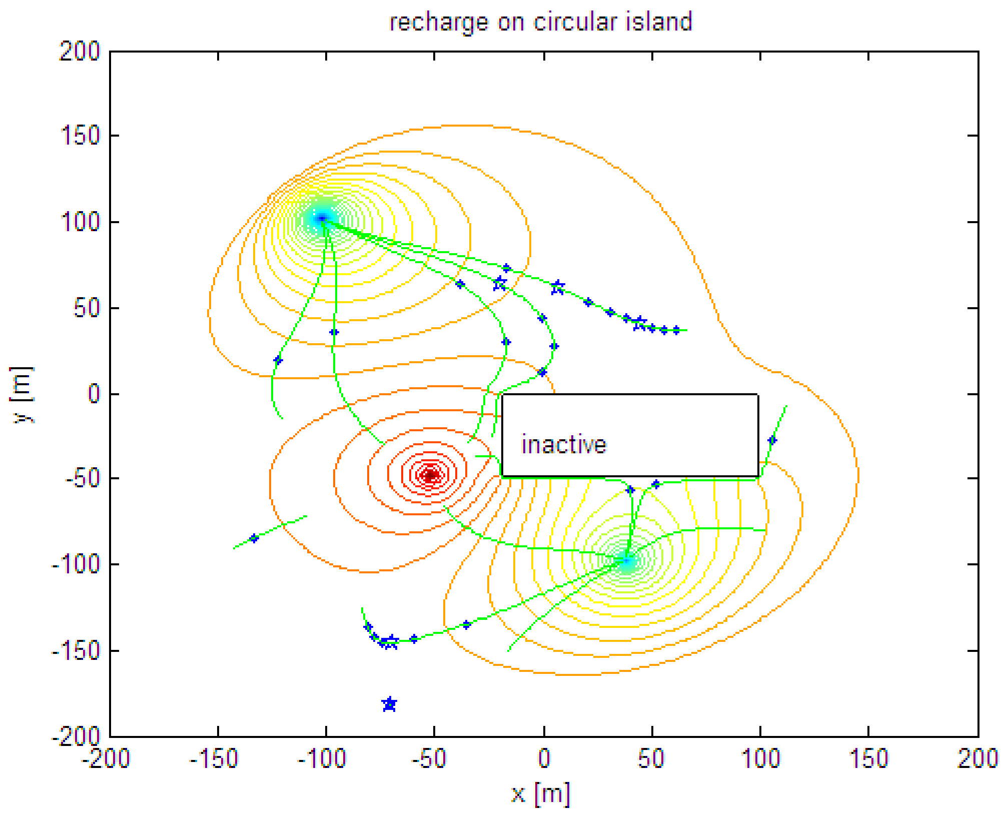
Syllabus CT 5440 (Geohydrology 2)

Your own Finite Difference Groundwater Model in Matlab (CT5440 exercises)



Several wells in circular island with wells an inactive part and some tracked particles

Prof. dr.ir. T.N.Olsthoorn

TUDelft, Water Resources Section

March 15, 2009

Table of contents

1 Abstract

This document explains the theory behind numerical groundwater modeling and how to make finite difference groundwater models in Matlab (The Mathworks ®). It is an exercise in the course CT5440, Geohydology 2. It aims in to provide a relevant insight in numerical groundwater modeling and focuses on finite difference models. The structure is general and largely also valid for other numerical model types such as finite element models and surface water models.

The models will be built by the student in Matlab. There will be a combined flat and radial symmetric model, one for steady-state and one for transient flow. The examples serve to demonstrate what may be done with them and also to show their accuracy with some pitfalls and tricks.

The models are small Matlab functions, elegant yet powerful. They should provide a thorough insight and practice in numerical modeling in general. With these tools you will be able to do quite sophisticated modeling. And yet, a 3D model is not given. To make one is easy and straightforward and left to the student. 3D models do not much good to this course and your experience, because much more time would then go into more complicated data handling and visualization, especially with transient computations (yielding 4D arrays). If you really have to do detailed regional transient 3D modeling with all ins and outs, I suggest using a regular model that is coupled to a GIS, so you can make use of maps and other database information.

Even though you could do all of your modeling in Matlab (even regional 3D multilayer transient modeling), a regular GIS based approach may be advisable using models that have been widely applied and intensively tested in practice.

Yet, knowing how to use these handy Matlab models gives you a powerful tool to deal with many practical groundwater problems in a very short time. It also comes in handy if you may ever need to check a large model.

The syllabus treats the theory without unnecessary details, shows how it works in Matlab and also shows how to get started with Matlab. You may copy parts of Matlab code directly from this word file whenever necessary, or rather make everything yourself.

2 Numerical groundwater modeling

We will first give a general description of groundwater modeling and then derive an actual model, which will finely be converted into a finite difference model by choosing the network and the way the conductances between model cells are computed. The general overview is true for all kinds of numerical models. We will follow the general approach as long as possible as that provides the best insight and the least clutter.

2.1 General overview numerical models

Numerical models, whether Finite Element or Finite Difference Models divide the area to be modeled into elements or cells with given properties and nodes (or cells) where the head will be computed (Figure 1). In the Finite Element Method these cells may be of arbitrary shape, while the shape in the Finite Difference Method is more limited, for instance regular hexagons or rectangular.

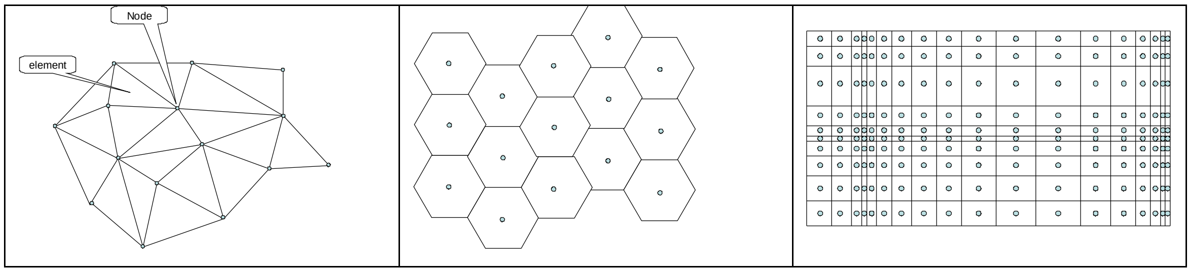


Figure 1: Difference model meshes (grid). The left figure shows a finite element triangular network with the nodes at the element corners. The center figure shows a hexagonal finite difference network with nodes in the center of hexagons. The ight figure shows a rectangular finite difference network with nodes in the center of the rectangles (called cells). A rectangular finite difference network with nodes at the cell corners is also possible, but not shown here. Area properties will be give for elements/cells, heads and flows in and out of the model will be computed for the nodes.

The locations where the heads are going to be computed i.e. the nodes may be the corners of the cells/elements or their center. We will make a finite difference model that subdivides a rectangular area into *Ny* rows and *Nx* columns, where the columns and rows may be of any size. We will compute the head in the center of these rectangles (see right-hand picture of Figure 1). This method is straightforward, easy to understand and easy to implement and successful, because the same approach is used by the world's most famous and most use groundwater model, i.e. MODFLOW of the United States Geological Survey (USGS) (McDonald & Harbaugh, 1988).

After any of the possible derivations for the model equations, either in the finite element or finite difference method (abbreviated to FEM or FDM), the final result comes down to a system of equations, each of which is a water balance for a node of the model. This system of equations represents all nodal water balances, which must be simultaneously fulfilled. This is achieved by solving the system, i.e. computing the heads in the nodes, such that all nodal balances are matched simultaneously.

The FDM directly writes out the water balance for the nodes, while the FEM takes a more general approach by requiring the governing partial differential equation (which is a water balance on infinitesimal scale) to be optimally fulfilled within all of the elements. The cost of the FEM is more complication in deriving the equations and setting up the model, but the bonus is definitely more flexibility in element shapes.

2.2 Deriving and assembling a numerical model

In the end, any model yields a set of water balances, one for each node. This is true for the FEM, the FDM as well as for any surface water model, in which the elements are replaced by links. So the number of equations, as well as the number of unknowns, equals the number of nodes. A finite difference model of 1000 rows and 1000 columns thus has a million equations with a million unknowns. This is what is required to compute a one-layer model of 1000 by 1000 m on 1 m resolution.

Figure 2 shows some of the nodes of an arbitrary model. For one node, number *i*, the adjacent nodes are shown to which it is directly connected through intermediate elements (FEM), intermediate cells (FDM) or connecting links (surface water model). The only difference between these types of model is the way in which the connections are computed. So most of the discussion about modeling and model construction can be done without bothering about these specific details. This keeps the discussion general and prevents us from getting lost in the details.

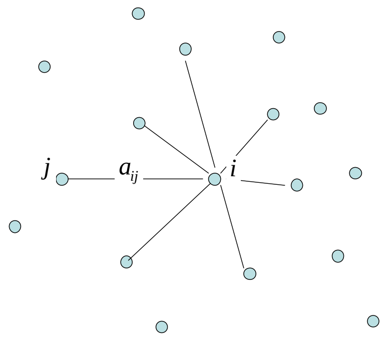


Figure 2: A model node with its surrounding connected neighbors

Just as general is, that the flow Q_{ij} from node *i* in the direction of adjacent node *j* with (still unknown) heads ϕ_i and ϕ_j respectively, is described by

$$Q_{ij} = C_{ij}\left(\phi_i - \phi_j\right) = \frac{1}{R_{ij}}\left(\phi_i - \phi_j\right) \tag{1}$$

where C_{ij} [(m^3/d)/m] is called the conductance and its reciprocal R_{ij} is the resistance [m/(m^3/d)]. This is true even for a surface-water model, be it that in that case C_{ij} will depend on the flow itself and therefore must be computed iteratively.

The physical meaning of the conductance is obvious; it is the flow of water from node *i* to node *j* in case the head difference $\phi_i - \phi_j$ equals 1.

The water balance of an arbitrary node *i* in the numerical model is described by the following equation

$$\sum_{j=1, j\neq i}^{j=N} Q_{ij} = Q_i \tag{2}$$

whose left-hand side represents the flow from node *i* through the model towards its connected neighbors, and the right-hand side equals the inflow “from the outside world” through node *i*. This is indeed a nodal water balance for a steady-state model. We will deal with transient models later. The nodal inflow Q_i (nodal outflows or extractions are just negative values of Q_i), is the total of all present sources of water from the outside world into this node (negative if extractions). It combines recharge, injections wells, leakage, drainage and so on.

Using the conductances, the nodal water balance becomes:

$$\sum_{j=1, j\neq i}^{j=N} C_{ij}\left(\phi_i - \phi_j\right) = Q_i \qquad \textbf{(3)}$$

Notice that *i* and *j* run over all the nodes of the model. This means that in this expression node *i* may be connected with all other nodes of the model. Of course it is only connected to its direct neighbors. Therefore, most of the conductances C_{ij} are zero. In case a node has *n* connecting neighbors, only $n+1$ of these coefficients are non-zero for each node. Therefore, of a model of N_y rows by N_x columns, heaving N_y*N_x nodes, each line only has (n+1)/(N_y*N_x) nonzero coefficients. For *n*=4 as is the case of a 2D one-layer model, and N_y= N_x=1000, only 5/100000=0.0005% of the coefficients are nonzero. The matrix containing the coefficients is therefore extremely sparse. We will make use of this when storing the system matrix and solving the model.

Writing out the above balance equation yields

$$-C_{i1}\phi_1 - C_{i2}\phi_2 - \ldots + \left(\sum_{j=1, j\neq i}^{j=N} C_{ij}\right)\phi_{ii} \ldots - C_{i,N-1}\phi_{N-1} - C_{iN}\phi_N = Q_i \qquad \textbf{(4)}$$

Using a_{ii} as general matrix element or matrix coefficient instead of the specific conductance, then (4) can be written as a general linear equation

$$a_{i1}\phi_1 + a_{i2}\phi_2 + \ldots + a_{ii}\phi_{ii} \ldots + a_{i,N-1}\phi_{N-1} + a_{iN}\phi_N = Q_i \qquad \textbf{(5)}$$

in matrix form

$$\sum_{j=1}^{N} a_{ij}\phi_j = Q_j \qquad \textbf{(6)}$$

where

$$a_{ij, j\neq i} = -C_{ij} \qquad \textbf{(7)}$$

$$a_{ii} = \sum_{j=1, j\neq i}^{j=N} C_{ij} = -\sum_{j=1, j\neq i}^{j=N} a_{ij} \qquad \textbf{(8)}$$

so that the sum taken over all matrix elements in a row (node) equals zero

$$\sum_{j=1}^{j=N} a_{ij} = 0 \qquad \textbf{(9)}$$

The physical meaning of diagonal matrix element a_{ii} is the amount of water flowing from node *i* to all its adjacent nodes if the head in node *i* is exactly 1 higher than that of its neighbors.

The equation above is equivalent to the matrix equation

$$\mathbf{A} * \Phi = \mathbf{Q} \tag{10}$$

With **A** the square system matrix holding the entries (matrix elements) a_{ij} as defined. Both *i* and *j* may take values from 1 to *Ny*Nx*. Therefore, the size of **A** is *Ny*Nx* rows by *Ny*Nx* columns, which may be really huge. Φ is the column vector of still unknown heads (length *Ny*Nx*) and **Q** the column vector of nodal inflows (length *Ny*Nx)*.

To fill the system matrix, all we have to do is compute the conductances between all connected nodes given by their indices *i* and *j* and put their negative value into the matrix at location *i,j*. When done, the coefficients for the diagonal, a_{ii} are computed by summing the other coefficients in the line of the matrix representing this node (see eq (7))

The model is based on the water balance of its nodes. The flow between the nodes is determined by the properties of adjacent model cells as well and their head difference. This intercellular flow is determined by Darcy's law using the heads of adjacent cells and the conductance between them. We will first consider the model's boundary conditions.

2.3 Boundary conditions

Boundary conditions connect (constrain) the model (or, for that matter, the partial differential equation) to the outside world. We already met one type of boundary condition, namely the given inflow of the nodes. The other type has to do with fixation of heads. We treat this in a general way, i.e. by writing out how fixed heads in the outside world connect to nodes of the model through a resistance, or rather a conductance (inverse resistance). Heads that are directly fixed to nodes then become a limiting case in which the resistance approaches zero or, reversely, where the conductance approaches infinity.

2.3.1 General head boundaries

Consider flow $Q_{ex,i}$ from the external world with fixed head h_i the model node *i* having an unknown head ϕ_i. This flow through conductance C_i equals

$$Q_{ex,i} = C_i \left(\phi_i - h_i \right) \tag{11}$$

This flow can be simply added to the right-hand side of the model equation to give

$$\sum_{j=1, i \neq j}^{N} a_{ij} \phi_j + a_{ii} \phi_i = Q_i + C_i \left(h_i - \phi_i \right) \tag{12}$$

in which the diagonal was taken out of the matrix for clarity in the sequel.

The external flow $Q_{ex,i} = C_i \left(\phi_i - h_i \right)$ represents inward flow (positive if inward), just like the given inflow Q_i.

This way, each model node may be connected to the outside world having arbitrary fixed heads (lakes, rivers and so on).

The constant part $C_i h_i$ functions exactly like a fixed injection. The variable part, $C_i \phi_i$, may be put to the left-hand side of the equation, yielding

$$\sum_{j=1, i\neq j}^{N} a_{ij}\phi_j + \left(a_{ii} + C_i\right)\phi_i = Q_i + C_i h_i \tag{13}$$

This comes down to adding C_i to the diagonal matrix entry, $a_{ii} \rightarrow a_{ii} + C_i$.

In matrix form for Matlab

$$\left(\mathbf{A} + diag\left(\mathbf{C}\right)\right)\Phi = \mathbf{Q} + \mathbf{C}.*\mathbf{h} \tag{14}$$

Where $diag\left(\mathbf{C}\right)$ is an $N \times N$ diagonal matrix with the elements C_i. This is indeed equivalent to adding C_i to the diagonal elements a_{ii}.

The boundary conditions explained in this section are so-called general head boundary in Modflow jargon. Fixed-head boundaries are dealt with further down.

Modflow has two variants of these general head boundaries: drains and rivers. Drains differ in that they only discharge when the head is above drain level while river head boundaries differ in that the head below the river (i.c. in the model) does not affect the inflow when it falls below the river bottom. In that case the river bottom is used.

Drains and rivers thus make the model non-linear as they imply a switch which depends on the head itself. Such conditions are most efficiently implemented using iterative matrix solvers, so that the conditions can be updated during the solution process. Here we ignore this efficiency, we will use Matlab's standard matrix solver (backslash operation).

2.3.2 Drain boundaries

Drain works as general head boundaries if the head is above the drain elevation, while the discharge of the drain is zero when the head falls below this elevation. For the drains we thus need a drain elevation, i.e. a vector FD next to the drain conductances C_d.

The switch may be implemented as a Boolean vector H_d defined for drains to be 1 if $\phi > h_d$ and 1 if $\phi < h_d$:

$$\mathbf{H}_d = \left(\Phi > \mathbf{h}_d\right) \tag{15}$$

Hence, the drains are implemented as follows:

$$\left(\mathbf{A} + diag\left(\mathbf{C} + \mathbf{C}_d.*\mathbf{H}_d\right)\right)\Phi = \mathbf{Q} + \mathbf{C}.*\mathbf{h} + \mathbf{C}_d.*\mathbf{H}_d.*\mathbf{h}_d \tag{16}$$

2.3.3 River boundaries

River boundaries are also general head boundaries as long as the head remains above the bottom of the river. When it falls below the bottom, the infiltration is assumed to pass through the unsaturated zone without suction from the fallen head. So

$$Q_R = h_R - \phi, \ \phi \ge h_B$$
$$Q_R = h_R - h_B, \ \phi < h_B$$

Or

$$Q_R = C_R\left(h_R - \phi\right)\left(\phi \ge h_B\right) + C_R\left(h_R - h_B\right)\left(\phi < h_B\right)$$

$$Q_R = C_R\left(h_R - \phi\right)\left(\phi \ge h_B\right) + C_R\left(h_R - h_B\right)\left(1 - \left(\phi \ge h_B\right)\right)$$
$$Q_R = C_R H_R h_R - C_R H_R \phi + C_R h_R - C_R h_B - C_R H_R h_R + C_R H_R h_B$$
$$Q_R = C_R\left(h_R - h_B\right) - C_R H_R\left(\phi - h_B\right)$$

Where $H_R = \left(\phi \ge h_B\right)$

Similarly rivers can be implemented as follows

$$\left(\mathbf{A} + diag\left(\mathbf{C} + \mathbf{C}_d.*\mathbf{H}_d + \mathbf{C}_R.*\mathbf{H}_R\right)\right)\Phi = \mathbf{Q} + \mathbf{C}.*\mathbf{h} + \mathbf{C}_d.*\mathbf{H}.*\mathbf{h}_d + \mathbf{C}_R.*\left(\mathbf{h}_R - \mathbf{h}_B\right) + \mathbf{C}_R.*\mathbf{H}_R.*\mathbf{h}_B \quad \textbf{(17)}$$

The model then includes al general head, drain and river boundaries.

2.4 Solving the model and checking the results

From this the heads may be solved in Matlab simply by the backslash operator:

$$\Phi = \left(\mathbf{A} + diag\left(\mathbf{C}\right)\right) \backslash \left(\mathbf{Q} + \mathbf{C}.*\mathbf{h}\right) \quad \textbf{(18)}$$

$\mathbf{C}.*\mathbf{h}$ uses Matlab's ".*" operator, which stands for element-by-element multiplication instead of matrix multiplication. $\mathbf{C}.*\mathbf{h}$ is therefore a column vector with elements $c_i h_i$.

The latter system equation, which solves for the heads Φ ***is the complete system equation (i.e. the complete model) including all its boundaries.***

Now with the heads computed, we may calculate the net inflow of the nodes by the following matrix multiplication

$$\mathbf{Q}_{in} = \mathbf{A} * \Phi \quad \textbf{(19)}$$

Which must be zero when summed over the entire model

$$sum\left(\mathbf{Q}_{in}\right) = 0 \quad \textbf{(20)}$$

This is an easy check.

