads. Also make a contour plot of h1-h0 through the following steps:
    - Go to Project → Project Manager to add an extra variable for each node that can be used to hold intermediate results. A window will appear with a list of all MicroFEM files for this model.

- Click the button with the bright-green + sign, select Xtra worksheet, check ‘New’ in the ‘Create data from’ field (otherwise you will be asked for a file to read the data from) and click OK. You can choose to have as many registers (= new input fields) as you like, but 1 will be enough for our purposes.
- Note that the project manager is also the place to add special types of boundaries. Close the project manager and note that an new tab has been created that is called Xtra. It contains an edit field for a variable x1. This variable can contain any value we assign to it without affecting the calculations. Since we are interested in the head difference across the semi-confining unit we assign the result of the formula ‘h1-h0’ to all nodes.

The positive values of the head difference indicate that there is upward seepage in the whole model domain. Check the water balance. How much is the upward seepage through the semi-confining unit?

15. Then activate the well again. Calculate the model and check the water balance. Indicate the zones of upward seepage and infiltration by drawing the difference between h0 and h1. Calculate the volumetric rate of upward seepage/infiltration (in m^3/day) for each node (in the Xtra worksheet). Check your calculation by comparing it to the water balance.
16. What proportion of the well discharge infiltrates through the semi-confining unit and what proportion derives from regional groundwater flow?

Chapter 6

Infiltration canal

This problem serves to demonstrate the supremacy of finite element models over finite difference models when it comes to grid design. A shallow infiltration canal loses water to two fully penetrating drains. The drains are 120 m apart, with water levels at 40 and 35 m (left and right side, respectively) above an impervious base. The water level of the 5 m deep and 25 m wide infiltration canal is 55 m (see figure 6.1). The aquifer is homogeneous and isotropic, with hydraulic conductivity $k = 25$ m/day.

We will simulate this problem as a profile model. A hollow groundwater table will develop between the infiltration canal and the drains. The grid must be deformed in order to represent the saturated part of the model domain. Remember that at the groundwater table the pressure of the water $P = P_{abs} - P_{atm} = 0$ so that the hydraulic head $h = z + \frac{P}{\rho g} = z$, where z is the so-called elevation head. In other words, the elevation of a water table node above a reference level (in this case the impermeable base) needs to be equal to the calculated head. The mesh needs to be adjusted after each calculation so that the top model boundary coincides with the groundwater table.

1. In order to save you some time, the configuration of the network has already been prepared and can be found on Blackboard. The fixed nodes, triangles and quadrangles that make up the mesh are stored in the file 'profile model.fen'. Open this file in MicroFEM by going to Files $\rightarrow$ New Grid and pressing OK. A file selector window appears. Open the fen file and check how the FemMesh network has been defined.

2. Create the network and go to MicroFEM. Select 1 as the number of layers. A profile model is implemented in MicroFEM by assigning an infinite resistance to the upper confining unit (i.e., c1 = 0) and entering the hydraulic conductivity (k-value) instead of the transmissivity.

3. Then set up the model: assign fixed heads to the nodes that represent the canal and the drains and set the appropriate values for h1. Enter the transmissivity (= hydraulic conductivty). As a preparation for subsequent steps

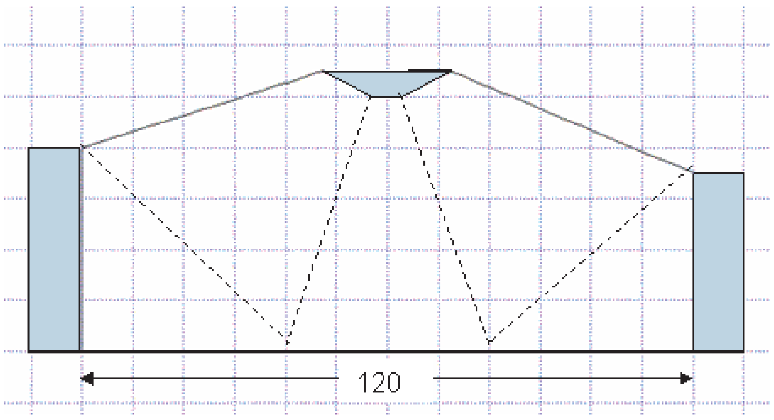


Figure 6.1: Configuration of the flow problem of the infiltration canal

give the nodes that sit between the canal and the drains the label ‘water table’.

4. You are now ready to calculate the model for the first time. New heads will be calculated, which provide a first estimate of the shape of the groundwater table. We can use these to relocate the nodes at the top of the model domain. Take the following steps and follow these carefully (otherwise MicroFEM could crash):

   - Go to Export → Special ASCII files and select CSV-file. For each node all the model parameters, together with the x- and y-coordinates, will be exported.
   - The trick is now to change the y-coordinates of the ‘water table’ nodes and set these equal to the head we just calculated (parameter h1). You could in theory do this in Excel but it’s a pain. Python is much better suited for this purpose. Remember also that this is only a small file but the output from a real-world model may not even fit in Excels datasheet. A file called ’read_csv_file.py’ is available on Blackboard. Download it to the same directory where you saved the csv file. The comments in the file explain in detail what it does. Inspect the file and make sure that you understand what is going on. Then run it.
   - Return to MicroFEM and select Import → Special ASCII files. Select CSV-file and open the file you just saved in Excel. The new coordinates will be read and the nodes on the upper model boundary have shifted downwards. The grid has been deformed and now is a better approximation of the shape of the saturated part of the model domain.
   - With the nodes shifted, the shape of the elements is not necessarily ideal. Therefore we will let MicroFEM optimize the shape of the net-

work. In the Walking mode, mark all nodes. Then go to the Alter grid mode and press F10 (‘Relocate marked nodes’). You can see the network being optimized. It doensn’t hurt to do this 2 more times to get the most optimal grid.

- Now calculate the heads again and repeat the steps above until the shape of the grid (or the position of the groundwater table) no longer changes. The flow problem is then solved.

In this example, the shape of the grid depends on the calculated heads. The flexibility of finite element methods and the powerful capabilities of MicroFEM allow you to solve this problem. Solving the same type of problem in MODFLOW is not so easy!

5. Draw flowlines starting at *all* nodes of the infiltration canal bed. Hint: give these nodes the same label (e.g. ‘canal’).
6. Set up a water balance. How much water is lost to each drain?
7. If you have time left, you can play around with the hydraulic conductivity and see what effect it has. For example, change use a value of $k = 1$ m/day and note the effect on the head contours and the water balances. Or you could introduce anisotropy (e.g. $k_h = 25$ and $k_v = 1$ m/day).

Part IV: Linear algebra

Chapter 7

Systems of linear equations

Numerical approximations of the governing equations for groundwater flow lead to a set of algebraic equations that can be solved for the unknown heads by (i) iteration or (ii) direct solution techniques. With direct solutions, the system of equations is transformed into matrices and the unknown heads are solved using the techniques of linear algebra. Therefore, some knowledge these subjects is indispensable if you are working with numerical groundwater models.

This document is intended for self-study and will give you a basic comprehension of the relevant topics for the practicing hydrologist. Some of the exercises require the use of Python. Python provides support for matrix operations through the Numpy library. These are not described here but you are expected to try and look these up yourself in the on-line documentation for Numpy. A good starting point is the following website:
`http://docs.scipy.org/doc/numpy/reference/index.html`
But of course, when in doubt or when you get stuck, do not hesitate to ask your instructor!

7.1 Linear equations

A *linear equation* is an algebraic expression with one or more variable in which each term is a constant or the product of a constant and a variable. For example:

$$h(x) = Ax + B \tag{7.1}$$

where h is the dependent variable, x is the independent variable, A is the *slope* or gradient and B is the *intercept*, i.e. the value of h when $x = 0$.