We may compute the inflow from all external fixed-head sources (negative if the flow is outward) from

$$\mathbf{Q}_{fh} = \mathbf{A} * \Phi - \mathbf{Q} \quad \textbf{(21)}$$

As already explained, most of the possible inter-nodal connections (a_{ij}) are zero, so that the final system matrix tends to be extremely sparse. This too is valid for all numerical models. To prevent having to deal with the zeros in over 99% of the system matrix, Matlab offers sparse matrices and sparse matrix functions. These sparse matrices work exactly like ordinary matrices and ordinary matrix functions, but sparse matrices only store and deal with the non-zeros elements. Sparse matrices make computing large models feasible on a PC.

2.5 Dealing with fixed heads (Dirichlet boundaries)

What we did above is using so-called general-head boundaries, i.e. fixed heads in the outside world that connect with the model through a conductance (or resistance). The general head boundaries were extended to specific forms of general head boundaries, the so-called drains and river boundaries. However, most models define directly fixed-head boundaries, separately (and in addition to) the variants of the general head boundaries mentioned.

One way to deal with fixed head boundaries is through the use of a very last conductance in applied general head boundaries, i.e. $C_i \to +\infty$.

So let us use an arbitrary conductance Γ which a very high value conductance (in practice take a value of 10^{10} or so) representing an infinite value of C_i.

Then for the fixed head nodes we have

$$\sum_{j=1, j\neq i}^{N} a_{ij}\phi_j + \left(a_{ii} + \Gamma\right)\phi_i = Q_i + \Gamma h_i \qquad \textbf{(22)}$$

Because $\Gamma \to \infty$ and so $\Gamma \gg \left|a_{ij}\right|$, then by dividing the left and right hand side by Γ, this equation reduces to

$$\phi_i \approx h_i \qquad \textbf{(23)}$$

This may be all what is needed to fix these heads and it works well in Matlab; if $\Gamma = 10^{10}$ or so, Most of the time, Matlab has no difficulty in solving the system, while a very accurate result is obtained.

2.5.1 Including fixed head boundaries directly

Rather than using an arbitrary large conductance to implement fixed heads, we may directly implement them. This improves the condition of the matrix and it reduces the amount of work, because the fixed heads nodes don't have to be computed at all, which reduces the computational size of the model.

Let the model be described by

$$\mathbf{A} * \Phi = \mathbf{Q} \qquad \textbf{(24)(25)}$$

Let the vector of these cell numbers with fixed heads be $\mathbf{I}_{\text{fh}}$ and let the vector of the remaining cells numbers be **I**, such that the union of **I** and $\mathbf{I}_{\text{fh}}$ comprises all cell numbers. Then $\mathbf{A}\left(:, \mathbf{I}_{fh}\right)$ represents all columns in **A** which will be multiplied by a fixed head. The fixed heads are represented by the vector $\Phi\left(\mathbf{I}_{fh}\right)$. Hence, $\mathbf{A}\left(:, \mathbf{I}_{fh}\right) * \Phi\left(\mathbf{I}_{fh}\right)$ is a constant vector which may be put directly to the right-hand side of the matrix equation, leaving the remaining columns $\mathbf{A}(:,\mathbf{I})$ to the left hand side to be multiplied with the remaining non-fixed heads $\Phi(\mathbf{I})$. This changes the model to:

$$\mathbf{A}(:,\mathbf{I})*\Phi(\mathbf{I})=\mathbf{Q}-\mathbf{A}(:,\mathbf{I}_{fh})*\Phi(\mathbf{I}_{fh}) \tag{26}$$

Because we only have to compute the heads at locations **I**, we get the reduced system of equations. The rows corresponding to the fixed heads may also be eliminated as the fixed heads need not to be computed at all. This results in the following matrix equation:

$$\mathbf{A}(\mathbf{I},\mathbf{I})*\Phi(\mathbf{I})=\mathbf{Q}(\mathbf{I})-\mathbf{A}(\mathbf{I},\mathbf{I}_{fh})*\Phi(\mathbf{I}_{fh}) \tag{27}$$

Hence the right hand side contains the constants and the left hand size contains the remaining equation (rows and columns), after removing those corresponding the fixed heads. The result is a computationally smaller model, that moreover is better conditioned. The model also can be up to ten times faster under certain circumstances.

So, the unknown heads become

$$\Phi(\mathbf{I})=\mathbf{A}(\mathbf{I},\mathbf{I})\backslash\left(\mathbf{Q}(\mathbf{I})-\mathbf{A}(\mathbf{I},\mathbf{I}_{fh})*\Phi(\mathbf{I}_{fh})\right) \tag{28}$$

If we initialize the head matrix with the fixed head matrix, the fixed head values are also contained in the head matrix. If the fixed head matrix contains NaN’s in all non-fixed head cells these cells are easily found

```
Ifh=Nodes(~isnan(FH));

I   =Nodes(  isnan(FH));
```

The nodal flow can be computed by

$\mathbf{Q}=\mathbf{A}*\Phi;$ (taken over all cells, including the fixed head cells). This gives the injection flow into each of the cells.

2.5.2 Inactive cells

In a large model, often substantial parts are inactive, i.e. represent bedrock or other parts without groundwater. The computation time may be substantially reduced if such cells are excluded beforehand. In practice it should be straightforward to find such cells from the input, as they represent cells where no water can flow or be stored, hence cells with all conductivities and the storage coefficient equal to zero. In Matlab these cells may be found as follows

Active cells that have a fixed head (preventing errors from input of fixed heads in non-active cells):

$$\mathbf{I}_{fh}=\left(\mathbf{K}x>0\,|\,\mathbf{K}_y>0\,|\,\mathbf{K}_z>0\,|\,\mathbf{S}>0\right)\&\left(\Phi_{FH}\neq NaN\right); \tag{29}$$

Active cells that are not fixed head (these cells have to be computed)

$$\mathbf{I}_a=\left(\mathbf{K}x>0\,|\,\mathbf{K}_y>0\,|\,\mathbf{K}_z>0\,|\,\mathbf{S}>0\right)\&\left(\Phi_{fh}==NaN\right); \tag{30}$$

In Matlab in 2D steady-state (omitting Kz and S)

Ifh=Nodes((Kx>0 | Ky>0) & ~isnan(FH)); % active fixed heads 2D

Ia=Nodes((Kx>0 | Ky>0) &   isnan(FH)); % active non fixed heads 2D

Matrices Kx, Ky,FH all have size (Ny,Nx).

The remaining cells are the active cells, denoted by the cell number vector $\mathbf{I}_a$.

All we need to do is exclude these non-active cells, which reduces the model to the number of active cells minus the number of fixed head cells

$$\mathbf{A}\left(\mathbf{I}_a,\mathbf{I}_a\right)*\Phi\left(\mathbf{I}_a\right)=\mathbf{Q}\left(\mathbf{I}_a\right)-\mathbf{A}\left(\mathbf{I}_a,\mathbf{I}_{fh}\right)*\Phi\left(\mathbf{I}_{fh}\right) \qquad \textbf{(31)}$$

This way, we have now reduced our model by both the fixed head and the inactive cells.

Solving for the active, non fixed head cells yields

$$\Phi\left(\mathbf{I}_a\right)=\mathbf{A}\left(\mathbf{I}_a,\mathbf{I}_a\right)\backslash\left(\mathbf{Q}\left(\mathbf{I}_a\right)-\mathbf{A}\left(\mathbf{I}_a,\mathbf{I}_{fh}\right)*\Phi\left(\mathbf{I}_{fh}\right)\right) \qquad \textbf{(32)}$$

The equation expresses that we only use the active and fixed head parts of the information; hence we still have the entire matrices and vectors available and thus may immediately compute the nodal water balances from.

To compute the cell-inflows to all active cells including the fixed-head cells, we use all active cells (2D steady state, leaving out Kz and S):

$$\mathbf{I}_{a1}=\left(\mathbf{K}x>0\,|\,\mathbf{K}_y>0\right) \qquad (33)$$

And compute

$$\mathbf{Q}\left(\mathbf{I}_{a1}\right)=\mathbf{A}\left(\mathbf{I}_{a1},\mathbf{I}_{a1}\right)*\Phi\left(\mathbf{I}_{a1}\right) \quad (34)$$

In Matlab

Ia1=Nodes(Kx>0 | Ky>0);

Q=zeros(size(Phi));

Q(Iact1)=A(Iact1,Iact1)*Phi(Iact1);

The non active cells will have Q=0;

2.6 Finite difference modeling

Until now, everything said was true for all numerical models and nothing specific has been said or done for finite difference modeling.

What remains to be done is the computation of the conductances. This is specific for each method. We also need a mesh (model network or grid that divides the model area in parts). The grid type is more or less specific for the method employed. The model grid determines the number of connections between the individual nodes.

We will deal with a 2D (1 layer) rectangular grid (with rectangular cells) with nodes (head points) at the cell centers (right-hand picture in Figure 1). In such a model, the flow from one node, *i*, to its neighbor, *j*, first passes half a cell with the conductivity (or transmissivity), k_i, of the first cell and then half a cell with the conductivity (or transmissivity), k_j, of the receiving cell. This is flow through two media placed in series, for which resistances add up (not conductances). Therefore, we compute the resistance to the flow between the two nodes and then take its inverse to get the conductance.

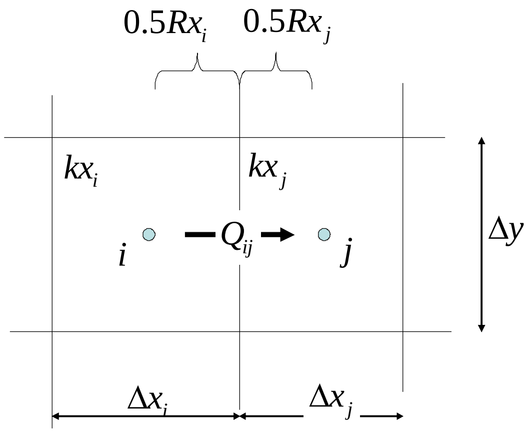


Figure 3: Resistance *Rx*=0.5*Rx*$_i$+0.5*Rx*$_j$ for flow from node *i* to *j*. Where *Rx*$_i$ and *Rx*$_j$ are the resistances for flow through an entire cell in *x*-direction

Let $\mathbf{Dx}$ be row vector with column widths and $\mathbf{Dy}$ a column vector with cell heights (width in y-direction) and $\mathbf{kx}, \mathbf{ky}$ the 2D matrices with the with the horizontal and vertical cell conductivities respectively. The resistance between adjacent nodes in *x*-direction then becomes:

$$\begin{aligned}\mathbf{Rx} = 0.5*\left(1./\,\mathbf{Dy}(:,1:end-1)\right)*\mathbf{Dx}(:,1:end-1)./\,\mathbf{kx}(:,1:end-1)+...\\ 0.5*\left(1./\,\mathbf{Dy}(:,2:end\quad)\right)*\mathbf{Dx}(:,2:end\quad)./\,\mathbf{kx}(:,2:end\quad)\end{aligned} \tag{35}$$

The conductances between these nodes is then given by

$$\mathbf{Cx} = 1./\,\mathbf{Rx} \tag{36}$$

Likewise for the resistance between adjacent nodes in y-direction gives

$$\begin{aligned}\mathbf{Ry} = 0.5*\mathbf{Dy}(1:end-1,:)*\left(1./\,\mathbf{D}x(1:end-1,:)\right)./\,\mathbf{ky}(1:end-1,:)+...\\ 0.5*\mathbf{Dy}(2:end\quad,:)*\left(1./\,\mathbf{D}x(2:end\quad,:)\right)./\,\mathbf{ky}(2:end\quad,:)\end{aligned} \tag{37}$$

and

$$\mathbf{Cy} = 1./\,\mathbf{Ry} \tag{38}$$

Note that here it is assumed that **D**y is a column vector and **D**x a row vector. Multiplying a column vector of length *Ny* with a row vector of length *Nx* yields a matrix of size *Ny***Nx* with elements equal to the product of the corresponding elements of the column and the row. In this case these elements are *dx*/*dy* and *dy*/*dx* respectively.

2.7 Numbering cells (nodes)

In equations (3) through (13) indices *i* and *j* were used to denote connected cells (or nodes). To allow numbering of cells in our model, we need to generate such cell numbers, which can be arbitrary, as is the case in a finite element model. The only requirement is that the numbers are unique.

When numbering cells of a finite difference model we will, of course, exploit the regular mesh structure. In Matlab it is straightforward to generate a vector of increasing numbers starting with 1 and ending with $Ny \times Nx$, which is the number of cells in the model (We will have $Ny \times Nx \times Nz$ cells in a 3D model):

$$Nodes = 1..Ny \times Nx;$$

Next we fold this vector into the shape of the model grid with its *Ny* rows and *Nx* columns. This is done using Matlab's reshape function:

$$Nodes = reshape(1..Ny * Nx, Ny, Nx)\,; \tag{39}$$

This way, the variable *Nodes*, is the matrix holding the cell numbers. The number of an arbitrary cell within the grid, say row *j* and column *i*, is now obtained from

cellNumber=Nodes(*j*,*i*)

2.8 Assembling the system matrix

In equations (3) through (13) the indices refer to cells that are connected. Hence a_{ij} is the coefficient (i.e. the negative of the conductance) between the cells *i* and *j*, which, of course, are just adjacent cells.

The system matrix element a_{ii} is the negative value of the conductance between these cells *i* and *j*, where the first index refers to the equation number, i.e. the cell for which the water balance is computed, and the second index, *j*, refers to an adjacent cell, to which it is connected. Clearly $a_{ij}=a_{ji}$, which makes the system matrix **A** symmetric.

To put a coefficient (negative of the conductance) at the correct position in the system matrix, we need its value and its indices *i* and *j*: i.e. the triple i,j,a_{ij}.

This is now straightforward with the available numbering.

We may generate the West-East cell pair indices with their corresponding coefficients as follows:

$$[\text{Nodes(:,1:end-1), Nodes(:,2:end), -}C_x] \tag{40}$$

Writing

$$I_W = \text{Nodes(:,1:end-1);} \tag{41}$$

$$I_E = \text{Nodes(:,2:end);} \tag{42}$$

we could write this more compactly as

$$[I_W \;\; I_E \;\; -C_x] \tag{43}$$

With

$$\mathrm{I_N{=}Nodes(1{:}end\text{-}1,{:});} \tag{44}$$

$$\mathrm{I_S{=}Nodes(2{:}end,{:});} \tag{45}$$

we may write the North-South pairs as

$$[\mathrm{I_N}\ \ \mathrm{I_S}\ \ \text{-}\mathrm{C_y}] \tag{46}$$

We may put all combinations in a single matrix

$$\begin{aligned}
&[\ [\mathrm{I_W}\ \ \mathrm{I_E}\ \ \text{-}\mathrm{C_x}];\ldots\\
&\quad [\mathrm{I_E}\ \ \mathrm{I_W}\ \ \text{-}\mathrm{C_x}];\ldots\\
&\quad [\mathrm{I_N}\ \ \mathrm{I_S}\ \ \text{-}\mathrm{C_y}];\ldots\\
&\quad [\mathrm{I_S}\ \ \mathrm{I_N}\ \ \text{-}\mathrm{C_y}]\ldots\\
&\ ]
\end{aligned} \tag{47}$$

Where the "…" is Matlab's line continuation and the ";" places what follows in subsequent rows of the matrix.

The problem with the last expression is that the embedded matrices are not column vectors but matrices having the shape of the model; i.e. the $\mathrm{I_W}$ $\mathrm{I_E}$ and $\mathrm{C_x}$ being of size $Ny\times(Nx-1)$ and the $\mathrm{I_N}$, $\mathrm{I_S}$ and $\mathrm{C_y}$ of size $(Ny-1)\times Nx$. We can turn them into column vectors by using the Matlab's (:) operator. This operator shapes matrices of any shape into their equivalent column vectors.

Hence,

$$\begin{aligned}
&[\ [\mathrm{I_W}(:)\ \ \mathrm{I_E}(:)\ \ \text{-}\mathrm{C_x}(:)];\ldots\\
&\quad [\mathrm{I_E}(:)\ \ \mathrm{I_W}(:)\ \ \text{-}\mathrm{C_x}(:)];\ldots\\
&\quad [\mathrm{I_N}(:)\ \ \mathrm{I_S}(:)\ \ \text{-}\mathrm{C_y}(:)];\ldots\\
&\quad [\mathrm{I_S}(:)\ \ \mathrm{I_N}(:)\ \ \text{-}\mathrm{C_y}(:)]\ldots\\
&\ ]
\end{aligned} \tag{48}$$

does the job.

The number of rows in this three-column matrix is thus

$$2Ny(Nx-1)+2(Ny-1)Nx$$

This completes all the connected nodes in this 2D model.

Because the matrix is symmetric, we could skip the East-West and South-North pairs and exploit this symmetry. We will show this below.

The three-column matrix above represents three column vectors with elements i,j,a_{ij} and j,i,a_{ji} respectively. These triples contain all non-zero elements of the system matrix with their position, except the diagonal elements. The latter may be readily computed by

$$a_{ii} = \sum_{j=1, j \neq i}^{Nodes} a_{ij} \tag{49}$$

To generate the sparse system matrix, we have to use the Matlab function *sparse* and pass it the indices and the matrix element value. However, sparse requires the columns *i*, *j* and a_{ij} to be specified as separate vectors. This is now readily done as follows:

$$\begin{aligned}\mathbf{A} = sparse(&[\ \ \mathbf{I_w}(:); \quad \mathbf{I_E}(:); \quad \mathbf{I_N}(:); \quad \mathbf{I_S}(:)],...\\ &[\ \ \mathbf{I_E}(:); \quad \mathbf{I_W}(:); \quad \mathbf{I_S}(:); \quad \mathbf{I_N}(:)],...\\ &[-\mathbf{C_x}(:); \ -\mathbf{C_x}(:); \ -\mathbf{C_y}(:); \ -\mathbf{C_y}(:)],...\\ &Ny*Nx,...\\ &Ny*Nx,...\\ &5*Ny*Nx)\end{aligned} \tag{50}$$

where the last three arguments of this function call are the size of the system matrix and the number of non-zero elements respectively (5 in a row for a 2D finite difference mode). The first three arguments are respectively the column of *i* indices, the column of *j* indices and the column of corresponding matrix coefficients, i.e. the negative values of the conductances. The minus sign is a sign choice. It implies that the flow from a node to its surrounding neighbors is positive, and so is an injection into the model which must deliver this net flow from a node to its neighbors. Hence, extractions such was water supply wells are negative, as is evaporation, while recharge is positive. This sign convention is easily remembered as any flow into the model gives rise to increasing water levels and vice versa.

Notice that in the actual model listed further below we use an equivalent Matlab form to enter this three column matrix which is just an exercise in reading Matlab matrices.

An alternative to expression (50) is using only the W-E and N-S pairs and exploiting the matrix symmetry:

$$\begin{aligned}\mathbf{A} = sparse(&[\ \ \mathbf{I_W}(:); \quad \mathbf{I_N}(:)];...\\ &[\ \ \mathbf{I_E}(:); \quad \mathbf{I_S}(:)];...\\ &[-\mathbf{C_x}(:); \ -\mathbf{C_y}(:)];...\\ &Ny*Nx,...\\ &Ny*Nx,...\\ &2Ny*Nx)\end{aligned} \tag{51}$$

And then due to symmetry construct the system matrix from this one and its transpose

$$\mathbf{A} = \mathbf{A} + \mathbf{A}'; \tag{52}$$

Having come so far, we only have 4 matrix elements per row and still miss the diagonal matrix elements a_{ii}. These diagonal matrix elements may be put into **A** using the function *spdiags* as follows

$$\mathbf{A} = spdiags(-sum(\mathbf{A},2),0,\mathbf{A}) \tag{53}$$

Where the 0 means zero-offset from the diagonal of matrix **A**, in which to put the diagonal, $-sum(\mathbf{A},2)$. We need Matlab’s function *spdiags* instead of its equivalent *diags* because we work with sparse matrices.

3 The actual model in Matlab

The model will be a Matlab function accepting the needed arguments and yielding the heads and the computed nodal flows. This function resides in an “m.file” which is Matlab’s text file storage of scripts and functions having extension “.m”. A second m.file will be a Matlab script, which is just set of commands in a file. This script will be used to set up the model, specify its boundary conditions, call the function (i.e. the actual model) and finally visualize the results by contouring the computed heads.

To make the model in Matlab, launch Matlab browse to the directory where you want to store the “m.files” of this model.

Then start with opening a new file by pressing the correct icon of the Matlab editor and immediately save it with the desired model name, for instance *fdm2*. “*fdm2.m*” will be the file name given by Matlab and *fdm2* the name of the Matlab function it contains.