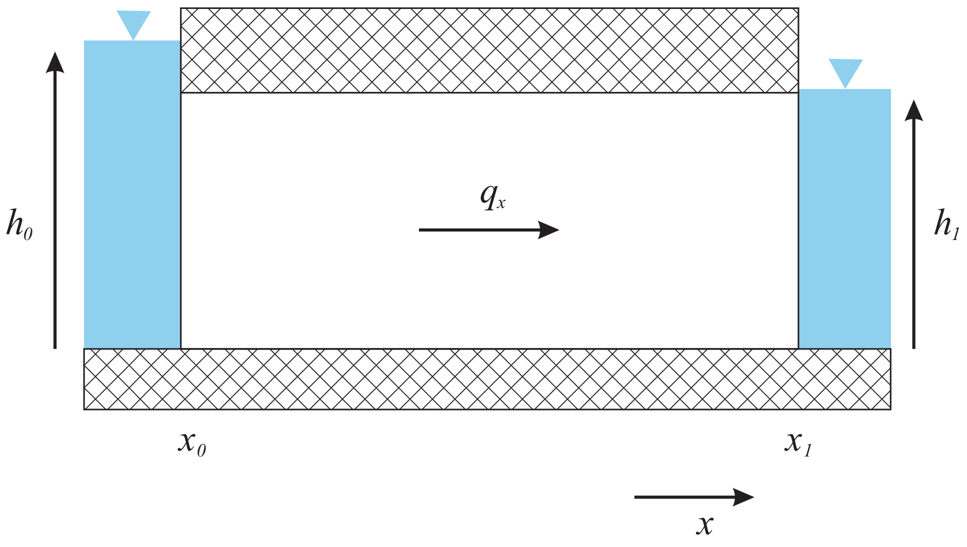


Figure 7.1: Schematic cross-section depicting a confined aquifer with one-dimensional horizontal flow.

7.2 Systems of linear equations

Two variables

Recall that Equation 7.1 is the general solution of a one-dimensional partial differential equation

$$\frac{\partial^2 h}{\partial x^2} = 0 \tag{7.2}$$

that describes the hydraulic head h in a confined aquifer between two canals as depicted in Figure 7.1. As a demonstration example of a *system of linear equations*, let's try to determine the integration constants A and B from boundary conditions. So instead of treating h and x as the dependent and independent variables, respectively, their values become fixed by the following boundary conditions:

$$h\Big|_{x=x_0} = h_0 \tag{7.3}$$

$$h\Big|_{x=x_1} = h_1 \tag{7.4}$$

Combining conditions 7.3 and 7.4 with Equation 7.1 and yields a system of two linear equations:

$$h_0 = Ax_0 + B \tag{7.5}$$
$$h_1 = Ax_1 + B \tag{7.6}$$

in which A and B are the unknowns and x_0, x_1, h_0 and h_1 are known constants. If we wish to find the numerical values of A and B we must find *the solution* to this system of linear equations. This means finding the values of A and B so that both Equation 7.5 and 7.6 are satisfied.

Assume that $h_0 = 10$, $h_1 = 9$, $x_0 = 50$ and $x_1 = 150$[1]. Let B now be the dependent and A be the independent variable so that we have:

$$B = 10 - 50A \tag{7.7}$$

$$B = 9 - 150A \tag{7.8}$$

One way to find the solution is to plot Equations 7.7 and 7.8 in a graph and to look up the point where they intersect. This is shown in Figure 7.2. Obviously, this method can be inaccurate when the graph is drawn by hand and is not suitable for systems of equations with more than 2 variables.

Exercise: Find the graphical solution to Equations 7.5 and 7.6 for A and B assuming $x_0 = 0$ and the other parameters as listed above.

A better way would be to use the method of substitution. This involves reducing the number of unknowns by expressing A as a function of B using either one of the Equations 7.7 or 7.8 and substituting the result into the other. The equation obtained this way can be solved for B. The value of A is subsequently found by substituting the calculated value of B in either Equations 7.7 or 7.8 and solving for A.

Exercise: Find the solution of Equations 7.7 and 7.8 by the method of substitution. Verify that your answer is the same as in Figure 7.2. Note that the solution is written as $\{(A, B)\}$, i.e. a set containing an ordered pair.

For the system described by Equations 7.7 or 7.8 there is one solution and therefore it is called a *consistent* system. An *inconsistent* system on the other hand has no solution. For example, the linear system:

$$50A + B = 10 \tag{7.9}$$

$$50A + B = 9 \tag{7.10}$$

represents two parallel lines and therefore has no solution. This is denoted by a null or an empty set: $\emptyset$ or $\{\}$. Note that substitution would yield $10 = 9$, which is a contradiction from which it is clear that there can be no solution.