The first line in *fdm2.m* gets the function name, and its (multiple) output and inputs. All inputs are matrices and vectors to be defined in the script that we will use to call the model.

To write the calling script, open another new file. Save it with the name “*modelscript*” for example.

The model will be called from this script as follows

```
[Phi,Q]=fdm2(x,y,kx,ky,FH,FQ);
```

However, since the model does not yet exist and to ease debugging we use the model file as a script. So, “comment out” this call as follows by prefixing a “%” and further prefix the call to the file *fdm2.m*:

```
fdm2;      % [Phi,Q]=fdm2(x,y,kx,ky,FH,FQ);
```

This command (call) will simply run the lines in the file *fdm2.m* as typed-in commands.

To set up an arbitrary trial model of ten columns that are 10 m wide and twelve rows that are 6 m high, place the following lines in the modelscript before the call of the *fdm2* file

```
x=-100:10:100; y=(100:-10:-100)’;     % x hor, y vert (transposed)
Dx=diff(x); Dy=abs(diff(y));          % compute column and row sizes
Nx=length(Dx); Ny=length(Dy);         % compute size of model
xm=0.5*(x(1:end-1)+x(2:end));         % coordinates of cell centers
ym=0.5*(y(1:end-1)+y(2:end));         % coordinates of cell centers
Kx=10*ones(Ny,Nx); Ky=5*ones(Ny,Nx);  % conduct. (transmissivities)
FH=NaN*ones(Ny,Nx); FH(:,end)=0;      % fixed head matrix
FQ=zeros(Ny,Nx);  FQ(2,3)=-2400;      % fixed flow (one well at (2,3))
N=0.001;                              % recharge (0.001 m/d)
Q=Q+N*Dy*Dx;                          % add recharge equal to N as flow
fdm2;      % [Phi,Q,Qx,Qy]=fdm2(x,y,kx,ky,FH,FQ);  % call the model
contour(xm,ym,Phi);                   % contour the computed heads
%surf(xm,ym,Phi);                     % if you like show heads as 3D
xlabel(’x in m’); ylabel(’y in m’); title(’head contours’);
% Some checks
sum(sum(Q))          % overall model water balance (must be zero)
sum(Q(:,end))        % total outflow across right hand boundary
sum(Q(find(~isnan(FH))) % total flow over all fixed head boundaries
```

This completes the model script. It sets up the model dimensions, the cell properties, fixed head and flow boundaries for all nodes and runs the model. Finally it contours the heads and computes some integrated flows. Note that the heads are computed for the cell centers, so that we need to compute those centers first.

Now let's focus on the model and the model script *fdm2.m*. The first line should be the function heading

```
function [Phi,Q,Qx,Qy]=fdm2(x,y,kx,ky,FH,FQ)
```

It defines a function called *fdm2* with arguments to be passed to it (and will be local inside the function). It also defines its output, which may be multiple as is the case here, where we will obtain the computed heads, the computed nodal flows, the computed horizontal flows across cell faces and the computed vertical flow across cell faces.

To start, comment this line out, because during the construction of the model we will run the file as a script to ease debugging

```
% function [Phi,Q,Qx,Qy]=fdm2(x,y,kx,ky,FH,FQ)   % "%" is used to
comment out
```

When run as a script, the parameters are not local; the parameters in the model script and visible in *fdm2* as well as the parameters in *fdm2* in *modelscript*. This will be no longer the case when the first line in *fdm2.m* is a proper function definition (function header).

Below the function definition line, insert a number of comment lines (i.e. all starting with "%") to provide the information that Matlab given whenever you type

>>help fdm2

in the command window.

```
function [Phi,Q,Qx,Qy,Psi]=fdm2(x,y,kx,ky,FH,Q)
% function [Phi,Q,Qx,Qy,Psi]=fdm2d(x,y,kx,ky,FH,Q)
% 2D block-centred steady-state finite difference model
% x,y mesh coordinates, kx,ky conductivities
% FH=fixed heads (NaN for ordinary points), Q=fixed nodal flows
% Phi,Q computed heads and cell balances
% Qx is horizontal cell face flow positive in positive x direction
% Qy is vertial    cell face flow, postive in positive y direction
% Psi is stream function assuming bottom of model is zero (impervious)
% TO 991017  TO 000530 001026 070414 080226

x=x(:)'; Nx=length(x)-1; dx=diff(x); xm=0.5*(x(1:end-1)+x(2:end));
y=y(:);  Ny=length(y)-1; dy=abs(diff(y));

Nodes = reshape(1:Nx*Ny,Ny,Nx);               % Node numbering
IE=Nodes(:,2:end);   IW=Nodes(:,1:end-1);
IS=Nodes(2:end,:);   IN=Nodes(1:end-1,:);

warning('off','all');  % allow division by zero for inactive cells
RX=0.5*(1./dy)*dx./kx; Cx=1./(RX(:,1:end-1)+RX(:,2:end)); % hor  conductances
RY=0.5*dy*(1./dx)./ky; Cy=1./(RY(1:end-1,:)+RY(2:end,:)); % vert conductances
warning('on','all'); % restore warning message

A=sparse([IE(:);IW(:);IN(:);IS(:)],...
         [IW(:);IE(:);IS(:);IN(:)],...
         -[Cx(:);Cx(:);Cy(:);Cy(:)],...
         Ny*Nx,Ny*Nx,5*Ny*Nx);                 % System matrix
Adiag= -sum(A,2);                              % Main diagonal
```

```
IAct =Nodes( kx>0 | ky>0);              % active cells
IAct1=Nodes((kx>0 | ky>0) &  isnan(FH)); % active cells but not fixed heads
Ifh  =Nodes((kx>0 | ky>0) & ~isnan(FH)); % active cells with fixed heads

Phi=FH(:);  % make sure Phi and Q are column vectors, otherwise it won't work
Q  =FQ(:);  % in case the groundwater problem is a single row.

% solve
Phi(IAct1)=spdiags(Adiag(IAct1),0,A(IAct1,IAct1))\(  Q(IAct1)-A(IAct1,Ifh)*Phi(Ifh));
Q(IAct)   =spdiags(Adiag(IAct ),0,A(IAct ,IAct ))* Phi(IAct ); % nodal flows

Phi=reshape(Phi,[Ny,Nx]);   % reshape back to shape of original model
Q  =reshape(Q  ,[Ny,Nx]);   % same for Q

Qx=-Cx.*diff(Phi,1,2)*sign(x(end)-x(1)); Qx(isnan(Qx))=0;  % horizontal cell face flows
Qy=-Cy.*diff(Phi,1,1)*sign(y(end)-y(1)); Qy(isnan(Qy))=0;  % vertical cell face flow

Psi=[flipud(cumsum(flipud(Qx),1));zeros(size(Qx(1,:)))];   % Stream function
% =====================================================
```

The model must be set up and debugged line by line. This is done by selecting one or more lines, running them by pressing F9 and checking if they are correct. Once all lines run smoothly and correctly, remove the comment in the first line of the fdm2 file. This makes the file a function. Also change the call to the file *fdm2* into a function call. So

```
fdm2;      % [Phi,Q,Qx,Qy,Psi]=fdm2(x,y,kx,ky,FH,FQ);  % call the model
```

becomes

```
[Phi,Q,Qx,Qy,Psi]=fdm2(x,y,kx,ky,FH,FQ);  % call the model
```

Then the model can be run with any changed input.

Note: There are no error checks in the model. This is to keep the file short. You may add checks that verify the size of the input matrices and vectors with respect to the model dimensions implied in the *x* and *y*.

3.1 Exercises

1 Prove that your model is correct, by comparing its results with analytical solutions

--- Compute the heads in a 1d model with recharge

The analytical solution can be found in Geohydrology I

$\phi = \frac{n}{2kD}\left(L^2 - x^2\right)$ with L and x are measured from the water divide to the boundary

```
% script to compute this case and compare with analytical solution
kD=100;             % transmissivity to be used
L =200;             % half width of model
x=-L:5:L;         % generate x-coordinates for mesh
y= [10 -10]';     % one row suffices because problem is 1D

xm=0.5*(x(1:end-1)+x(2:end)); dx=diff(x);      Nx=length(dx);
ym=0.5*(y(1:end-1)+y(2:end)); dy=abs(diff(y)); Ny=length(dy);

kx=kD*ones(Ny,Nx); ky=kx;   % same kD in every cell, kyD=kxD (ky=kx)
```

```
FH=NaN*ones(Ny,Nx); FH(:,[1,end])=0.0;        % Fix left and  right head

n=0.001; FQ=n*dy*dx;                          % Set recharge

[Phi,Q,Qx]=fdm2(x,y,kx,ky,FH,FQ);             % Run model

fi=n/(2*kD)*(L.^2-xm.^2);                   % Analytical solution
plot(xm,Phi,'r+',xm,fi,'b');              % Plot results
title('compare model with phi=n/(2kD)(L^2-x^2)');
xlabel('x [m]'); ylabel('Phi [m]');
legend('model','analytical');
```

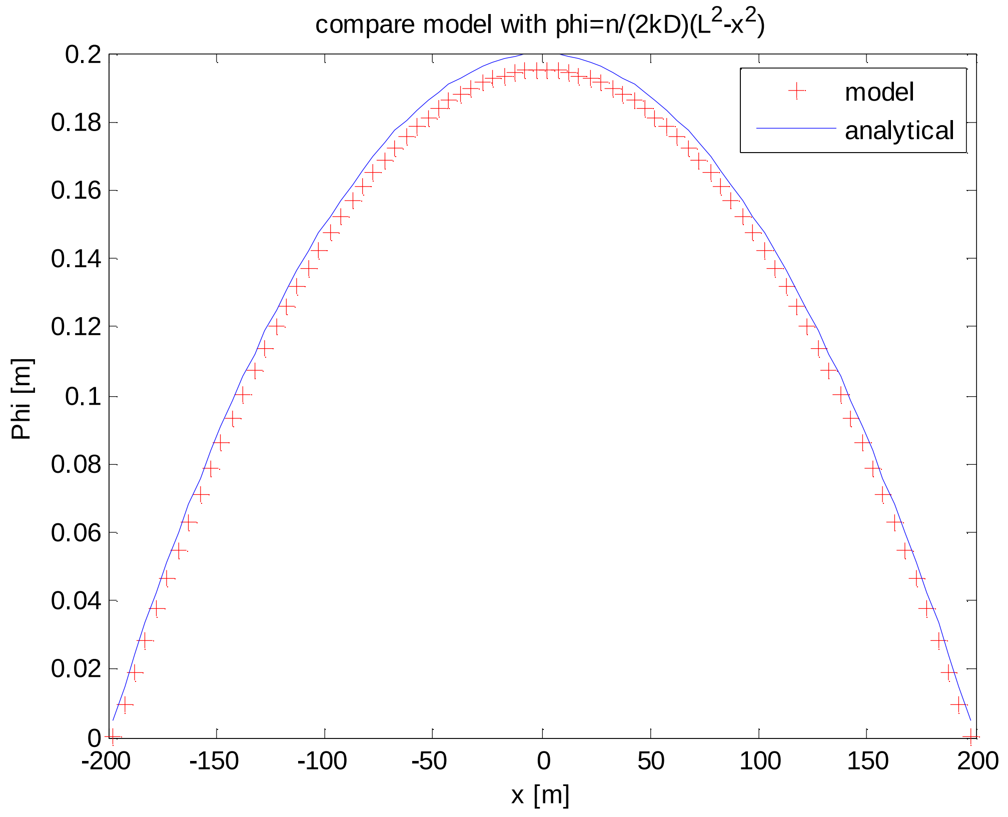


Figure 4 Comparison between model and analytical solution. The match is not perfect due to the fact that the exact locations of the outer cells in the model do not coincide with –L and L, the exact location of the outer nodes is in the cell center, see below to solve this.

As is clear the numerical and analytical solutions do not match completely. This is due to the fact that the boundaries nodes of the numerical model are not at –L and L but at the cell centers with is at –L+2.5=-197.5 m and at L-2.5=+197.5 m.

To solve this you may add a very thin outer cell at both ends

or set

```
x=-L-2.5:5:L+2.5;  % generate x-coordinates for mesh
```

This will yield the perfect match between model and analytical solution

--- Compute the heads in a 1d model with leakage for an aquifer below an aquitard, while above the aquitard the head is maintained at zero. The flow is symmetrical with *x*=0 in the center. All heads are relative to the maintained water level above the aquitard while the head at *x*=*L* and *x*=-*L* is maintained at *H*.

$$\phi = H\frac{\cosh\left(x/\lambda\right)}{\cosh\left(L/\lambda\right)}$$

This problem is a 1-d problem which may be solved by two rows representing the cross section with the first layer being the aquitard with resistance c and the second being the aquifer with given transmissivity *kD*.

```
% Cross section through polder with fixed head H at both sides
L=1000; kD=1600; c=100; lambda=sqrt(kD*c); % +/- Xsec of Bethune polder
x=[-L-5:10:L+5];
y=[0 -10 -40]';

xm=0.5*(x(1:end-1)+x(2:end)); dx=diff(x);      Nx=length(dx);
ym=0.5*(y(1:end-1)+y(2:end)); dy=abs(diff(y)); Ny=length(dy);

k=[dy(1)/c kD/dy(2)]';   % conductivities from c, kD and thickness
kx=k*ones(1,Nx); ky=kx;

H=-2.75;                 % head at left and right boundary
FH=NaN*ones(Ny,Nx);      % NaN matrix to store fixed heads
FH(1,:)=0;               % head above aquitard
FH(2,[1,end])=H;         % head at left and right boundary

FQ=zeros(Ny,Nx);         % matrix to store fixed Q's

[Phi,Q,Qx]=fdm2(x,y,kx,ky,FH,FQ);  % run model

fi=H*cosh(xm/Lambda)./cosh(L/Lambda); % analytical

plot(xm,Phi,'+',xm,fi,'b');
title('compare model with phi=H*cosh(x/lambda)/cosh(L/lambda)');
xlabel('x [m]'); ylabel('Phi [m]');
legend('fixed head','model','analytical');
```

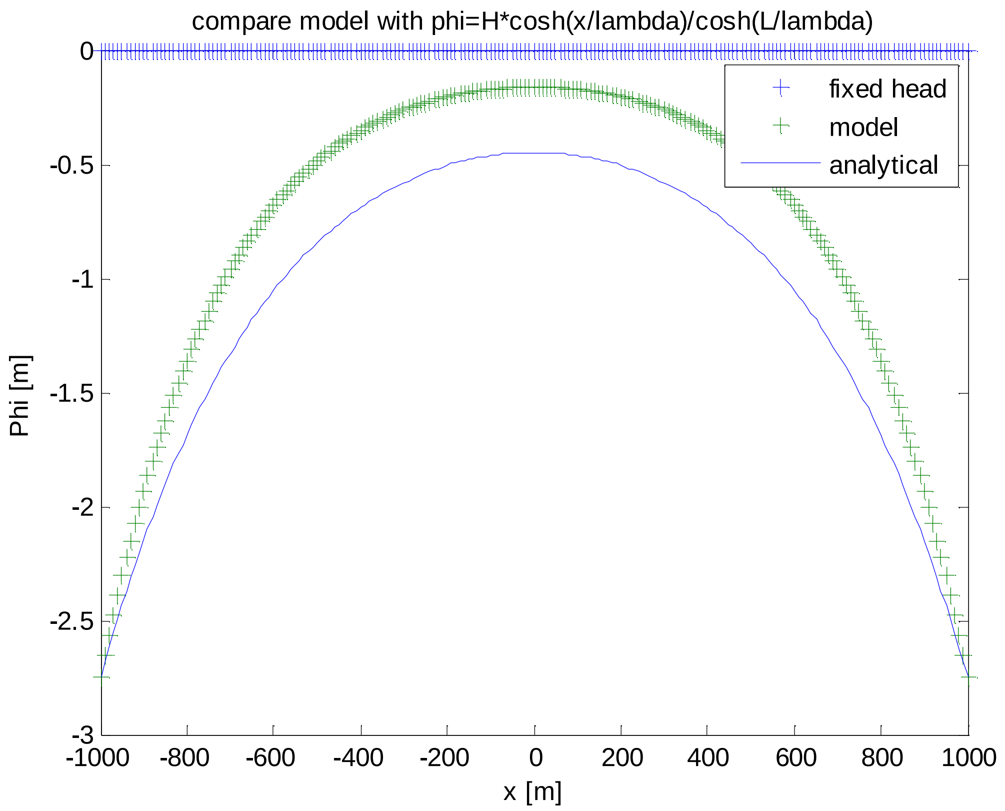


Figure 5: Comparison between model and analytical solution of cross section through polder. Obviously there is a big difference between the two. See text below to solve.

The outcome of the script is given in the figure above. The model results for both layers are shown. The first is the fixed heads in the top layer and the second are the computed heads in the second layer. Clearly, the model is way off compared with the analytical solution. Again this is due to the fact that the model nodes are in the center of the cells and therefore in the center of the layer. The resistance to vertical flow between the model and the bottom of the first layer is therefore only half of that between the top and the bottom of this layer. To solve this, you may use a thin layer on top and specify the head in that one. Or you may double the thickness of the first layer so that the resistance between the node and the bottom of this layer equals the desired value. Or you just half the conductivity of the first layer to get the same result. So, in the last case do this

```
k=[0.5*dy(1)/c kD/dy(2)]';   % conductivities from c, kD and thickness
```

--- Compute the drawdown due to well in the center, corners

The drawdown due to a well with extraction Q in the center of a circular island with radius R around which the head is maintained at a value zero reads

$$s = \frac{Q}{2\pi kD}\ln\left(\frac{R}{r}\right)$$

--- recharge on a square island of half length L, with given parameters

This is simple, generate coordinates, set FH=0 along all 4 boundaries, add recharge to the nodes and run the model.

```
L=200; dL=5; kD=150; R=170; n=0.001;
x=-L-0.5*dL:dL:L+0.5*dL;
y=flipud(x');
xm=0.5*(x(1:end-1)+x(2:end)); dx=diff(x);      Nx=length(dx);
ym=0.5*(y(1:end-1)+y(2:end)); dy=abs(diff(y)); Ny=length(dy);
kx=kD*ones(Ny,Nx); ky=kx;
FH=NaN*ones(Ny,Nx);
FH([1,end],:)=0; FH(:,[1,end])=0; % all boundaries 0
FQ=n*dy*dx;                          % recharge as nodal flows
 [Phi,Q]=fdm2(x,y,kx,ky,FH,FQ);       % run model
contour(xm,ym,Phi);
title('recharge on square island'); xlabel('x [m]'); ylabel('y [m]');
```

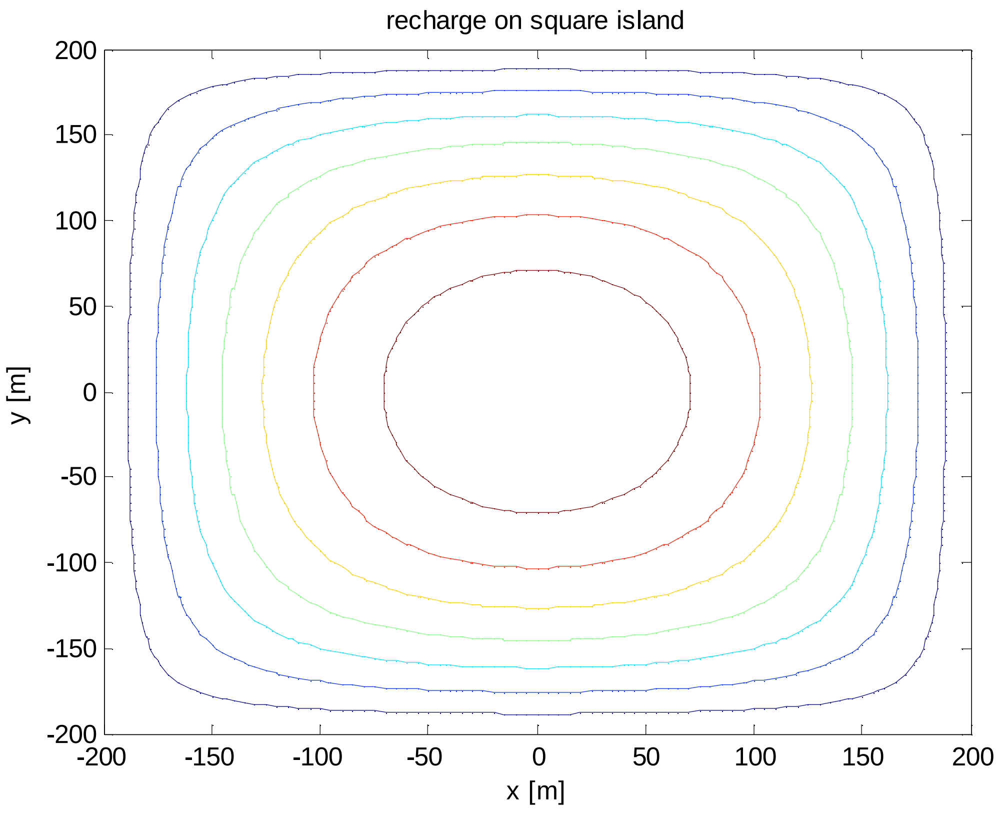


Figure 6: Recharge on a square island, contours may be labeled (see *help contour* in Matlab and how to use *clabel*, you may also use colorbar to get a colorscale see *help colorbar* in Matlab)

--- Recharge on a circular island

Problem with the rectangular island is that there is no easy analytical solution for it. Therefore to proof our model is correct, let’s compute the head in a circular island with the same model.