A system of two linear equations with two variables will have an infinite number of solutions when the two lines coincide. This occurs if the equations are a multiple of each other. For example:

$$50A + B = 10 \tag{7.11}$$

$$100A + 2B = 20 \tag{7.12}$$

[1]All having length units, for example m

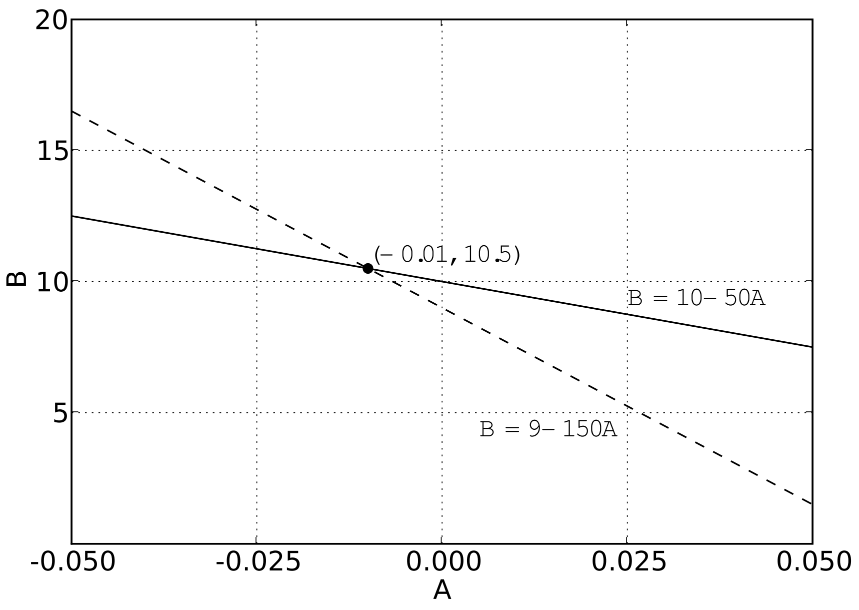


Figure 7.2: Graphical representation of Equations 7.7 and 7.8.

represent the same line in a plot of A versus B. This is called a *dependent* system and the solution is either of these two equations: $\{(A, B) \mid 50A + B = 10\}$.

Three (or more) variables

To develop a system of linear equations of three variables, suppose we would like to solve Equation 7.2 numerically. For this purpose, the domain between x_0 and x_1 is first discretized into n cells. The numerical approximation yields (central in space, cf. lecture notes):

$$\frac{\partial^2 h}{\partial x^2} \approx \frac{h_{c-1} - 2h_c + h_{c+1}}{(\Delta x)^2} = 0 \tag{7.13}$$

or:

$$2h_c = h_{c-1} + h_{c+1} \tag{7.14}$$

where the subscript c indicates the cell number ($1 \leq c \leq n$). Suppose that $n = 5$ and that the heads in the first and fifth cells are fixed heads with $h_1 = 10$ and $h_5 = 6$. The heads in the nodes h_2, h_3 and h_4 are then given by:

$$\begin{aligned} 2h_2 &= 10 + h_3 \\ 2h_3 &= h_2 + h_4 \\ 2h_4 &= h_3 + 6 \end{aligned}$$

that can be written as a system of linear equations with three variables:

$$\begin{array}{rcrcrcl} 2h_2 & - & h_3 & & & = & 10 \\ -h_2 & + & 2h_3 & - & h_4 & = & 0 \\ & - & h_3 & + & 2h_4 & = & 6 \end{array} \tag{7.15}$$

This system system of equations can be solved for h_2, h_3 and h_4 by an algorithm called *Gaussian elimination*. It uses *elementary row operations* to transform the linear equations into a form that is easily solved. Application of these operations does not change the solution of the system of linear equations. The operations are:

1. Equations can be interchanged
2. An equation can be multiplied with a non-zero constant
3. An equation can be multiplied by a constant and added to another equation