The analytical solution is derived from the partial differential equation and boundary condition that h=0 for r=R.

$$-2\pi rkD\frac{dh}{dr}=n\pi r^2, \quad \rightarrow \quad h=\frac{n}{4kD}\left(R^2-r^2\right)$$

Below is the same script as before, only the boundary condition is changed. What we do we set all nodes where *r*>=*R* equal to zero. *R* is here the distance from point *x*=0, *y*=0 and R the given radius of the model. Given the coordinates of the nodes *xm* (row vector) and *ym* (column vector), we can compute a matrix with the size of the Nodes containing the distance *r* in each cell as follows:

```
r=sqrt((ym*ones(size(xm))).^2+(ones(size(ym))*xm).^2);
```

and then set the fixed head FH equal to zero in all nodes for which *r*>=*R*:

FH(r>=R)=0;

Hence

```
% Head in circular island with recharge

L=200; dL=5; kD=150; R=170; n=0.001;
x=-L-0.5*dL:dL:L+0.5*dL;
y=flipud(x');
xm=0.5*(x(1:end-1)+x(2:end)); dx=diff(x);      Nx=length(dx);
ym=0.5*(y(1:end-1)+y(2:end)); dy=abs(diff(y)); Ny=length(dy);
kx=kD*ones(Ny,Nx); ky=kx;
FH=NaN*ones(Ny,Nx);
r=sqrt((ym*ones(size(xm))).^2+(ones(size(ym))*xm).^2);  %Distance
FH(r>=R)=0;
FQ=n*dy*dx;                                % recharge as nodal flows
[Phi,Q]=fdm2(x,y,kx,ky,FH,FQ);             % run model
contour(xm,ym,Phi);
title('recharge on circular island'); xlabel('x [m]'); ylabel('y [m]');
```

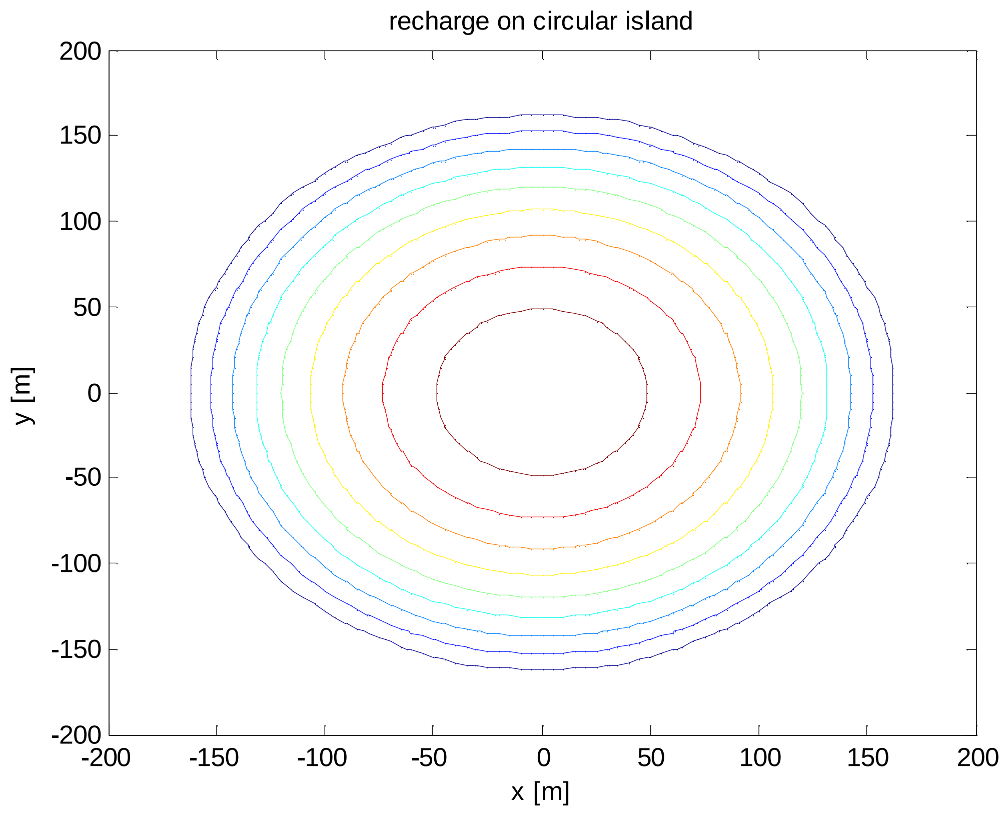


Figure 7: Head in circular island of *R*=170 m with recharge

To compare this with the analytical solution, plot the head through the center together with the analytical solution. So add these lines

```
figure
fi=n/(4*kD)*(R.^2-xm.^2);
plot(xm,Phi(xm==0,:),'r+',xm,fi,'b');
legend('numeric','analytic');
title('compare model with Phi=n/(4kD)*(R^2-r^2)'); xlabel('r [m]');
ylabel('Phi [m]');
```

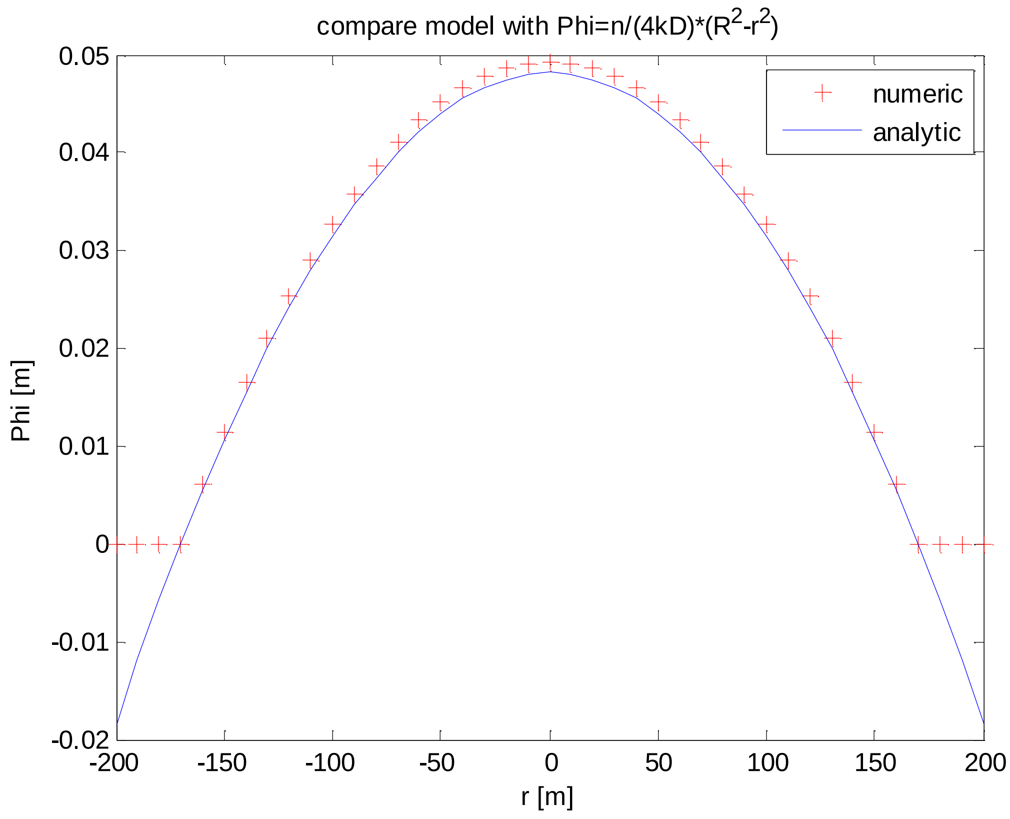


Figure 8: Comparison between model and analytical solution. Obviously there is a small difference. See text below how to solve this.

The figure above shows a small difference between the head through the center of the model and the analytical solution. Is this model wrong? No and yes. It is correct but just a little inaccurate. If we reduce the size of the elements to 2 m and we'll see that the difference between the two models has completely disappeared. So the model is correct after all, but for this computation we need a smaller cell size, which will be due to the fact that square or rectangular cells do not nicely match with circular shapes like the boundary of the island and especially the true head contours.

--- Show the effect of anisotropy?

Anisotropic situations can be readily computed if the main conductivities align with the *x* and *y* axes of the model grid. Just try it. However, it is not straightforward to apply anisotropy in arbitrary directions, unless the model grid can be rotated to align with the main conductivity directions. In the finite element method, anisotropy in arbitrary direction in each cell is natural. In the finite difference model it is generally limited to the main directions of the grid itself. There exist however several methods to apply anisotropy in rectangular grids. This is beyond this course.

--- Generate and solve a complex cross section with a given boundary condition at the top

To model a cross section, the y direction is simply regarded vertical. Nothing changes, except that in cross sections we pass the conductivities of the cells, while in flat aquifers we pass the transmissivities instead. For the model this makes no difference. Because cross sections combine so nicely with streamlines, we skip this till after we implemented these streamlines.

5 Generate a random conductivity field and compute the heads given fixed head boundaries.

A random conductivity field may be generated using the rand(Ny,Nx) function.

6 Generate a river through your model and compute the heads with recharge

A river is a set of lines with fixed heads or heads that are fixed through a resistance, which are called “general head boundaries” in Matlab. Normally assigning rivers to a grid is a GIS action. The river has to be intersected with the model cells. For each cell the intersecting surface area A [m^2] is computed and converted into a conductance C [m^2/d] using the bottom resistance c [d] of the river $C = A / c$. So we skip this GIS action for now.

4 Stream lines

Streamlines are lines of constant stream-function value. Flow lines are lines followed by particles. Therefore, flow lines have to be computed by tracing particles, but streamlines (if they exist) can be computed by contouring the stream function, without tracing. However, the stream function is only defined in 2D steady-sate flow without sources and sinks (and leakage or recharge for that matter). In practice, individual sources and sinks can be dealt with and will look similar to wells in the 2D image.

Because our model is 2D, streamlines will often be an efficient manner to show the flow in a quantitative way. A very powerful characteristic of streamlines is that the flow is known between any pair of points in the model. Further, the flow between any pair of streamlines is constant and equal to the difference of the stream function values.

With respect to the stream function, any streamline may be designated the zero line, after which the values of all other streamlines are fixed. The stream function can be computed by integrating the flow across an arbitrary line cutting streamlines. Assuming the bottom of the model is a streamline, we can compute the stream function easily by integrating the horizontal flow across cell faces from the bottom to the top of the model.

Mathematically

$$\psi = \int_{y\min}^{y\max} q_x(y)\,dy \tag{1}$$

When we use the horizontal flow across the cell faces as an extra output of the model, we just cumulate these along the cell faces upward from the bottom of the model. This gives the stream function values at all cell corners (not the nodes). This stream function may subsequently be contoured, which yields stream lines.

To implement the stream function, open a new Matlab file and save it as "*Psi.m*". Type the following and save again

```
function P=Psi(Qx)
% P=Psi(Qx)
% Stream function assuming bottom is stream line
% size of Psi is Ny+1,Nx-1, to contour is do
% contour(x[2:end-1),y,Psi(Qx));
P=flipud(cumsum(flipud([Qx;zeros(1,Nx)])));
```

It does the following. It receives the horizontal flows across the cell faces, which is the third output to the model. It adds a line of zeros through the bottom, because this will be the starting stream line with stream function value zero. Then we want to cumulate this matrix vertically from the bottom upward. Matlab's *cumsum*(..) accumulates matrices (try it), but starts at the top working downward. So we flip the matrix up-down before calling *cumsum(..)*. When done we *flipud(..)* again to put it right.

Next, add the following lines to your script file

```
contour(x(2:end-1),y,Psi(Qx)); % streamlines
```

The values of the stream function is the flow between any point and the bottom of the model. If you want that in specific steps of say dPsi=0.1 m^2/d, just specify the contours, for instance

```
contour(x(2:end-1),y,Psi(Qx),[0:dPsi:max(Psi(Qx(:)))]); % streamlines
```

See *help contour* in Matlab for details.

Clearly, if the left hand boundary if you models is a streamline, you may just as well integrate the vertical flow across horizontal cell faces along the horizontal lines.

4.1 Exercises streamlines

1 A symmetric cross section through a long building with sheet pilings 15 m deep at x=20 m. Dewatering wells are placed inside this sheet piling between 6 and 11 m depth. Compute the necessary extraction to dewater the pit by 5 m. The aquifer is semi-confined. All elevations are relative to the fixed head at the top. The sheet piling is between x=19.9 and x=20 m, and z=0 and z=-15 m with conductivity kW=0.0001 m/d. Wells are between x=19.8 and 19.9 m and between z=-6 and z=-11 m and are modeled as an extraction line in this cross section. The layers are given as in the Matlab script below.

The results are shown in the two figures. One shows the total cross section and the second one a detail. This detail demonstrates the streamlines and the head lines in this cross section in the neighborhood of the wells and the sheet piling. It clearly shows how the groundwater flows underneath the 15 m deep sheet piling towards the wells at the inside of this sheet piling between -6 and -11 m. The extraction is 8.67 m^2/d, which can be computed by summing the *Q* over the wells. Alternatively one may compute the *Q* entering the model through the first layer (sum(Q(1,(:))). We set the FH of the wells at -5 m. The head in the center immediately below the building pit is then -4.6 m. By setting FH in the wells to -5.4 will make sure the drawdown under the building pit is the required 5 m. The extraction will then be 9.38 m^2/d. This demonstrates the influence of partial penetration.

Partial penetration means that the well screen only penetrates part of the aquifer thickness. This implies that the drawdown is larger than in the case of a fully penetrating screen. Partial penetration is the usual case, to save well money or to prevent upcoming of brackish water from below. If a building pit must be put dry, only the head at its bottom needs to be lowered, not at the bottom of the aquifer, which may be 100 m or more thick. Using short screens then reduces the amount of water that needs to be extracted.

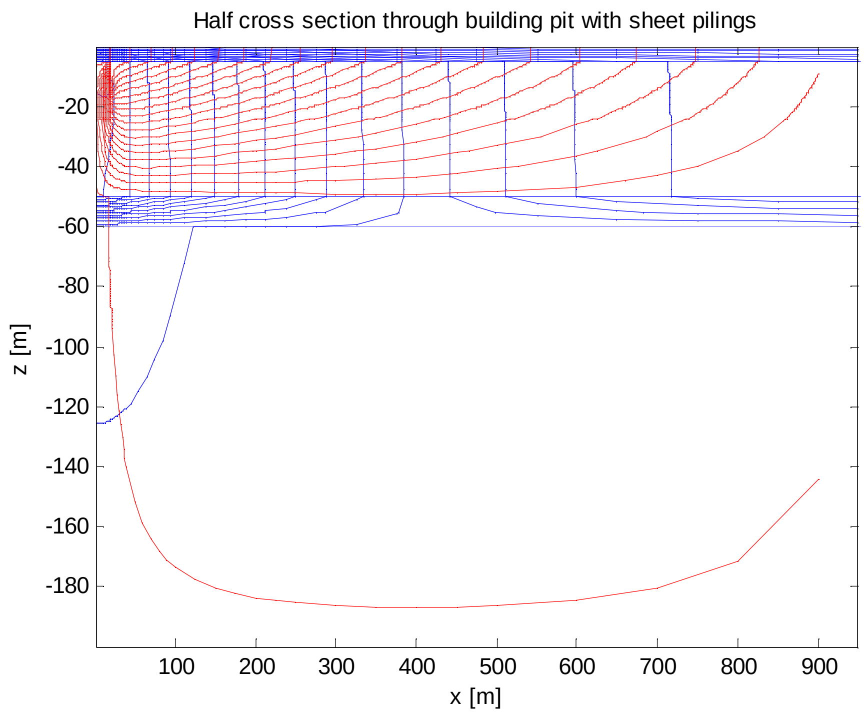


Figure 9: Building pit cross section with partially penetrating sheet piling and extraction wells at the inside

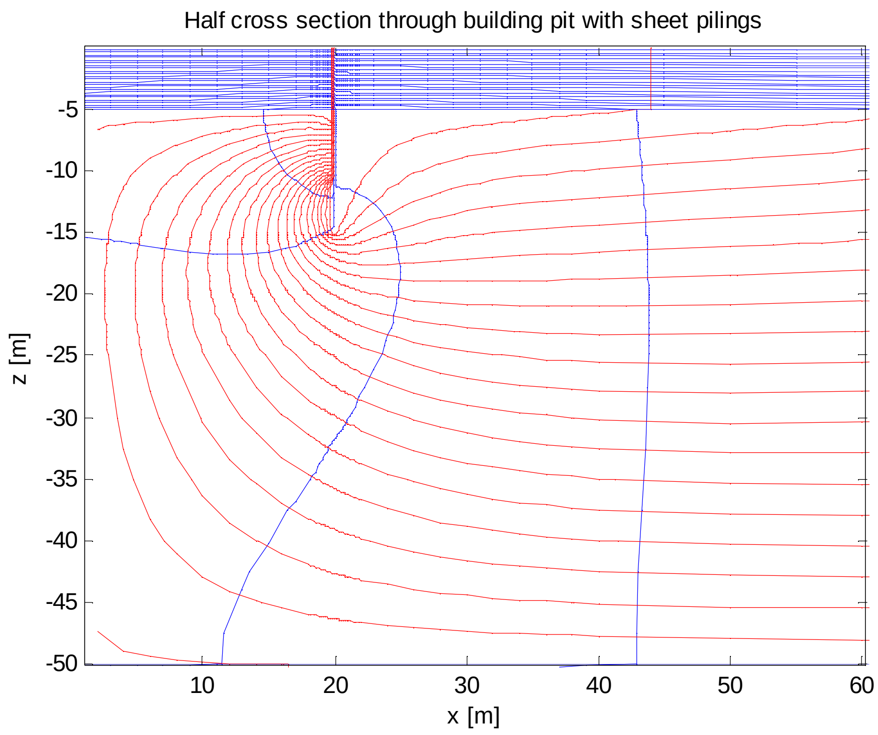


Figure 10: Detail showing the streamlines underneath the 15 m deep sheet piling towards the partially penetrating wells at its inside between 6 and 11 m depth