As an example, first reverse the first and second equation in the system defined by Equation 7.15 (Operation 1):

$$\begin{array}{rcrcrcl} -h_2 & + & 2h_3 & - & h_4 & = & 0 \\ 2h_2 & - & h_3 & & & = & 10 \\ & - & h_3 & + & 2h_4 & = & 6 \end{array} \tag{7.16}$$

Then add 2 times the first equation to the second equation (Operation 3):

$$\begin{array}{rcrcrcl} -h_2 & + & 2h_3 & - & h_4 & = & 0 \\ & & 3h_3 & - & 2h_4 & = & 10 \\ & - & h_3 & + & 2h_4 & = & 6 \end{array} \tag{7.17}$$

Finally, multiply the third equation by 3 (Operation 2) and add the second equation (Operation 3):

$$\begin{array}{rcrcrcl} -h_2 & + & 2h_3 & - & h_4 & = & 0 \\ & & 3h_3 & - & 2h_4 & = & 10 \\ & & & & 4h_4 & = & 28 \end{array} \tag{7.18}$$

This form of the system of equations is called *echelon form* (or *row echelon form*), meaning that that the first non-zero leading coefficient in an equation is to the right of the first leading coefficient in the equation above it. A system is in *reduced-echelon form* when the first coefficient in an equation is always one. To transform the system of equations into reduced-echelon form, equation 1 is multiplied by -1, equation 2 by $1/3$ and equation 3 by $1/4$:

$$\begin{array}{rcrcrcl} h_2 & - & 2h_3 & + & h_4 & = & 0 \\ & & h_3 & - & 2/3h_4 & = & 10/3 \\ & & & & h_4 & = & 7 \end{array} \tag{7.19}$$

which immediately yields $h_4 = 7$. The values of h_2 and h_3 are subsequently found by substitution.

Exercise: Find the values of h_2 and h_3 by back-substitution.

Exercise: Find the values of A and B in Equations 7.7 and 7.8 by Gaussian elimination.

Chapter 8

Matrices

8.1 Definition

A *matrix* is a rectangular array of numbers. It consists of m rows and n columns, which are written between square or round brackets. For example, the matrix A:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \tag{8.1}$$

This matrix has 2 rows and 3 columns. Its *order* is therefore 2×3. Each number in the matrix is called an *element*. Elements can be real or complex numbers. Elements are denoted by $a_{r,c}$ whereby subscripts r and c refer to the row and column number, respectively. For example, $a_{2,3}$ refers to the number in the second row and third column, i.e. $a_{2,3} = 6$. Sometimes the comma between the row and column indexes is omitted but this may lead to ambiguity when $m \geq 10$ or $n \geq 10$.

Two matrices are equal if their order is the same and when all the corresponding elements in both matrices are the same. The *transpose* of a matrix is obtained when its rows and columns are interchanged. The transpose of A is denoted by A^T and is defined as $A^T = (a_{c,r})$, for $1 \leq r \leq m$ and $1 \leq c \leq n$. The order of A^T is $n \times m$. For example:

$$A^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \tag{8.2}$$

Exercise: Define A in Python and find A^T.

8.2 Special matrices

A *row vector* is a matrix that has only one row, i.e. $m = 1$. Similarly, a *column vector* has $n = 1$. A square matrix has $m = n$. A square matrix is a *symmetric* *matrix* when $A^T = A$. For a *skew-symmetric matrix* $A^T = -A$.

The elements $a_{r,c}$ of a square matrix A for which $r = c$ together form the *principal* or *main diagonal*. The sum of these elements is called the *trace*. If a square matrix A has $a_{r,c} = 0$ when $r \neq c$ then this matrix is called a *diagonal matrix*.

A *triangular matrix* is a square matrix that has only non-zero values on or on one side of the principal diagonal. An *upper triangular matrix* has all non-zero elements on the principal diagonal or above it. All elements below its principal diagonal are zero. A *lower triangular matrix* has all non-zero elements on or below the principal diagonal and only zero elements above it.

A diagonal matrix of which all elements $a_{r,c} = 1$ when $r = c$ is called an *unit* or *identity matrix,* which is denoted by I. For example, an identity matrix of size 3:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.3}$$

A matrix whose elements all equal zero is called a *zero matrix,* which, unlike an identity matrix, is not necessarily square. It is usually denoted by O or 0. Zero and identity matrices play the same role as the numbers 0 and 1 in ordinary algebra.

Exercise: Use the standard Numpy commands to create a diagonal matrix, a zero matrix and an identity matrix, all of order 3×3.

8.3 Operations

Addition

If two matrices A and B are of the same order, their individual elements can be added. Their sum is defined as $A + B = (a_{r,c} + b_{r,c})$, for $1 \leq r \leq m$ and $1 \leq c \leq n$. For example:

$$B = \begin{bmatrix} 7 & 9 & 11 \\ 8 & 10 & 12 \end{bmatrix} \tag{8.4}$$

$$A + B = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 9 & 11 \\ 8 & 10 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 12 & 16 \\ 10 & 14 & 18 \end{bmatrix} \tag{8.5}$$

Matrix addition is both commutative: $A + B = B + A$ and associative: $A + (B + C) = (A + B) + C$. For a zero matrix of the same order as A it holds that $A + O = O + A = A$.

Subtraction

Similarly, matrices A and B can be subtracted. The difference is defined as $A - B = (a_{r,c} - b_{r,c})$, for $1 \leq r \leq m$ and $1 \leq c \leq n$. So:

$$A - B = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} - \begin{bmatrix} 7 & 9 & 11 \\ 8 & 10 & 12 \end{bmatrix} = \begin{bmatrix} -6 & -6 & -6 \\ -6 & -6 & -6 \end{bmatrix} \tag{8.6}$$

Scalar multiplication

Multiplying each element of a matrix A with a *scalar* λ number gives the *scalar product* $\lambda A = A\lambda = (\lambda a_{r,c})$, for $1 \leq r \leq m$ and $1 \leq c \leq n$. For example:

$$2 \cdot A = 2 \cdot \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 10 \\ 4 & 8 & 12 \end{bmatrix} \tag{8.7}$$

Matrix multiplication

A matrix can also be multiplied with another matrix provided that the left matrix has the same number of rows as the number of columns of the right matrix. The result is called the *matrix product*. If A has order $m \times n$ and C has order $n \times p$ then the product

$$A \cdot C = D = \left(\sum_{i=1}^{n} a_{r,i} c_{i,c} \right) \tag{8.8}$$

for $1 \leq r \leq m$ and $1 \leq c \leq p$. The order of D is $m \times p$. For example:

$$C = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} \tag{8.9}$$

$$A \cdot C = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} =$$

$$\begin{bmatrix} 1 \cdot 7 + 3 \cdot 9 + 5 \cdot 11 & 1 \cdot 8 + 3 \cdot 10 + 5 \cdot 12 \\ 2 \cdot 7 + 4 \cdot 9 + 6 \cdot 11 & 2 \cdot 8 + 4 \cdot 10 + 6 \cdot 12 \end{bmatrix} = \begin{bmatrix} 89 & 98 \\ 116 & 128 \end{bmatrix} \tag{8.10}$$

Exercise: Define A and C in Python and calculate AC.

Matrix multiplication is not commutative, i.e. $AB \neq BA$, because the dimensions may not agree if the order is reversed. Just like for real numbers, matrix multiplication is both associative, i.e. $(AB)C = A(BC)$ and distributive, i.e. $(A+B)C = AC + BC$ and $C(A+B) = CA + CB$.

The result of a multiplication of a matrix A by a zero matrix O is $A \cdot O = O = 0$. Multiplying a matrix A with an identity matrix I gives $A \cdot I = A$. For square matrices it holds that $A \cdot I = I \cdot A = A$ and $A \cdot O = O \cdot A = O$.

Note that matrix division is undefined. So if $A \cdot C = D$ then there is no such thing as $C = D/A$. If A and D are give, C can be found, however, by multiplying D with the so-called *inverse* of A (A has to be square). Before discussing this concept, it is necessary to introduce another matrix property, the *determinant*.