This is the script to create the cross section. You get the detailed one by zooming first horizontally and then unrestricted (select zoom and then right mouse button for options). To get you figure in word, use “edit copyfigure” in Matlab in the menu of the selected figure and then paste it in Word or PowerPoint as usual.

```
layers={                      % specify layers
  'clay' 0 -5  0.02           % material, top, bottom k
  'sand' -5 -50  20
  'clay' -50 -60 0.01
  'sand' -60 -200  30
};
xW    =[19.9 20  ]; yW    =[ 0 -15]; kW=0.0001;
xWells=[19.8 19.9]; yWells=[-6 -11]; FHWells=-5;

% the column and row coordinates are refined where needed to have
% a very detailed result (top and bottom of wells and sheet piling
% just add coordinates then apply unique to sort out
x=unique([0:2:18, 18:0.2:22 19:0.1:21, 22:2:40, 40:10:100,…
     100:25:250, 250:50:500, 500:100:1000]);  % fine mesh where needed
L=[-5 -50 -60 -200];  % layer boundaries for generating y values
y=[0 -0.01 L, L+0.01, -5:-0.1:-7, -7:-0.5:-14, -15:-0.1:-16, …
-16:-0.5:-19.5, -19.5:-0.1:-20.5, -20.5:-0.5:-25, -25:-5:-50];
y=sort(unique(y),'descend')';  % Unique + sort downward

xm=0.5*(x(1:end-1)+x(2:end)); dx=diff(x);      Nx=length(dx);
ym=0.5*(y(1:end-1)+y(2:end)); dy=abs(diff(y)); Ny=length(dy);

% get k values from specified layers
kx=zeros(Ny,Nx);
for i=1:size(layers,1);
    kx(ym<=layers{i,2}&ym>layers{i,3},:)=layers{i,end};
end

% set k in sheet piling to its given value
kx(ym<yW(1) & ym>yW(2), xm>xW(1) & xm<xW(2))=kW;  ky=kx;   % deep wall

FH=NaN*ones(Ny,Nx); FH(1,:)=0.0;

% set fixed head in wells to its given value
FH(ym<yWells(1) & ym>yWells(2), xm>xWells(1) & xm<xWells(2))=FHWells;

FQ=zeros(Ny,Nx);  % no fixed Q this time

[Phi,Q,Qx]=fdm2(x,y,kx,kx,FH,FQ);

close all   % ============plotting ===============================
contour(xm,ym,Phi,-5:0.2:0,'b');
hold on
contour(x(2:end-1),y,Psi(Qx),20,'r');
for i=1:size(layers,1)
    plot([x(1) x(end)],[layers{i,2},layers{i,2}]);
end
title('Half cross section through building pit with sheet pilings');
xlabel('x [m]'); ylabel('z [m]');

% ====water balance and computed head below pit ===================
sum(sum(Q(ym<yWells(1) & ym>yWells(2), xm>xWells(1) & xm<xWells(2))))
sum(sum(Q(1,:)))  % infiltration through top of model
sum(sum(Q))  % overall water balance
Phi(ym<-5 & ym>-6,1)   % head below building pit
```

Clearly this can just as well be done for a radial symmetric model, which we develop next. In fact, we only have to add the 7th argument in the model-call and the same problem will be computed for a circular building pit. Nothing else needs to be done.

```
[Phi,Q,Qx]=fdm2(x,y,kx,kx,FH,FQ,'radial');
```

The result for the detail is shown below

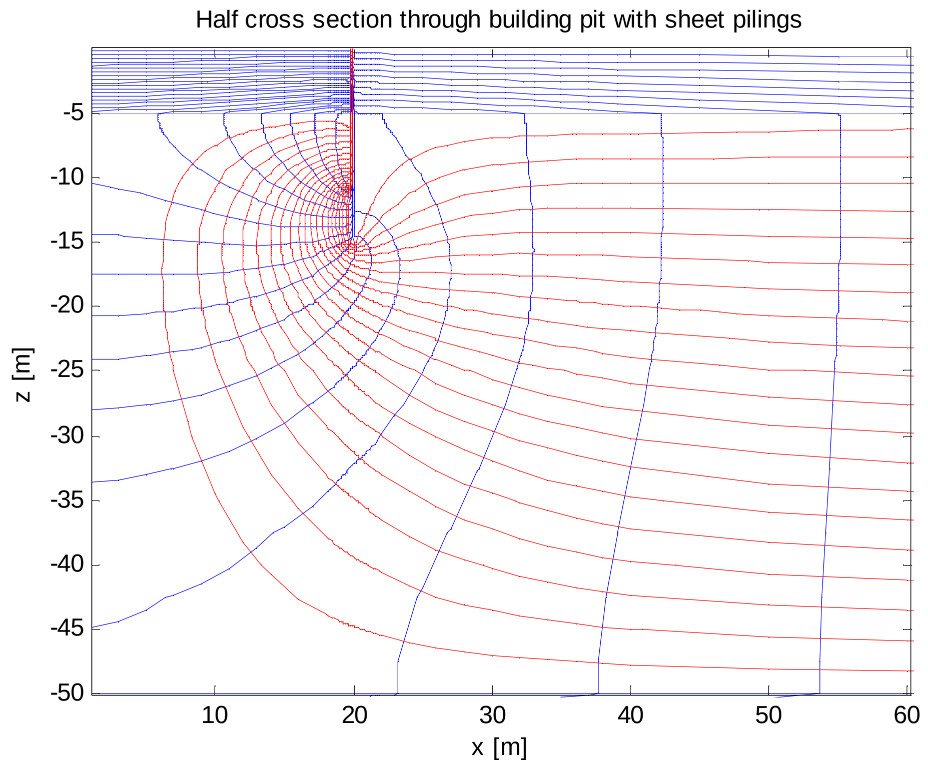


Figure 11: Same as above but for radial symmetric flow.

The extraction is now 5250 m^3/d (not m^2/d !) with the fixed head in the wells -6.7 m to reach a drawdown of 5 m in the center of the pit.

Extractions in a cross section look like wells, because the streamlines to the extraction all connect with the top of the model. These are so-called branch cuts and are unavoidable, as the stream function is multi-valued in the case of extractions or injections within the domain. This is, in fact, is nice for cross sections.

To make wells sharp, narrow the exaction column such that you only see a line of a column of the width of the borehole of the well.

3 Add the stream lines to the 5-layer cross section of your pumping test

This gives a good view on the origin of the water and the paths it takes toward the well.

7 Make a 5 layer vertical semi-confined cross section and show the heads in all layers if layer 4 is pumped

A multi-layer semi-confined model in a cross section is readily made with the Matlab model. The grid rows now represent layers. The conductivity of the layers determines if they represent (work as) aquifers or aquicludes. The head in the top layer is fixed. At other locations in the aquifers fixed heads or flows may be specified. This may also be done for the boundaries of the layers.

4 Color your cross section according to the conductivities before contouring this will yield a publication-ready picture.

Using *surface(x,y,kx)* the conductivities are colored and thus visualizes the structure of the model and make for instance the layers in a cross section clearly visible. To remove the grid lines between the cells use *shading('flat')*.

To overlay this with the contours, we may need an extra axis to draw them and place this axis on top of the *surface* to show them together.

First set up an axis with the correct scales, copy this axis in the same figure. Then draw the conductivity surface in the first axis. Remove the axis, so that only the surface is visible. Switch to the second axis, draw the contours and make the canvas transparent to show the underlying surface.

```
f1=figure;

a1=axes;

set(a1,'xlim',[x(1),x(end)],'ylim',[min(y),max(y)]);

xlabel('x in m'); ylabel('z in m'); title('cross section);

a2=copyobject(a1,f1);

axes(a1);

surface(x,y,kx); shading('flat'); set(gca,'visible','off');

axes(a2);

contour(xm,ym,Phi);

hold on;

contour(x(2:end-1),y,Psi(Qx));
```

5 Radial symmetric finite difference model

An example of the results of a radial symmetric model has already been shown above. Very often we have to deal with radial symmetric flows, for instance to wells. Therefore, it comes in handy to have also a radial symmetric model that is extremely accurate, more accurate than computing radial symmetric flow with the previous model by multiplying the *kx* with the distance to the left size of the model:

$$Kx = 2 * pi * xm * kx;$$

$$Ky = 2 * pi * xm * ky;$$

Using a flat model this way to compute a radial symmetric flow is course a possibility and a good exercise to compare it with a truly a radial symmetric model developed hereafter by converting our flat model into a radial symmetric one, or rather one that can serve both flat and radial symmetric flow problems.

However, in order to convert our model into a radial symmetric one we have to alter its conductances. But in doing so we are not going to destroy the flat model that we already developed; instead the model is going to work for both radial symmetric and flat cases. Keeping both situations in a single Matlab function reduces maintenance in the future.

To make the model work for radial symmetric situations, the only thing to do is compute the resistance between adjacent nodes.

We know that for radial symmetric horizontal flow between two radii the logarithmic analytical solution is valid, from which the resistance against horizontal radial flow is readily derived:

$$\phi_1 - \phi_2 = \frac{Q}{2\pi kD}\ln\left(\frac{r_2}{r_1}\right) \rightarrow Q = \frac{2\pi kD}{\ln\left(r_2/r_1\right)}\left(\phi_1 - \phi_2\right) \rightarrow R_r = \frac{\phi_1 - \phi_2}{Q} = \frac{\ln\left(r_2/r_1\right)}{2\pi kD}$$

Therefore, the resistance between two adjacent nodes becomes

$$R_{r,i,i+1} = \frac{\ln\left(r_{i+1}/r_{m,i}\right)}{2\pi k_i Dy} + \frac{\ln\left(r_{m,i+1}/r_{i+1}\right)}{2\pi k_{i+1} Dy} \rightarrow C_{r,i,i+1} = 1./R_{r,i,i+1}$$

The vertical resistance for entire cells equal

$$Rz = \frac{Dy}{\pi\left(r_{i+1}{}^2 - r_i{}^2\right)k_z} \rightarrow Rz = 0.5\left(Rz\left(1:end-1,:\right) + Rz\left(2:end,:\right)\right)$$

Or in Matlab:

```
RX=(1./dy)*log(x(2:end-1)./xm(1:end-1))./(2*pi*kx(:,1:end-1))+...
   (1./dy)*log(xm(2:end)./x(2:end-1)) ./(2*pi*kx(:,2:end));
RY=0.5/pi*dy*(1./(x(2:end).^2-x(1:end-1).^2))./ky;
Cx=1./RX;
Cy=1./(RY(1:end-1,:)+RY(2:end,:));
```

To add this to our model without destroying what we already have, implement it inside the following if statement

```
If nargin==7
   RX=(1./dy)*log(x(2:end-1)./xm(1:end-1))./(2*pi*kx(:,1:end-1))+...
      (1./dy)*log(xm(2:end)./x(2:end-1)) ./(2*pi*kx(:,2:end));
   RY=0.5/pi*dy*(1./(x(2:end).^2-x(1:end-1).^2))./ky;
   Cx=1./RX;
   Cy=1./(RY(1:end-1,:)+RY(2:end,:));
else
   RX=0.5*(1./dy)*dx./kx; Cx=1./(RX(:,1:end-1)+RX(:,2:end));
   RY=0.5*dy*(1./dx)./ky; Cy=1./(RY(1:end-1,:)+RY(2:end,:));
end
```

nargin is the number of input argument of a Matlab function that is always known within the function. Therefore if the function/model is called with 7 arguments instead of with the ordinary 6, it uses the conductances that are valid for radial symmetric flow and otherwise those for a flat model.

To let a function call

[Phi,Q,Qx]=fdm2(x,y,kx,ky,FH,FQ)

compute the radial symmetric solution, add an arbitrary seventh dummy argument, for instance the string ‘radial’)

[Phi,Q,Qx]=fdm2(x,y,kx,ky,FH,FQ,’radial’)

And it will do so.

Clearly, when setting up radial symmetric models you will often use r instead of x and z instead of y etc. In such a script you are likely to see a call like this

[Phi,Q,Qr]=fdm2(r,z,kr,kz,FH,FQ,’radial’)

But for the function that is called this makes no difference.

The model is now ready to compute both flat and radial symmetric groundwater flow cases and is, therefore, quite flexible.

5.1 Exercise radial symmetric model

Show that the model is correct using analytical solutions. Plot the head and the flow contours (stream function)

1 --- compare with Thiem

Thiem is confined radial symmetric flow with fixed-head boundary at distance *R*. The analytical solution for the drawdown *s* is

$$s=\frac{Q}{2\pi kD}\ln\left(\frac{R}{r}\right)$$

2 --- compare with De Glee

De Glee’s radial symmetric stead-state flow to a fully penetrating well in a semi-confined aquifer has the analytical solution:

$s = \frac{Q}{2\pi kD} K_o\left(\frac{r}{\lambda}\right)$ with $\lambda = \sqrt{kDc}$ and $K_o(...)$ the well-known Bessel function of second kind and zero order.

We may compute this flow with the model in radial mode and compare with the analytical solution

```
Qo=-2400; kD=500; c=350; lambda=sqrt(kD*c);

x=logspace(0,4,41);  y=[0,-1,-2]';

xm=0.5*(x(1:end-1)+x(2:end)); dx=diff(x);      Nx=length(dx);
ym=0.5*(y(1:end-1)+y(2:end)); dy=abs(diff(y)); Ny=length(dy);

kx=[0.5*dy(1)/c; kD/dy(2)]*ones(size(xm));   ky=kx;

FH=NaN*ones(Ny,Nx); FH(end,end)=0.0;

FQ=zeros(Ny,Nx); FQ(end,1)=Qo;

[Phi,Q,Qx]=fdm2(x,y,kx,ky,FH,FQ,'radial');

close all
fi=Qo/(2*pi*kx(end,1)*dy(end))*log(xm(end)./xm);
plot(xm,Phi(end,:),'r',xm,fi,'b+');
legend('analytic','numeric',4);
```

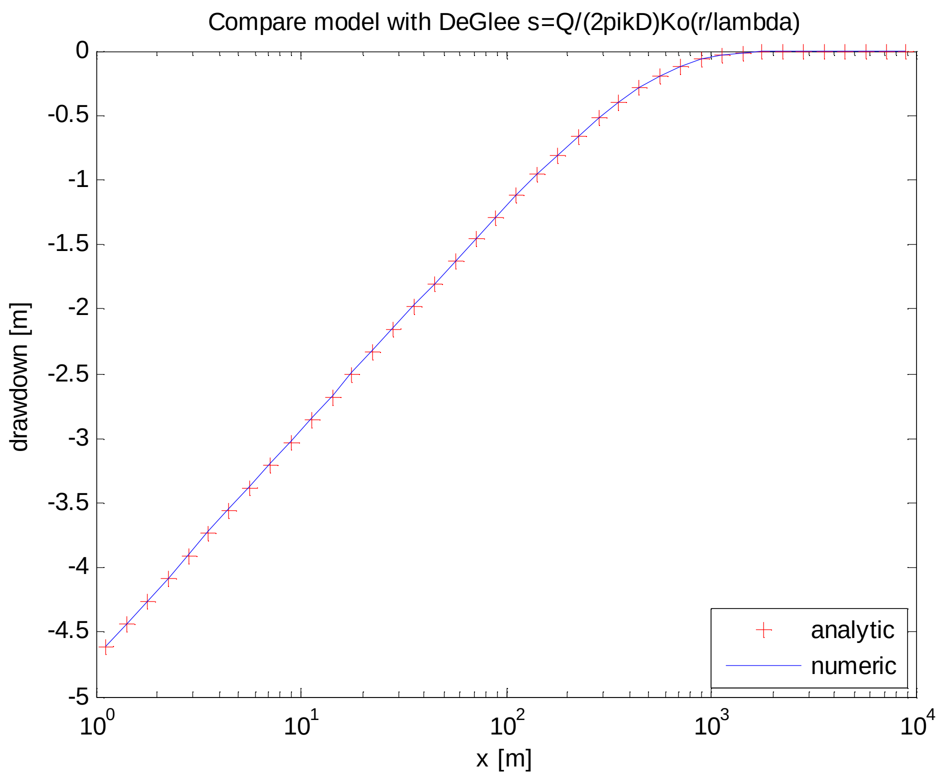


Figure 12: Comparing the model in radial mode with the semi-confined flow solution for steadhy state extraction from a well (De Glee).

3 --- Compare vertical anisotropy

This is straightforward

4 --- Compare with a circular island with recharge

The analytical solution has already been given. In the radial symmetric model the recharge in the top of the columns of the cross section is

FQ(1,:)=pi*(r(2:end).^2-r(1:end-1).^2)*n;

5 --- Compare with a circular island in semi-confined aquifer

In the semi-confined aquifer, the top is an aquitard with a fixed head above it. In the Matlab model, we may use the aquitard as the top layer. But then the fixed head is in the center of this layer. The resistance of the aquitard must than be generated by the half thickness of the top layer (between the node and the bottom of the cell). If the resistance of the aquitard is *c*, and the thickness of the top layer is *H*, then the vertical conductivity in this top layer must be set to $k_z = 0.5H/c$.

We may also use an extra layer on top of the aquifer, make it very thin and specify the head in this thin top layer. In that case the conductivity of the top layer must be set to $k_z = H/c$.

This is the only thing necessary to model a semi-confined aquifer with the radial symmetric model.

6 --- Compute pumping test in layer 4 of 5 layer model

This is trivial with the model.

7 --- Compute effect of partial penetration

As stated before partial screen penetration of the aquifer is the rule rather than the exception when installing wells.

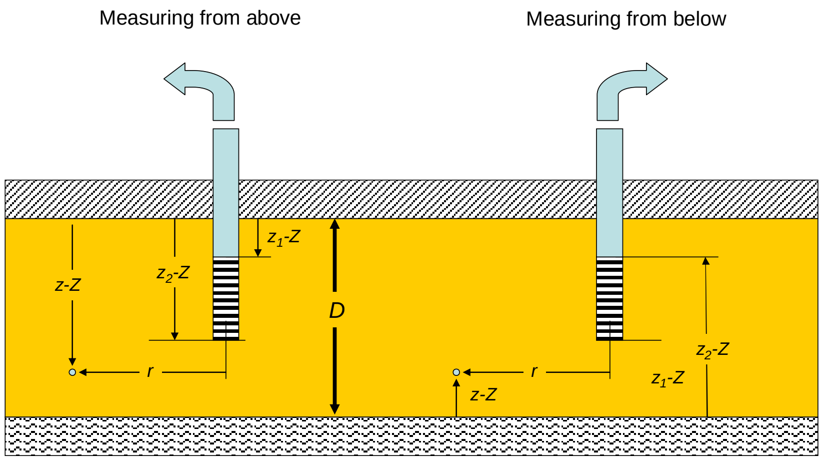


Figure 13: Partial penetrating wells with variables used in the formula

In Matlab one may compute both situations and subtract the two to get the extra drawdown and then compare it with the analytical formula.

The extra drawdown caused by partial penetration has been derived in the past and is given in several books on hydrogeology or pumping test analysis (e.g. Kruzeman & De Ridder, 1997):

$$\Delta s = \frac{Q}{2\pi kD}\frac{2D}{\pi\, d}\sum_{n=1}^{\infty}\left\{\frac{1}{n}\left[\sin\left(\frac{n\pi\left(z_1 - Z\right)}{D}\right) - \sin\left(\frac{n\pi\left(z_2 - Z\right)}{D}\right)\right]\cos\left(\frac{n\pi\left(z - Z\right)}{D}\right)K_o\left(\frac{n\pi\, r}{D}\right)\right\}$$

This drawdown is relative (has to be added to) the drawdown for fully penetrating wells. It was derived for a uniform extraction along the well screens in a homogeneous aquifer.

This formula is valid for uniform extraction along the screen. This is readily implemented as the boundary condition for the well. In the real case, the boundary is rather a fixed head along the screen. This too is readily modeled with the Matlab model. The drawdown along the screen will than vary.

To check the model with respect to partial penetration, compute the drawdown with a fully and with partially penetrating well. Subtract the two drawdown matrices. This difference, which is also a matrix of the size of the model, can be compared with the analytical solution.

```
figure
ds=0; Z=0; D=sum(dy(2:end));  % ds is partial penetration
for i=1:50  % analytical solution partial penetration
   ds=ds+1/i*(sin(i*pi*(ZS(1)-Z)/D)-sin(i*pi*(ZS(2)-Z)/D))...
       .*cos(i*pi.*(ym*ones(size(xm))-Z)/D)...
       .*besselk(0,i*pi.*ones(size(dy))*xm/D);
```

```
end
ds=ds*Qo/(2*pi*kD)*(2*D/(pi*(ZS(1)-ZS(2))));
contour(xm,ym,ds)
title('dspp penetration contours');
set(gca,'xscale','log')
```

5.2 Houskeeping with function modelsize(x,y)

To ease initializing the model, a small householding function may be applied like

```
[x,y,xm,ym,dx,dy,Nx,Ny]=modelsize(x,y)
```

It makes sure *x* and *y* are in the right order, sorted and contain no duplicate values, so that the coodinates may be given in any order. It then computes the centers of the cells, *xm* and *ym*, and given the size of the model *Ny*, Nx, i.e. the number of rows and the number of columns of the model. Using this simple function avoids clutter in your scripts.

```
function [x,y,xm,ym,dx,dy,Nx,Ny]=modelsize(x,y)
%[xm,ym,dx,dy,Nx.Ny]=modelsize(x,y)
% compute size of model and put x and y in correct order
x=unique(x(:)');
y=sort(unique(y(:)),'descend');  % first row is highest coordinated
xm=0.5*(x(1:end-1)+x(2:end));
ym=0.5*(y(1:end-1)+y(2:end));
dx=diff(x);
dy=abs(diff(y));
Nx=length(dx);
Ny=length(dy);
```

6 Transient modeling

With the previously developed models, all ingredients are already in place. The transient water balance during a given time step of length Δt reads

$$\sum a_{ij}\phi_j + C_i\phi_i = Q_i + C_i h_i - C_{S,i}\frac{\phi_i^+ - \phi_i^-}{\Delta t} \tag{2}$$

This is the same as before, a complete model including its boundaries, but now with the storage added to the balance. The left part is the flow outward of node *i*; the right-hand side says where this water comes from: injection into the node, a fixed head boundary and a release of storage over the considered time step. ϕ_i^+ is the nodal head at the end of the time step and ϕ_i^- is the head at the beginning of the time step. This head represents the initial condition necessary in all transient modeling. It is either the initial head at the start of the model or the head at the end of the previous time step. In any case, it is always known during the simulation.