8.4 Determinant

Every square matrix has a real number associated with it which is called the *determinant*. The determinant of matrix A is denoted by $\det A$ or sometimes $|A|$. It is not so easy to explain what a determinant is in words so let's just consider its mathematical definition and start with the determinant of a 2 $\times$ 2 matrix. It is defined as the product of the elements on the principal diagonal minus the product of the elements off the principal diagonal:

$$\det \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} = a_{1,1}a_{2,2} - a_{1,2}a_{2,1} \tag{8.11}$$

For square matrices of size 3 and more, a technique called *Laplace expansion* can be used to determine the determinant. It is based on the expansion of a matrix in *minors* and *cofactors*.

Minor

Suppose we have square matrix A. For every element $a_{r,c}$ in A there exists a quantity called *minor* which is the determinant of the matrix that results when the row r and column c are deleted from the original matrix. For example:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix} \tag{8.12}$$

then for $a_{1,1}$ the minor $M_{1,1}$ is:

$$M_{1,1} = \det \begin{bmatrix} 4 & -3 \\ 6 & -5 \end{bmatrix} = 4 \cdot -5 - -3 \cdot 6 = -2 \tag{8.13}$$

Cofactor

The *cofactor* is the *minor* $M_{r,c}$ multiplied by $(-1)^{r+c}$:

$$C_{r,c} = (-1)^{r+c} M_{r,c} \tag{8.14}$$

where $C_{r,c}$ is the cofactor. The factor $(-1)^{r+c}$ effectively determines the sign of the minor depending on its position within the matrix. So if $r + c$ is positive, the sign of the minor does not change. If $r+c$ is negative the sign of the minor reverses.

Exercise: Calculate the cofactors $C_{1,1}$, $C_{1,2}$ and $C_{1,3}$.

With the minor and cofactor defined, the determinant of a matrix can be found from:

$$\det A = \sum_{c=1}^{n} a_{r,c} C_{r,c} \tag{8.15}$$

for any row r. In words this formula reads: The determinant of a matrix is the sum of the product of the elements in a particular row multiplied by their cofactors. For example:

$$\det A = \det \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix} = a_{1,1}C_{1,1} + a_{1,2}C_{1,2} + a_{1,3}C_{1,3} \tag{8.16}$$

Inserting the numbers gives:

$$\det A = 1 \cdot -2 + 1 \cdot 1 + 1 \cdot 0 = -1 \tag{8.17}$$

Exercise: Calculate $\det A$ by selecting $r = 3$.

The same approach can be applied by selecting a particular column rather than a row. The determinant is the calculated from:

$$\det A = \sum_{r=1}^{m} a_{r,c} C_{r,c} \tag{8.18}$$

for any column c.

Exercise: Calculate $\det A$ by selecting $c = 2$.

For square matrices of size 4 and higher, the approach is the same. The work involved can be substantial: For a 4×4 matrix, four cofactors must be calculated, each requiring three 2×2 determinants to be calculated, i.e. a total of twelve 2×2 determinants. For any size n the number of 2×2 determinants to be calculated is $n!/2$. In practice this number can be less if the matrix contains zero elements: Because the cofactor is multiplied with the value of the element, there is no need to calculate the cofactor if it is multiplied by a zero anyway. Consider a matrix Z which contains 2 zero elements in column $c = 2$:

$$Z = \begin{bmatrix} 6 & 4 & 0 & 2 \\ 8 & 0 & 2 & 4 \\ 6 & 0 & 4 & 2 \\ 18 & 4 & 6 & 2 \end{bmatrix} \tag{8.19}$$

Expanding along column $c = 2$ will halve the number of calculations involved since there is no point in calculating the cofactors $C_{2,2}$ and $C_{3,2}$ because these are multiplied with $z_{2,2} = 0$ and $z_{2,3} = 0$.

Exercise: Use Python to calculate the determinant of Z.