All other heads and the flows have to be average values for the duration of the current time step and are still unknown.

C_S contains the storativity and the cell dimensions, and will be considered further down. The computation of this coefficient is specific to the numerical method, in our case finite differences.

Here we encounter two unknowns, ϕ_i and ϕ_i^+. We will only be able to resolve this situation if we assume some relation between the two. For instance that the head change during the time step is linear and that the average heads are those at time given by some value $t = t^- + \theta\Delta t$ where $0 \le \theta \le 1$, so that

$$\phi - \phi^- = \theta\left(\phi^+ - \phi^-\right) \rightarrow \phi^+ - \phi^- = \frac{\phi - \phi^-}{\theta} \tag{3}$$

and therefore,

$$\sum a_{ij}\phi_j + C_i\phi_i = Q_i + C_i h_i - \frac{C_{S,i}}{\theta\Delta t}\left(\phi_i - \phi_i^-\right) \tag{4}$$

Exactly like we did with the general fixed heads, we leave the fixed part at the right-hand side and put the variable part to the left hand side

$$\sum a_{ij}\phi_j + C_i\phi_i + \frac{C_{S,i}}{\theta\Delta t}\phi_i = Q_i + C_i h_i + \frac{C_{S,i}}{\theta\Delta t}\phi_i^- \tag{5}$$

The left-hand side is equivalent to adding C_i and $C_{S,i}/\left(\theta\Delta t\right)$ to the diagonal matrix coefficient. The right-hand side is equivalent to a permanent inflow into the node during this time step. In Matlab/matrix formulation

$$\left(\mathbf{A} + diag\left(C + \frac{\mathbf{C}_S}{\theta\Delta t}\right)\right) * \Phi = \mathbf{Q} + \mathbf{C}.*\mathbf{h} + \frac{1}{\theta\Delta t}\mathbf{C}_S.*\Phi^- \tag{6}$$

<u>This represents the complete transient model, including its initial and boundary conditions.</u>

Hence, to solve this model in Matlab for the time step:

$$\Phi = \left(\mathbf{A} + diag\left(C + \frac{\mathbf{C}_S}{\theta \Delta t} \right) \right) \backslash \left(\mathbf{Q} + \mathbf{C}.*\mathbf{h} + \frac{1}{\theta \Delta t} \mathbf{C}_S.*\Phi^- \right) \tag{7}$$

This yields average heads during the time step (based on the chosen value of θ). The head at the end of the time step requires a separate computation step:

$$\Phi^+ = \Phi^- + \frac{1}{\theta}\left(\Phi - \Phi^-\right) = \frac{1}{\theta}\Phi + \left(1 - \frac{1}{\theta}\right)\Phi^- \tag{8}$$

Then, by setting $\Phi^- = \Phi^+$ we enter into the next time step, with the heads at the end of the previous time step are the initial heads of the next one.

The value of θ is called the implicitness of the solution. $\theta = 0$ is called explicit and $\theta = 1$ is called fully implicit. Values above 0.5 yield stable solutions (without artificial oscillations). $\theta = 0$ requires small time steps in order to prevent oscillation. On the other hand computation steps are cheap because it does not require any solution of a system matrix. A value of 1 is called fully implicit. It may be less accurate in case of larger time steps, but it is rock-stable. Notice that MODFLOW just uses $\theta = 1$ without any choice for the user. An optimal value for finite element models seems $\theta = 2/3$. Anyway, all values above $\theta = 0.5$ yield unconditionally stable solutions. In practice, it may be most simple to use $\theta = 1$, which implies that the average flows and heads during the time step are well represented by those at the end of the time step. Given the success of MODFLOW there seems to be no real objection against $\theta = 1$. $\theta = 1$ makes the second step to update the heads at the end of the time step obsolete because it reduces to

$$\Phi^+ = \Phi^- \tag{9}$$

The only thing to be elaborated are the values of C_s. For the flat finite difference model these equal

$$C_S = S_S \Delta x \Delta y \Delta z \tag{10}$$

Where S_y is specific yield (water table storage) and S_s is the specific storage, which requires the thickness of the model cell to be given.

As can be seen, each cell is given both a specific yield (in case it has or gets a free water table) and an elastic storage for the saturated part. In our simple models we will not deal with variable aquifer or layer thickness during the simulation, although this is quite straightforward to implement.

For the radial symmetric model the storage coefficients equal

$$C_S = \pi\left(r^2_{i+1} - r^2_{i}\right)\Delta y S_s \tag{11}$$

In practice, S_y will be specified for the top cells with a free water table and Ss for all deeper cells.

What has to be changed to the model to make it transient?

The function call has to be extended with time, storage coefficients and initial heads, while θ may be specified or just set to a default value. We just keep θ as an internal parameter of the model. Here is the transient model.

```
function [Phi,Qt,Qx,Qy,Qs]=fdm2t(x,y,t,kx,ky,S,IH,FH,FQ,radial)
% function [Phi,Q,Qx,Qy,Qs]=fdm2(x,y,t,kx,ky,S,IH,FH,FQ,radial)
% 2D block-centered transient finite difference model
% IH=initial head [L]
% FH=fixed heads (NaN for ordinary points) [L]
% FQ=fixed nodal flows  [L3/T] constant in this model
% Phi [L3] output heads 3D matrix, all time steps    [Ny,Nx,Nt]
% Qt [L3/T] to adjacent nodes during time step =-Storage+inflow
% Qx [L3/T] is hor  cell face flow time step average [Ny,Nx,Nt-2]
% Qy [L3/T] is vert cell face flow time step average [Ny,Nx,Nt-1]
% Qs [L3] is nodal storage change, time step total!! [Ny,Nx,Nt-1]
% TO 991017  TO 000530 001026 070414 070426

theta=1;  % implicitness

x=x(:)'; Nx=length(x)-1; dx=diff(x); xm=0.5*(x(1:end-1)+x(2:end));
y=y(:);  Ny=length(y)-1; dy=abs(diff(y));
t=t(:);  Nt=length(t)-1; dt=diff(t);

Nodes = reshape(1:Nx*Ny,Ny,Nx);               % Node numbering
IE=Nodes(:,2:end);   IW=Nodes(:,1:end-1);
IS=Nodes(2:end,:);   IN=Nodes(1:end-1,:);

% resistances and conductances
If nargin==10
   RX=(1./dy)*log(x(2:end-1)./xm(1:end-1))./(2*pi*kx(:,1:end-1))+...
      (1./dy)*log(xm(2:end)./x(2:end-1)) ./(2*pi*kx(:,2:end));
   RY=0.5/pi*dy*(1./(x(2:end).^2-x(1:end-1).^2))./ky;
   Cx=1./RX;
   Cy=1./(RY(1:end-1,:)+RY(2:end,:));
   Cs=pi*dy*(x(2:end).^2-x(1:end-1).^2).*S;
else
   RX=0.5*(1./dy)*dx./kx; Cx=1./(RX(:,1:end-1)+RX(:,2:end));
   RY=0.5*dy*(1./dx)./ky; Cy=1./(RY(1:end-1,:)+RY(2:end,:));
   Cs=dy*dx.*S;
end

A=sparse([IE(:);IW(:);IN(:);IS(:)],...
         [IW(:);IE(:);IS(:);IN(:)],...
         -[Cx(:);Cx(:);Cy(:);Cy(:)],...
         Ny*Nx,Ny*Nx,5*Ny*Nx);           % System matrix
Adiag= -sum(A,2);                        % Main diagonal

C=zeros(size(FH)); C(~isnan(FH))=1e10;  % fixed heads using huge number
FH(isnan(FH))=0;

Phi=NaN*zeros(Ny,Nx,Nt);       % storage for head matrix
Qt =NaN*zeros(Ny,Nx,Nt-1);     % storage nodal flow matrix
Qx =NaN*zeros(Ny,Nx-1,Nt-1);   % storage hor  face flows
Qy =NaN*zeros(Ny-1,Nx,Nt-1);   % storage vert face flows
Qs =NaN*zeros(Ny,Nx,Nt-1);     % storage for head matrix
Phi(:,:,1)=IH;  Store in initial head as Phi at t=0

for it=1:length(dt)
  Fi=spdiags(Adiag+C(:)+Cs(:)/(dt(it)*theta),0,A)\...
    (FQ(:)+C(:).*FH(:)+reshape(Cs.*Phi(:,:,it)/dt(it)/theta,Ny*Nx,1));

  Phi(:,:,it+1)=reshape(Fi,Ny,Nx)/theta-(1-theta)/theta*Phi(:,:,it);
  Qt (:,:,it)  =reshape(spdiags(Adiag,0,A)*Fi,Ny,Nx);
  Qx (:,:,it)  =-Cx.*diff(reshape(Fi,Ny,Nx),1,2)*sign(x(end)-x(1));  % m3/d
  Qy (:,:,it)  =-Cy.*diff(reshape(Fi,Ny,Nx),1,1)*sign(y(end)-y(1));  % m3/d
  Qs (:,:,it)  =-Cs.*(Phi(:,:,it+1)-Phi(:,:,it))/dt(it);
end
```

The water balance can be checked as follows

$$\mathbf{Q}t = \sum_{j=1,\, j\neq i}^{N} C_{ij}\left(\phi_i - \phi_j\right) = \mathbf{Q}_{fixed} + \mathbf{Q}_{trough\ fixed\ head} + \mathbf{Q}_{storage} \qquad (12)$$

$\mathbf{Q}_t$ is thus the inflow to a node averaged over the time step. This is the flow towards its adjacent connected nodes. It balances with the given fixed inflows from the outside world, $\mathbf{Q}_{fixed}$, the inflow from the outside world through any fixed head nodes and the release of storage (see model code).

$$\mathbf{Q}_{Storage} = -\mathbf{C}_S\left(\Phi^+ - \Phi^-\right)/dt \qquad (13)$$

The ouput flows are all averages during for each time step. That is also true for the $\mathbf{Q}_{Storage}$. It is computed as the release, see equation and model code. The total release from storage for a node over the entire time step thus equals

$$Q_s dt = -C_s\left(\phi_i^+ - \phi_i^-\right)dt$$

To check this water balance, for a model with no fixed heads and only fixed flows,

```
dt=diff(t);
St=zeros(length(dt)); % Vector to store total storage per time step
for it=1:length(dt)
    St(it)=sum(sum(Qs(:,:,it)))*dt(it); % =FQ+QfromFH+QStoreRelease
end
sum(St(:))        % show total storage release over entire period
sum(FQ(:)*sum(dt) % Qw is well flow show total inflow from fixed flows
```

These two show the equal.

In the case of fixed head cells we may compute the fixed head cell inflows from

$$\mathbf{Q}_{FH}(:) = \mathbf{Q}_t(:) - \mathbf{Q}_{fixed}(:) - \mathbf{Q}_{storage}(:)$$

6.1 Exercises transient model

Prove that the model is correct

0 --- Check the water balance

To check the water balance, the storage must be included. Check for your self which flows must add up to zero

1 --- Compare the model with Theis's solution

Theis's solution is for a fully penetrating well in a confined aquifer. The well-known solution for the drawdown is

$$s = \frac{Q_o}{4\pi kD} E_0\left(\frac{r^2 S}{4kDt}\right)$$

With E_o the exponential integral or Theis's well function. In Matlab, for time t(i)

S=Qo/(2*pi*kD).*expint(r.^2.*S./(4*kD*t(i)));

```
% Compute radial transient model compare with Theis analytical solution

rW=0.1; k=10; Ss=0.0001; Sy=0.1; Qw=-2400; t=logspace(-3,2,51);
r=logspace(log10(rW),4,41);
z=[0, -20];
[r,z,rm,zm,dr,dz,Nr,Nz]=modelsize(r,z);   % small household function
K=k*ones(Nz,Nr);
S=Ss*ones(Nz,Nr); S(1,:)=Sy/dz(1);   % combine Ss and Sy (spec yield)
FH=NaN*zeros(Nz,Nr);    % fixed head not necessary for transient flow
IH=zeros(Nz,Nr);        % initial heads are always essential
FQ=zeros(Nz,Nr); FQ(1,1)=Qw;  % extraction at r=rm(1)

kD=k*sum(dz); SY=sum(S(:,1).*dz); % kD and S for analytical comp
fi=Qw/(4*pi*kD)*expint((1./(4*kD*t'))*(rm.^2*SY));   % analytical

[Phi,Qt,Qr,Qz,Qs]=fdm2t(r,z,t,K,K,S,IH,FH,FQ,'radial'); %run model

close all; figure; hold on   % start visualisation
for it=2:length(t)
   plot(rm,Phi(1,:,it),'x'); % numerical as crosses
end
plot(rm,fi);                 % analytical as function of rm, lines
title('Theis drawdown as function of r for different times');
xlabel('r [m]'); ylabel('dd [m]');
set(gca,'xscale','log');     % use log scale

figure; hold on
for ir=1:length(rm)
   plot(t,squeeze(Phi(1,ir,:)),'x-'); % must use squeeze if 1 layer
end
plot(t,fi');                % analytical as function of time, lines
title('Theis drawdown as function of t for different times');
xlabel('t [d]'); ylabel('dd [m]');
set(gca,'xscale','log'); % use log scale
```

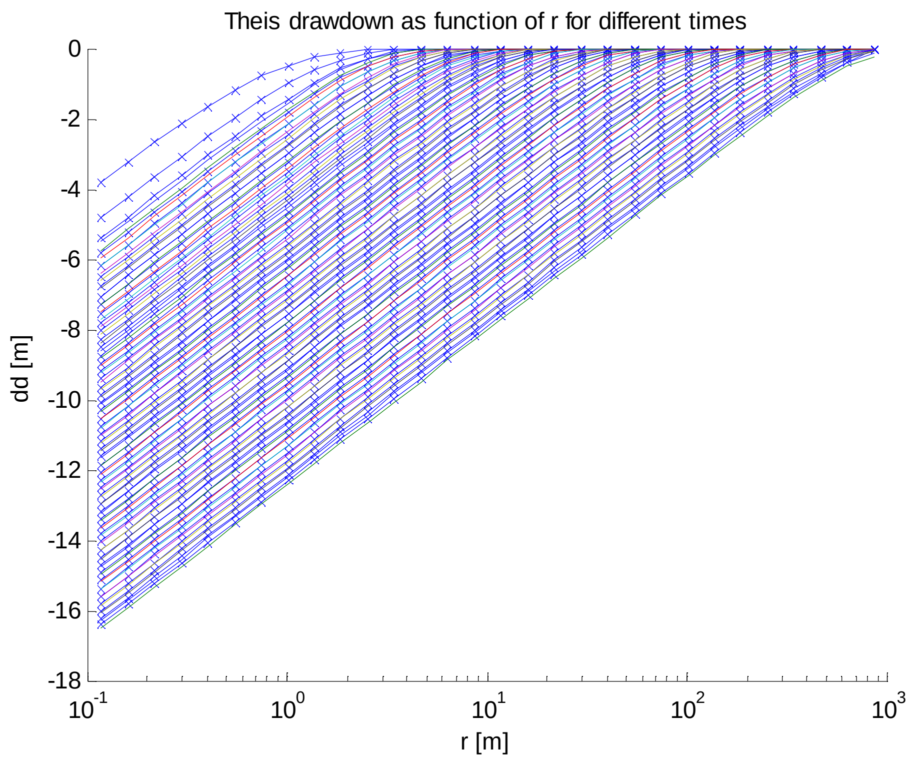


Figure 14: Drawdown as function of distance to the well for different times (x = numerical, lines are analytical solution according to Theis

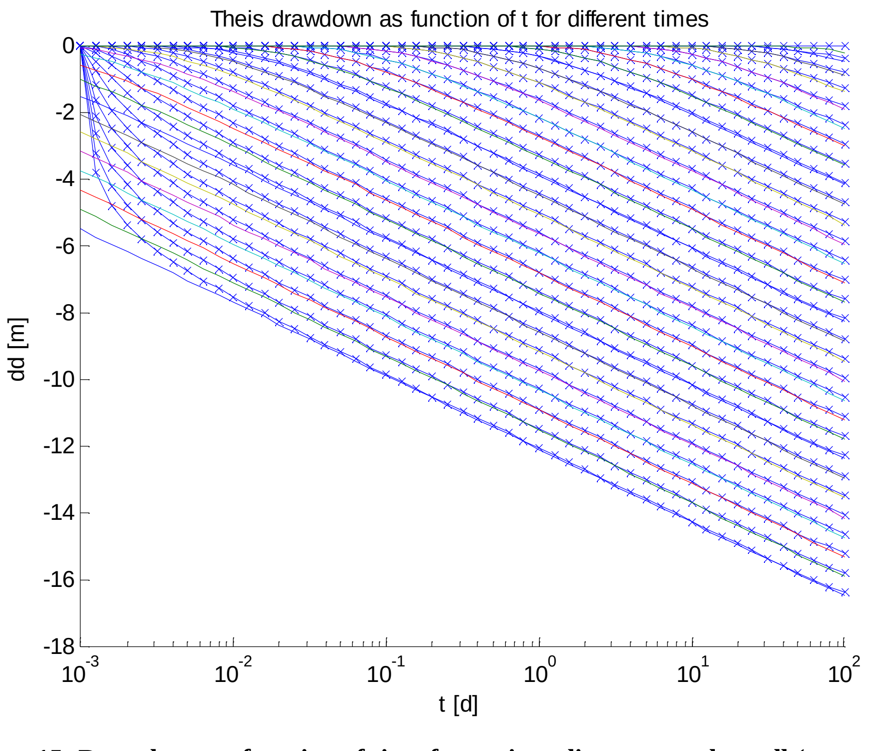


Figure 15: Drawdown as function of time for various distances to the well (numeric X, line analytical Theis solution)

The pictures show the accuracy of the model. The model seems to be off for very small times only. This is because the model starts with zero initial heads at a very non-zero time, while the analytical solution is only zero at zero time. This difference can be completely removed by using the analytical solution at the initial time as initial heads.

2 --- Compare the model with Hantush's solution

Hantush's solution concerns the drawdown due to a well in a semi-confined aquifer:

$$\mathrm{W}_h\left(u,\frac{r}{\lambda}\right)=\int_u^{\infty}\frac{1}{y}\exp\left(-y-\frac{1}{4y}\left(\frac{r}{\lambda}\right)^2\right)dy$$

It may be implemented by writing a function that carries out the integration

$$s=\frac{Q_o}{4\pi kD}W_h\left(u,\frac{r}{\lambda}\right),\quad u=\frac{r^2S}{4kDt}$$

Where $W_h(\ldots,\ldots)$ us Hantush's well function

$$W_h=\int_u^{\infty}\frac{e^{-y-(r/2\lambda)^2}}{y}dy$$

3 --- Compute delayed yield

Delayed yield may result from the drawdown above the aquitard that is caused by the leaking through the aquitard. It also results from the combination of elastic storage and water table storage in the same unconfined aquifer. Initially the drawdown is due to elastic storage, which expands fast. Slowly the water table will determine the drawdown and will show up at a later time. The combined drawdown curve shows two theis-curves in series, the first one determined by the elastic storage, the second one by the water table storage. In Matlab is it readily modeled by giving all cells a small elastic storage coefficient and only the top layer cells a larger one, the specific yield. Compute the time-drawdown curve and compare it with the two Theis curves

4 --- Compute well bore storage (Boulton)

The storage inside the well changes the drawdown shortly after the start of the pump. It may be implemented by modeling the well casing explicitly. A thin column may be given a zero horizontal conductivity to represent the impervious well casing. Then the top cell inside the casing is given a storage coefficient equal to 1. To represent the free water inside the screen and the casing, use a large vertical conductivity. The extraction may then be from any of the cells inside the screen or the casing. The large vertical conductivity inside the well makes sure the head is the same throughout the well screen and casing. The result should be compared with the analytical solution given by Boulton. A practical manner is comparing it with curves for Boulton in Pumping Test Books (e.g. Kruzeman & De Ridder, 1970, 1995)

NATWARGHAD – Dorpelingen uit Natwarghad, in de westelijke Indiase deelstaat Gujarat, verdringen zich rond een enorme waterput. Grote delen van India worden al enkele weken getroffen door droogte. Bij temperaturen die oplopen tot 47 graden en droge wind zijn de natuurlijke waterlopen en bronnen opgedroogd en kost het steeds meer moeite aan drinkwater te komen. Al bijna 900 mensen zijn gestorven als gevolg van de hittegolf. 'Water' is een van de thema's die worden besproken op de G8 in Evian. (Foto Reuters)

Figure 16: A large diameter well

Consider a large 5 m radius dug well that is also 5 m deep in a 20 m deep aquifer. The specific yield is 5% and the extraction 130 m3/d, i.e. 1000 people using 70 l/d plus 600 cattle using 100 l/d. What will be the drawdown in this well? Is it sustainable?

To analyze this situation, make an radial symmetric model, 20 m deep. Use a logarithmically increasing grid size with distance (say logspace(-1,4,41), such that the drawdown will not reach the outer boundary of the model) and say 20 layers of 1 m thickness vertically. Then refine around the diameter of the well and around its bottom, to accurately compute the concentrated flow in this region.

In the well use a very high conductivity, day k=10000, so that the well will obtain a uniform head like in the reality. The extraction may be put in an arbitrary model point inside this well. Then apply the storage coefficient to the model cells. All cells may be given the specific elastic storage coefficient by default. However, specific yield is different. It applies to the topmost cells only and we must use it there as a kind of elastic storage for the top row of cells. To do this, use S_y/dz for this row as storage coefficient. That is, do this for all top row cells and use $1/dz$ (i.e. S_y=1) for the cells representing the inside of the well.

```
rW=5; zW=-5; R=10000; k=1; Ss=0.0001; Sy=0.05; Qw=-130;
r=[logspace(0,log10(R),50), rW+[-0.5 -0.25 -0.1 0 0.1 0.25 0.5 1]];
z=[0:-1:-20, zW+[-0.5 -0.25 -0.1 0 0.1 0.25 0.5]];
t=logspace(0,3,31);
[r,z,rm,zm,dr,dz,Nr,Nz]=modelsize(r,z);

K=k*ones(Nz,Nr); K(zm>zW, rm<rW)=10000;
S=Ss*ones(Nz,Nr); S(1,:)=Sy/dz(1); S(1,rm<rW)=1/dz(1);
```

```
FH=NaN*zeros(Nz,Nr);
IH=zeros(Nz,Nr);
FQ=zeros(Nz,Nr); FQ(1,1)=Qw;
[Phi,Qt,Qr,Qz,Qs]=fdm2t(r,z,t,K,K,S,IH,FH,FQ,'radial');

kD=k*sum(dz); SY=sum(S(:,end).*dz);
fi=Qw/(4*pi*kD)*expint((1./(4*kD*t(:)))*(rm.^2*SY));

close all
plot(rm,squeeze(Phi(1,:,:)),'+',rm,fi,'-'); hold on
title('Drawdown insize pp large diameter well, compare with Theis');
xlabel('r [m]'); ylabel('dd [m]');
set(gca,'xscale','log');

figure
plot(t,squeeze(Phi([1,end],1,:)),'+',t,fi(:,1),'-');
set(gca,'xscale','log');
legend('in well','below well, bottom aquifer','Theis');
title('Drawdown in partially penetrating large diameter well, compare with Theis');
xlabel('t [d]'); ylabel('dd [m]');
set(gca,'xscale','log');
```

Example of a large diameter well

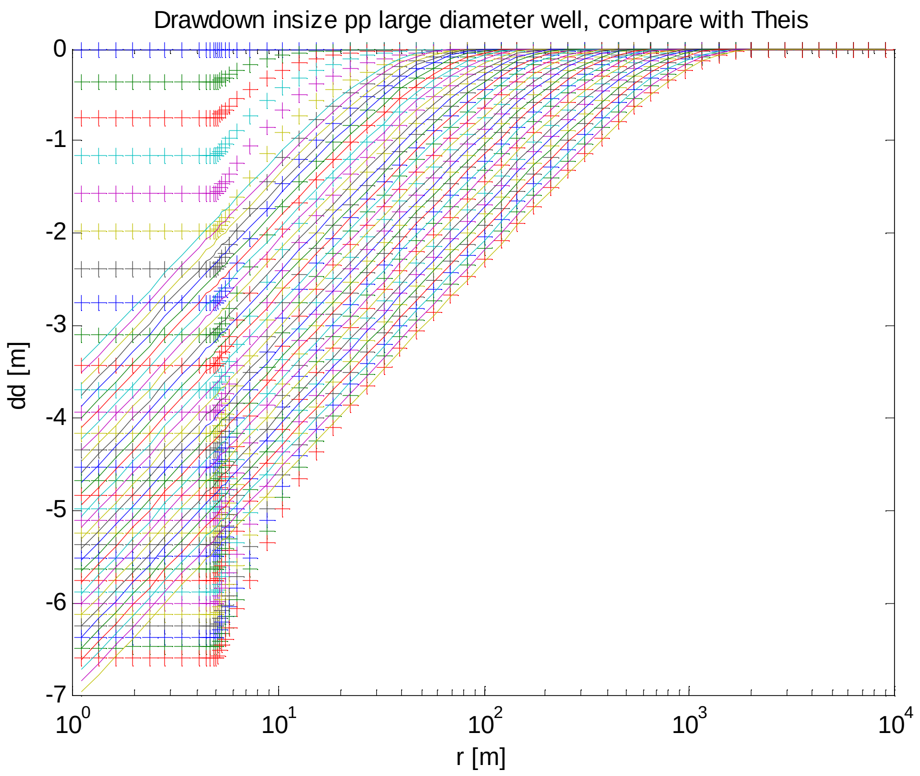


Figure 17: Drawdown (numeric +) along z=0 through well and at top of aquifer. Comparison with Theis solution (lines). The horizontal lines is the head inside the well (5 m radius)

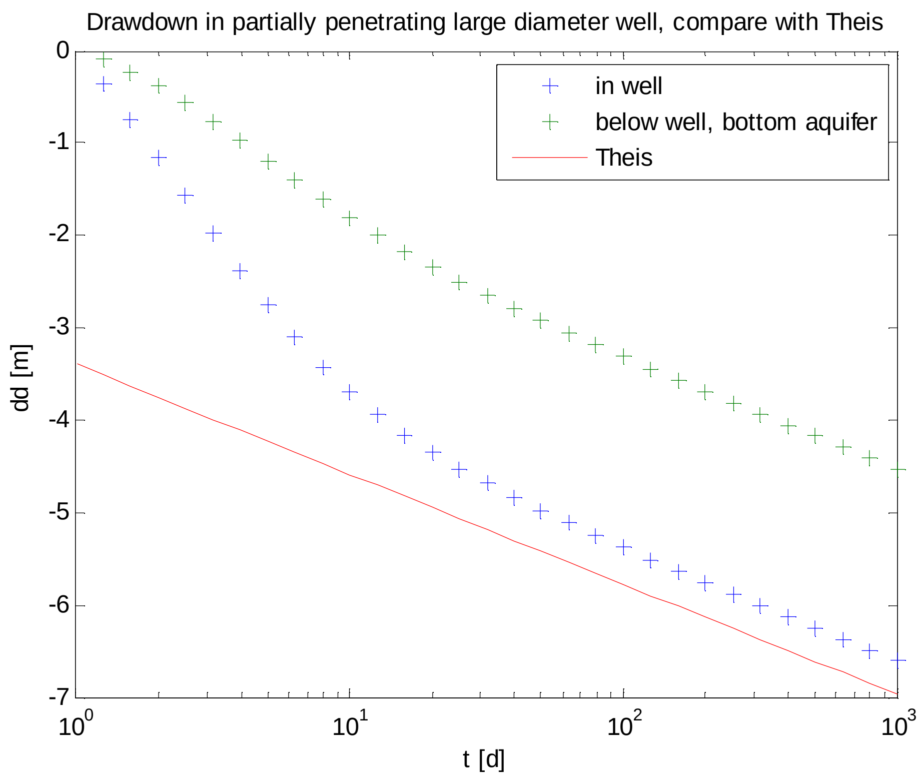


Figure 18: Drawdown inside large diameter well, below it at the bottom of the aquifer and comparison with Theis solution (this drawdown is quite substantial). The drawdown inside and below the well is less than Theis, because the large diameter compensates the partial penetration. The drawdown at the bottom of the aquifer is much less than inside the well due to partial penetration. Initially the drawdown in the well is less than Theis and declines more or less linearly due to the large storage inside it.

To check the water balance, see if the total extraction from the well over the entire period matches the water released from storage

```
dt=diff(t);
St=zeros(length(dt));
for it=1:length(dt)
    St(it)=sum(sum(Qs(:,:,it)))*dt(it);
end
FromStorage=sum(St(:))
Injected   =Qw*sum(dt)
```

Matlab gives:

FromStorage = 1.2987e+005

Injected =   -129870

These are indeed the same and equal the total extraction. Now check with the given well extraction (-130 m^3/d * 999 days)

>> Qw*sum(dt)

```
ans =  -129870
```

Which indeed matches the given infiltration (extraction = negative infiltration)

To visualize the flow, compute the steady-state model

```
FH(:,end)=0;   % now we must have some boundary fixed
[Phi,Qn,Qx]=fdm2(r,z,K,K,FH,FQ,'radial'); % steady state model
contour(rm,zm,Phi); hold on          % head lines
contour(r(2:end-1),z,Psi(Qx));       % stream lines
set(gca,'xscale','log');             % log scale
title('large diameter well'); xlabel('r [m]'); ylabel('z [m]');
```

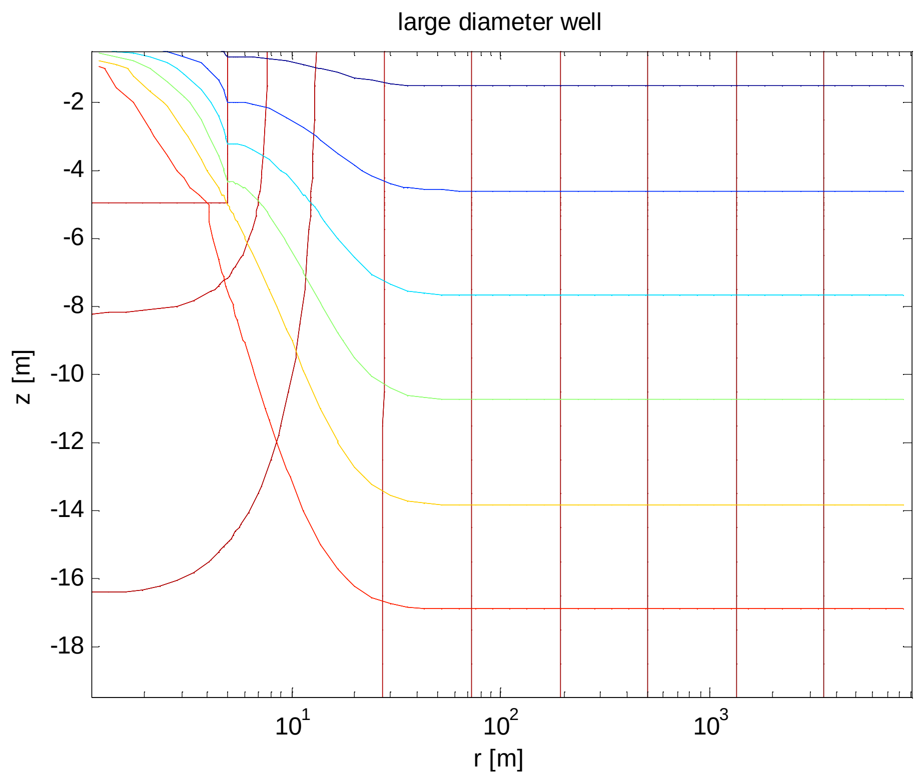


Figure 19: The contours of the steady-state computation with fixed head at R=10000. Inside the wells the stream lines continue to the point of extraction. Notice the logarithmic scale used to visualize the situation

5 --- Compute the effect of a shower of rain on a parcel of land compare with analytical solution

A shower of rain on a parcel of land cause the water to be raised instantaneously, but the head at the edges of the parcel remains equal to the ditch level. This comes down to an immediate drawdown at the edges of the parcel, which progresses into the parcel, initially fast becoming slower and slower over time.

To model this in Matlab, set the fixed head in the ditches equal to zero and use an initial head equal to n/S_y, where n is the shower in mm and S_y the specific yield. Then follow the drawdown over time. Used increasing time steps to track the fast initial drawdown well.

t=logspace(-3,3,61) % logarithmically increasing series starting at10^{-3}, ending at 10^3 in 60 steps (61 values, 10 per log cycle).

Compare the results with the analytical solution

$$s=\frac{n}{S_y}\left(1-\frac{4}{\pi}\sum_{j=1}^{\infty}\left(\frac{(-1)^{j-1}}{2j-1}\cos\left((2j-1)\pi\frac{x}{L}\right)\exp\left(-(2j-1)^2\pi^2\frac{kD}{L^2S}t\right)\right)\right)$$

Where L is the half-width of the cross section through the parcel (Carslaw & Jaeger, 1959, p97, eq 8; Verruijt, 1999, p87).

7 Model wrapping

Having a transient model, the steady-state version can be created by running the transient model with the proper input. This would make the maintenance of a separate steady –state version of the model obsolete. On the other hand, a transient model requires more input and is more complicated to use. A good compromise may then be to make a wrapper around it, that looks like a steady-state model, but all it does is augmenting the input with dummies and then call the transient model with its complete input. The use then only bothers with the wrapper as if it were a steady-state version of the model. Working with wrappers may easy model usage and at the same time reduce maintenance as only a single model has be updated. This is especially important as models get more complicated.

```
function [Phi,Q,Qx,Qy]=fdm2wrap(x,y,kx,ky,FH,FQ,radial)
% function [Phi,Q,Psiy,Psix]=fdm2(x,y,kx,ky,FH,FQ,radial)
% 2D block-centred steady-state finite difference model
% x,y mesh coordinates, kx,ky conductivities
% FH=fixed heads (NaN for ordinary points), Q=fixed nodal flows
% Phi,FQ computed heads and cell balances
% Qx is horizontal flow direction increasing column number
% Qy is vertial cell wall flow in directin of increasing row number
% TO 991017  TO 000530 001026 070414

Dummy=zeros(size(FQ)); tDummy=[0 1];
[Phi,Qt,Qx,Qy]=fdm2t(x,y,tDummy,kx,ky,Dummy,Dummy,FH,FQ,radial);

Phi=squeeze(Phi(:,:,end));
Q  =squeeze(Qt);
Qx =squeeze(Qx);
Qy =squeeze(Qy);
% squeeze eliminates one dimension if it has length 1: 3D→2D
```

8 Particle tracking (flow lines)

Particle tracking is one of the functions most used in a groundwater model. Contrary to stream lines that require steady-state 2D flow without sources and sinks, particles may always be tracked to create flow lines. Clearly, particles starting at the same location may not follow the same path if released at different times in a transient model. In the random walk technique particles are even given a random displacement at each time step to simulate dispersion, which alters the path of individual particles in an unforeseen manner, thus simulating dispersion.

Particle tracking in finite difference models is quite straightforward. The flows perpendicular to the cell faces are known and, therefore, the specific discharge at theses faces may be approximated by dividing by their surface area. Average. As the porosity in the cells at either side of a cell face may differ, so may the groundwater velocity perpendicular to the cell face, even though the specific discharge does not.

In finite difference modeling, the flow in *x*, *y* and *z* direction, which is parallel to the axes of the model, is linearly interpolated between that at opposite cell faces. This implies that the flow in *x*-direction (and velocity for that matter) is only a function of *x*, the velocity in *y*-direction only a function of *y* and the one in *z*-direction only depends on *z*. This is consistent with the model assumptions and largely simplifies the analysis. However, for large cells it may not be accurate. So it may be necessary to use smaller cells where large variations in velocity occur in value and direction. On the other hand the elegance of this approach is that the divergence remains zero in a cell. This means no water is lost, so that the flow paths by themselves are consistent.

To show this, take the divergence for an arbitrary point within a 3D cell without sources and sinks. This divergence must be zero:

$$\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_x}{\partial z}=0 \qquad (14)$$

Written out in the flows generated by the model for a cell with size $\Delta x,\ \Delta y,\ \Delta z$ gives

$$\frac{Q_{x2}-Q_{x1}}{\Delta x\Delta y\Delta z}+\frac{Q_{y2}-Q_{y1}}{\Delta x\Delta y\Delta z}+\frac{Q_{z2}-Q_{z1}}{\Delta x\Delta y\Delta z}=0\rightarrow Q_{x2}-Q_{x1}+Q_{y2}-Q_{y1}+Q_{z2}-Q_{z1}=0 \qquad (15)$$

which must be true because it is the cell’s water balance (without internal sources).

To analyze particle tracking within de realm of FDM further, consider the average velocity at cell faces, computed from the flows perpendicular to the cell faces (Q_x and Q_y) and the porosity ε of the cell. In 2D, the velocity in a cell with porosity ε may be computed for the local flow in *x* and *y* direction. For generality we also have to consider the thickness of the cell, *D*, perpendicular to the *x*,*y* plane.

$$v_x=\frac{1}{\varepsilon H}\frac{Q_x}{\Delta y};\ v_y=\frac{1}{\varepsilon H}\frac{Q_y}{\Delta x} \qquad (16)$$

Within a cell the flow is interpolated between that of the opposite cell faces

$$Q_x = Q_{x-} + \left(x - x_-\right)\frac{Q_{x+} - Q_{x-}}{x_+ - x_-}, \;\; Q_y = Q_{y-} + \left(y - y_-\right)\frac{Q_{y+} - Q_{y-}}{y_+ - y_-} \tag{17}$$

$$\begin{aligned} v_x &= \frac{Q_x}{\varepsilon H \Delta y} = \frac{Q_{x-}}{\varepsilon H \Delta y} + \left(x - x_-\right)\frac{Q_{x+} - Q_{x-}}{\varepsilon H \Delta x \Delta y} = \frac{\Delta x Q_{x-}}{\varepsilon V} + \left(x - x_-\right)\frac{Q_{x+} - Q_{x-}}{\varepsilon V} \\ v_y &= \frac{Q_y}{\varepsilon H \Delta x} = \frac{Q_{y-}}{\varepsilon H \Delta x} + \left(y - y_-\right)\frac{Q_{y+} - Q_{y-}}{\varepsilon H \Delta x \Delta y} = \frac{\Delta y Q_{y-}}{\varepsilon V} + \left(y - y_-\right)\frac{Q_{y+} - Q_{y-}}{\varepsilon V} \end{aligned} \tag{18}$$

In which the indices + and – refer to the sides of this cell.

Hence $\Delta x = x_+ - x_-$, $\Delta y = y_+ - y_-$

By dividing by $\varepsilon H \Delta y$ and $\varepsilon H \Delta x$ within the cell we obtain the groundwater velocities

$$\begin{aligned} v_x &= \frac{Q_x}{\varepsilon H \Delta y} = \frac{Q_{x-}}{\varepsilon H \Delta y} + \left(x - x_-\right)\frac{Q_{x+} - Q_{x-}}{\varepsilon H \Delta x \Delta y} \\ v_y &= \frac{Q_y}{\varepsilon H \Delta x} = \frac{Q_{y-}}{\varepsilon H \Delta x} + \left(y - y_-\right)\frac{Q_{y+} - Q_{y-}}{\varepsilon H \Delta x \Delta y} \end{aligned} \tag{19}$$

Or,

$$v_x = v_{x-} + \frac{v_{x+} - v_{x-}}{\Delta x}\left(x - x_-\right); \;\; v_y = v_{y-} + \frac{v_{y+} - v_{y-}}{\Delta y}\left(y - y_-\right) \tag{20}$$

With

$$\alpha_x = \frac{dv_x}{dx} = \frac{v_{x+} - v_{x-}}{dx} \;\; \textbf{and} \;\; \alpha_y = \frac{dv_y}{dy} = \frac{v_{y+} - v_{y-}}{dy} \tag{21}$$

This simplifies to

$$v_x = v_{x,-} + a_x\left(x - x_-\right) \textbf{, and } v_y = v_{y-} + a_x\left(y - y_-\right) \tag{22}$$

Working this out in terms of particle displacement yields

$$\frac{dx}{dt} = v_{x-} + \alpha_x\left(x - x_-\right); \; \frac{dy}{dt} = v_{y-} + \alpha_y\left(y - y_-\right) \tag{23}$$

We leave out the *y*-direction for now to limit the length of this paper.