Transposing a square matrix does not change it determinant, i.e. $\det A = \det A^T$. If any two rows or columns are interchanged the numerical value of the determinant remains the same but its sign changes. If a matrix is multiplied by a scalar, then the determinant is also multiplied by this scalar, i.e. $\det \lambda A = \lambda \det A$. For matrix multiplication it holds that when A and B are square and of the same order, $\det AB = \det A \det B$

8.5 Cramer's rule

One application of determinants is the solution of a system of linear equations using Cramer's rule. Suppose we have a system of 3 linear equations with 3 unknowns:

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & = & 9 \\ 2x_1 & + & 4x_2 & - & 3x_3 & = & 1 \\ 3x_1 & + & 6x_2 & - & 5x_3 & = & 0 \end{array} \tag{8.20}$$

which can be written in matrix form as (cf. Equation 8.8):

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9 \\ 1 \\ 0 \end{bmatrix} \tag{8.21}$$

or, with A, X and R representing the respective matrices:

$$AX = R \tag{8.22}$$

Cramer's rule states that the elements of X can be found from:

$$x_r = \frac{\det A_c}{\det A} \tag{8.23}$$

for $1 \leq r \leq m$ and $c = r$. A_c is the matrix that is obtained when column c in A is replaced by column vector R. For example, to find x_1, the first step is to calculate $\det A_1$, which is:

$$\det A_1 = \det \begin{bmatrix} 9 & 1 & 1 \\ 1 & 4 & -3 \\ 0 & 6 & -5 \end{bmatrix} = -7 \tag{8.24}$$

The determinant of A was calculated earlier (Equation 8.17). Inserting these values into Equation 8.23 gives:

$$x_1 = \frac{-7}{-1} = 7 \tag{8.25}$$

Exercise: Apply Cramer's rule to find x_2 and x_3.

The determinant of a matrix can be zero if an entire row is zero or if a row (or column) is a multiple of another row (or column). This also includes rows (or columns) which are the same (i.e. the multiplier is 1). If the determinant of a matrix is zero, it is called a *singular* matrix. Since $\det A$ appears in the denominator of Equation 8.23, Cramer's rule can not be applied when a matrix is singular. Several conditions may apply depending on the values of the determinants of A and R. These are summarized in table 8.1.

A hydrological application of Cramer's rule is in the derivation of the finite element approximation of the groundwater equation.

8.6 Inverse of a matrix

The inverse of a square matrix A is denoted by A^{-1} and is defined by $AA^{-1} = I$. The inverse can be used to solve a system of linear equations. The solution of X in Equation 8.22 is:

$$X = A^{-1}R \tag{8.26}$$

For a square matrix A the inverse can be found from:

$$A^{-1} = \frac{C^T}{\det A} \tag{8.27}$$

Table 8.1: Possible solutions for systems of linear equations depending on the values of $\det A$ and $\det R$.

	$\det R \neq 0$	$\det R = 0$
$\det A \neq 0$	unique solution	$X = 0$ (trivial solution)
$\det A = 0$	infinite number of solutions	infinite number of solutions if $\det A_c = 0$, $1 \leq c \leq m$

in which C is a matrix of cofactors of A and its transpose C^T is the so-called *adjugate* of A. From Equation 8.27 it follows that the inverse can only exist if $\det A \neq 0$. If this is the case then A is called a *non-singular* matrix (as opposed to a *singular* matrix).

For matrix A in Equation 8.12, C is:

$$C = \begin{bmatrix} -2 & 1 & 0 \\ 11 & -8 & -3 \\ -7 & 5 & 2 \end{bmatrix} \tag{8.28}$$

so:

$$A^{-1} = \frac{1}{\det A} C^T = \frac{1}{-1} \begin{bmatrix} -2 & 11 & -7 \\ 1 & -8 & 5 \\ 0 & -3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -11 & 7 \\ -1 & 8 & -5 \\ 0 & 3 & -2 \end{bmatrix} \tag{8.29}$$

Exercise: Insert A^{-1} and R (Equation 8.21) into Equation 8.26 to find X.

Exercise: Express the system of linear equations defined by Equation 7.15 in matrix form, i.e., $Ah = R$. Find h by calculating A^{-1} and applying Equation 8.26.

Obviously, calculating the inverse of a matrix by hand can be a lot of work and mistakes are easily made. Computers are much better at these sorts of calculations than humans.

Exercise: Use Python to calculate A^{-1} and $A^{-1}R = X$.

References

Chiang, W. H., 2005. 3D-Groundwater modeling with PMWIN. Springer, Berlin Heidelberg, Germany.

Wang, H. F., Anderson, M. P. 1982. Introduction to Groundwater Modeling: Finite Difference and Finite Element Methods. W. H. Freeman and Co, Gordonsville, Virginia, USA.