Integration yields

$$\frac{d\left(v_{x-} + \alpha_x\left(x - x_-\right)\right)}{v_{x-} + \alpha_x\left(x - x_-\right)} = \alpha_x dt \quad \textbf{-}\!\!\rightarrow \ln\left(v_{x-} + \alpha_x\left(x - x_-\right)\right) = \alpha_x t + C \tag{24}$$

With x=x_o at t=t_o C may be computed, giving

$$\ln\left(\frac{v_{x-}+\alpha_x\left(x-x_-\right)}{v_{x-}+\alpha_x\left(x_o-x_-\right)}\right)=\alpha_x\left(t-t_o\right) \tag{25}$$

This equation is useful to compute at what time the particle hits a cell face given its initial position in the cell. Clearly, its velocity must not be zero (denominator), neither must the velocity at the target position be zero (numerator), or must the fraction be negative (which means opposite velocities at current and target positions, implying a water divide in between).

Reversing this formula yields the position of the particle at a given time

$$x=x_-+\left(\frac{v_{x-}}{a_x}+\left(x_o-x_-\right)\right)\exp\left(\alpha_x\left(t-t_o\right)\right)-\frac{v_{x-}}{a_x} \tag{26}$$

Which may be rewritten in relative coordinates

$$u=\frac{x-x_-}{\Delta x}=\left(\frac{v_{x-}}{\Delta v}+u_o\right)\exp\left(\alpha_x\left(t-t_o\right)\right)-\frac{v_{x-}}{\Delta v}\textbf{, with } \Delta v=v_+-v_- \tag{27}$$

This gives the relative position in the cell at given time starting at an arbitrary initial position x_o in the cell at time t_o. For this to work, the only condition is that $\alpha_x \neq 0$ in which case the velocity is constant and the new position becomes.

$$x=x_o+v_x\left(t-t_o\right) \textbf{ or } t-t_o=\frac{x-x_o}{v_x} \tag{28}$$

The model must capture this situation as it needs to use an alternative formula to compute the velocity at another location or the time to get to some other locations (i.e. the cell face).

If

$$\frac{v_{x-}+\alpha_x\left(x-x_-\right)}{v_{x-}+\alpha_x\left(x_o-x_-\right)}\leq 0$$

The logarithm does not exist. The reason is that the velocity at the target location x is opposite to that at the current location x_o so that the particle never reaches the target location. This happens if there is a water divide between the current and target locations. In this case no time can be computed. On the other hand the position of the particle may then be computed for any time up to infinity. In the model these situations must be captured.

The model

The logic of the particle tracking model is as follows:

First compute the velocities at the upstream and downstream sides of all cells, both in x and y direction, using the cell face flows at the cell interfaces and the porosities of this cells.

Then, given an arbitrary point x_p, y_p, find in which cell it is and start tracking during a given time step *DT*

In case the velocity is zero the point remains at its position and the time moves on to *t*+*DT*. Particle position and cell index remain unchanged.

If the velocity is negative, compute movement of particle in the direction of the upstream cell face:

If the velocity is constant, use the appropriate formula. Compute the time *dt* to reach the cell face. If this time exceeds *DT*, use *DT* instead and compute the new particle position using *DT*. Don't update the cell index.

Else check the velocity direction at the target cell face and see if the particle will ever reach it. If so, compute the time *dt* until the hit. Provisionally update the particle position to this cell face and reduce the cell index for the x-direction by 1. Now check if *dt*>*DT* to see if the target *DT* is reached before we hit the cell face. If this is the case, use *DT* instead and compute the particle position using *DT*. Don't update the cell index.

If the particle will never reach the upstream cell face, use *DT* and compute the particle position after *DT*. Don't update the cell index.

The same logic is used for the downstream cell face in the case the velocity is positive.

The same logic also applies for the y-direction.

Finally we have a provisional new position *xpN*, *ypN* with provisional change of cell indix *dic*, *djc* (both -1, 0 or +1) in *x* and *y* direction respectively and two times dt_x and dt_y that meet the criteria in both the *x* and *y* direction respectively.

The smallest of the two determines the final particle update. In case this is dt_x, than the values *xp=xpN*, *ic=ic+dic* and *dt=dtx* will hold and the y-position of the particle has to be recomputed using the new *dt*. In case the smallest of the two is dt_y it is the other way around.

Clearly, *dt* may be smaller than the initial target time step *DT*, as a cell face is hit much sooner. Then *DT* is reduced by *dt* and the procedure is done all over again, causing the particle to move through the next cell. This is repeated until *DT* has become zero. This makes sure that the particle position at given time points will be stored together with the positions and times that a particle crosses cell faces.

The procedure is repeated with a new time step, until all have been worked through.

Because the cell face flows at the outer boundary faces of the model are always zero in the finite difference model, particles can never escape the model and need no special care in that respect. However, particles will enter extraction cells, where they would simply slow down indefinitely as, because such cells behave as having a water divide inside (or a distributed extraction over the cell area). Therefore, it is better to capture particles entering cells that have an extraction which is beyond a given fraction of the throughflow of the cell. This is the same approach as MODPATH.

The extraction is provided by the output of fdm2 and the throughflow is computed as the sum of the absolute values of the flows across all 4 cell faces. This threshold fraction may be set at 15% or so. So the loop is broken off as soon as the particle enters a cell being a sink according to this criterion.

To allow dealing with the *x* and *y* (and possibly the *z*-axis) in the same manner, so that the program uses the same logic for the three axes, one must guarantee that the cell indices are aligned (increase) with the positive axis-direction. This is checked in the beginning and if necessary the concerned matrices are flipped accordingly left-right or upside-down.

To allow backward tracing, the matrices *Qx* and *Qy* are multiplied by -1. Backward tracing is signaled by using negative times in the input.

The implementation is such that the function *fdmpath* is called after the model has been run and the necessary nodal and cell-face flow matrices computed. The extra information that the *fdmpath* needs is the porosity of all cells and either the thickness of each cell or the sign that the model be computed in radial fashion

```
[XP YP TP]=fdmpath(x,y,DZ|radial,Q,Qx,Qy,por,T,[markers])
```

Use the x,y,Q,Qx and Qy matrices that are the output of *fdm2*.

Make sure the size of the porosity matrix equals the size of Q or use a scalar.

Make sure that the absolute values of the time series T are increasing. Use negative values for backward tracing. You don't need to start with a zero first time.

The third argument is either a matrix *DZ* of the cell thicknesses or a string such as 'R' or 'radial' to indicate the radial symmetric case. You may use a scalar for *DZ*. An empty matrix [] will be regarded the same as *DZ*=1.

In the case of radial flow the horizontal and vertical velocities at the cell faces are computed as

$$v_x = \frac{Q_x}{\varepsilon \Delta y\, 2\pi x} \quad \textbf{and} \quad v_y = \frac{Q_y}{\varepsilon\, \pi \left(x_+{}^2 - x_-{}^2 \right)} \qquad \textbf{(29)}$$

The optional markers is a string consisting of letters that are valid Matlab marker indicators. The default is (see doc marker).

```
'+o*.xsdph^v<>'
```

There will be a marker plotted in the paths for each given time (except zero) according to this list of markers, which is repeated of there are more times than markers given. For instance `'oooop'` gives you four 'o-markers' and each fifth `a 'p'=pentagon`.

When the model is run, it picks up the current figure (assuming it is a contour plot generated after running the model) by letting you click at a point in it (left mouse button). The program will immediately compute and show the flow path with the markers (No markers will be visible if the particles leave the model before the first time is reached).

You can repeat this as long as you like.

To stop use the right-hand mouse button. Upon this the program stops and yields the XP, YP and TP coordinates of all the generated lines. These points and times include all points where particles crossed cell borders and all points at the given times. The individual lines in these matrices are separated by a NaN ("Not a Number" value). The coordinates at the given times can be picked out of these matrices:

```
for it=1:length(T)
      I=find(abs(TP-T(it))<1e-6);
      plot(XP(I),YP(I),'o');
end
```

As with all Matlab functions, you can always get help by typing

help *fdmpath*

fdmpath has a self test built in. It will be run if you type

fdmpath

Finally, it is a good exercise to work this out for transient flow. In that case the flows are dependent of the time which requires some extra housekeeping. It is also a good exercise to work this out for 3D, possibly 3D and transient. This is not very difficult, but the convenience will be much less due to more difficult visualization and data handling. If you really need to do a complicated 3D transient modeling project, rather use standard software with an advanced user interface with easy entry and management of the input, output and visualization. Having said all this, the current modeling provided in this syllabus comprise a practical, powerful and efficient modeling toolbox with many uses for practical real-life groundwater modeling.

8.1 Checking the particle tracking

To check the particle tracking use some convenient analytical solutions

A cross section, thickness *H*, porosity ε and recharge *n*, with a water divide at *x*=0 center obeys the following relations

$$\begin{aligned} &v_x = \frac{dx}{dt} = \frac{nx}{\varepsilon H} \rightarrow \frac{dx}{x} = \frac{n}{\varepsilon H} dt \rightarrow \ln(x) = \frac{n}{\varepsilon H} t + C \quad (with\ t = t_o,\ x = x_o) \\ &\ln(x_o) = \frac{n}{\varepsilon H} t_o + C \rightarrow C = \ln(x_o) - \frac{n}{\varepsilon H} t_o \\ &\ln\left(\frac{x}{x_o}\right) = \frac{n}{\varepsilon H}(t - t_o) \rightarrow x = x_o \exp\left(\frac{n}{\varepsilon H}(t - t_o)\right) \end{aligned} \tag{30}$$

This can be used to check the travel time in the model in two directions.

Another simple check is a well in a confined aquifer. Here we have

$$Qt = \varepsilon H \pi R^2 \rightarrow R = \sqrt{\frac{Qt}{\pi \varepsilon H}} \tag{31}$$

So set up a model, run it, contour the results, run *fdmpath*, and check its results by clicking a point near the well

```
%% radial flow to a well, check fdmpath:
clear all; close all;
Qo=2400; por=0.35
x=logspace(-1,4,51); y=[0 -1 -2]; % need 2 y layers to contour
[x,y,xm,ym,dx,dy,Nx,Ny]=modelsize(x,y);   % housekeeping
k=10*ones(Ny,Nx);                         % conductivity
FH=NaN*ones(Ny,Nx); FH(:,end)=0;          % fixed head boundary
FQ=zeros(Ny,Nx); FQ(:,1)=Qo.*dy./sum(dy); % divide Q over the 2 layers
[Phi,Q,Qx,Qy]=fdm2(x,y,k,k,FH,FQ);        % run model fdm2
contour(xm,ym,Phi,30); set(gca,'xlim',[0 3000]);  % contour results
T=3650;                                 % time series (one point only)
[XP YP TP]=fdmpath(x,y,'radial',Q,Qx,Qy,por,T,'o');  % run fdmpath
R=sqrt(Qo*T(end)/(pi*por*sum(dy)))    % check this yields 1996 m
```

8.2 Listing of fdmpath

```
function [XP YP TP]=fdmpath(x,y,DZ,Q,Qx,Qy,por,T,markers)
% [XP YP TP]=fdmpath(x,y,DZ,Q,Qx,Qy,por,T,markers,[radial])
% 2D particle tracking.
% To use: generate a 2D steady state model, launch this file and on the
% pictures click for a starting point. The line will be immediately drawn
% Repeat this for more lines. Click te right hand button to stop
% type fdmpath for selftest and demo
% x y mesh coordinates
% DZ thickness of the cells (if empty, 1 is used, matrix not needed)
% if DZ is a character string radial flow is assumed so use 'radial' or so
% for DZ in case a radially symmetric flow is desired
% Q Qx Qy output of fdm2 (steady state only)
% por matrix of porosities (scalar is enough)
% T time points where markers is desired, 0 not necessary, will be added
% use negatigve times values to trace backward in time
% markers is a series of markers for the consecutive times
% e.g.   '>+o*.xsdph^v<'
% XP YP TP coordintates of flow paths, there is a [NaN NaN NaN] between
% consecutive tracks.
% TO 070424 070501

if nargin==0; selftest; return; end

if nargin<9
    markers='+o*.xsdph^v<>';
end
Lm=length(markers);

Nx=length(x)-1; Ny=length(y)-1;

if isempty(DZ), DZ=ones(Ny,Nx);  elseif isscalar(DZ),  DZ =DZ *ones(Ny,Nx); end
if isscalar(por),                                      por=por*ones(Ny,Nx); end

%first make sure the positive direction of the grid is aligned with the positive x and y
directions
if sign(x(end)-x(1))<0, x=fliplr(x); Q=fliplr(Q); Qx=fliplr(Qx); Qy=fliplr(Qy);
DZ=fliplr(DZ); por=fliplr(por); end
if sign(y(end)-y(1))<0, y=flipud(y); Q=flipud(Q); Qx=flipud(Qx); Qy=flipud(Qy);
DZ=flipud(DZ); por=flipud(por); end

dx=diff(x); dy=diff(y);

% then check which cell are sinks

if T(end)<T(1)  % if times negative then track particles backward in tme
    Qx=-Qx;
    Qy=-Qy;
    T=-T;
end

sinkfrac=0.15;
```

```
Qthrough=zeros(size(Q));
if ~isempty(Qx)
    Qthrough=Qthrough+[zeros(size(Qx(:,1))),abs(Qx)]+[abs(Qx),zeros(size(Qx(:,1)))];
end
if ~isempty(Qy)
    Qthrough=Qthrough+[zeros(size(Qy(1,:)));abs(Qy)]+[abs(Qy);zeros(size(Qy(1,:)))];
end
sink= Q < -sinkfrac*Qthrough;

if ischar(DZ) % then the flow is radial symmetric
    if ~isempty(Qx)
        A=dy*2*pi*x;
        vx2=[Qx, zeros(size(Qx(:,1)))]./(A(:,2:end)  .*por);
        vx1=[zeros(size(Qx(:,1))), Qx]./(A(:,1:end-1).*por);
        ax=(vx2-vx1)./(ones(size(Qx(:,1)))*diff(x));
    end
    if ~isempty(Qy)
        A=ones(size(dy))*(pi*(x(2:end).^2-x(1:end-1).^2));
        vy2=[Qy; zeros(size(Qy(1,:)))]./(A.*por);
        vy1=[zeros(size(Qy(1,:))); Qy]./(A.*por);
        ay=(vy2-vy1)./(diff(y)*ones(size(Qy(1,:))));
    end
else
    if ~isempty(Qx)
        vx2=[Qx, zeros(size(Qx(:,1)))]./((dy*ones(size(dx))).*por.*DZ);
        vx1=[zeros(size(Qx(:,1))), Qx]./((dy*ones(size(dx))).*por.*DZ);
        ax=(vx2-vx1)./(ones(size(Qx(:,1)))*diff(x));
    end
    if ~isempty(Qy)
        vy2=[Qy; zeros(size(Qy(1,:)))]./((ones(size(dy))*dx).*por.*DZ);
        vy1=[zeros(size(Qy(1,:))); Qy]./((ones(size(dy))*dx).*por.*DZ);
        ay=(vy2-vy1)./(diff(y)*ones(size(Qy(1,:))));
    end
end

XP=([]); YP=([]); TP=([]);
while 1
    [Xp, Yp, button]=ginput(1);  if button~=1; break; end % get starting points for
stream lines

    DT=diff(T(:)); if T(1)~=0, DT=[T(1);DT]; end

    for ip=1:length(Xp);
        xp=Xp(ip); yp=Yp(ip); t=T(1);
        XP=[XP;NaN;xp];
        YP=[YP;NaN;yp];
        TP=[TP;NaN; t];
        iLast=length(TP); % to later plot only this  line

        ic=find(x<xp,1,'last');
        jc=find(y<yp,1,'last');

        for idt=1:length(DT);
            dt=DT(idt);
            while dt>0
                if isempty(Qx)
                    dic=0; dtx=dt;
                else

[xpN,dic,dtx]=postime(xp,x(ic),x(ic+1),vx1(jc,ic),vx2(jc,ic),ax(jc,ic),dt);
                end
                if isempty(Qy)
                    djc=0; dty=dt;
                else

[ypN,djc,dty]=postime(yp,y(jc),y(jc+1),vy1(jc,ic),vy2(jc,ic),ay(jc,ic),dt);
                end

                [ddt,i]=min([dtx,dty]);
                switch i
                    case 1
                        if ~isempty(Qy)
                            xp=xpN;
                            yp=pos(yp,y(jc),vy1(jc,ic),ay(jc,ic),ddt);
                        end
                        ic=ic+dic;
                    case 2
                        if ~isempty(Qx)
                            xp=pos(xp,x(ic),vx1(jc,ic),ax(jc,ic),ddt);
                            yp=ypN;
                        end
                        jc=jc+djc;
                end

                dt=dt-ddt; t=t+ddt;
                XP=[XP;xp]; YP=[YP;yp]; TP=[TP;t];
                if length(XP)>20000; break; end

              if dt==0
                  m=mod(idt,Lm); if m==0, m=Lm; end
                  line(xp,yp,'marker',markers(m)); hold on;
              end

              if sink(jc,ic);
                  break;  % from while
              end

            end

            if sink(jc,ic);
              break; % from for
            end
        end
        line(XP(iLast:end),YP(iLast:end),'color','g');
    end
end
XP=[XP;NaN]; YP=[YP;NaN]; TP=[TP;NaN];


function [xp,dic,dt]=postime(xp,x1,x2,v1,v2,ax,Dt)
EPS=1e-6;

v=v1+ax*(xp-x1);
if abs(v)<EPS
    % ic=ic
    dt=Dt;  % immediately jumpt to end of time step
    dic=0;
    return; % x remains same location
end

if v<0  % point moves to face at left side
    if abs(ax)<EPS  % v will be constant
        dt=(x1-xp)/v;
        if dt>Dt
            dt=Dt;
            xp=xp+v*dt;
            dic=0;
        else
            xp=x1;
            dic=-1;
        end
    elseif v1>=0           % point will never reach left face
        dt=Dt;         % immediately jump to end of time step
        xp=pos(xp,x1,v1,ax,dt); % compute position at Dt
        dic=0; % ic=ic
    else
        dt=tim(xp,x1,x1,v1,ax);
        if dt>Dt
            dt=Dt;
            xp=pos(xp,x1,v1,ax,dt);
            dic=0;
        else
            xp=x1;
            dic=-1;
        end
    end
end

if v>0
    if abs(ax)<EPS
        dt=(x2-xp)/v;
        if dt>Dt
            dt=Dt;
            dic=0;
            xp=xp+dt*v;
        else
            xp=x2;
            dic=+1;
        end
    elseif v2<=0
        dt=Dt;
        xp=pos(xp,x1,v1,ax,dt);
        dic=0;
    else
        dt=tim(xp,x2,x1,v1,ax);  % CHECK
        if dt>Dt
            dt=Dt;
            xp=pos(xp,x1,v1,ax,dt);
            dic=0;
        else
            xp=x2;
            dic=+1;
        end
    end
end

function xp=pos(xstart,x1,v1,ax,dt)
EPS=1e-6;
if abs(ax)<EPS
    vx=v1+ax*(xstart-x1);
    xp=xstart+vx*dt;
else
    xp=x1+(v1/ax+(xstart-x1))*exp(ax*dt)-v1/ax;
end

function dt=tim(xstart,xtarget,x1,v1,ax)
    dt=1/ax*log((v1+ax*(xtarget-x1))/(v1+ax*(xstart-x1)));

function selftest
    help fdmpath
    clear all; close all
    y=linspace(-2500,2500,22);
    x=linspace(-2500,2500,22);

    [x,y,xm,ym,dx,dy,Nx,Ny]=modelsize(x,y);

    DZ=50;
    k =10;
    n=0.001;

    kx= ones(Ny,Nx)*k*DZ; ky=kx;
    FH=zeros(Ny,Nx)*NaN; FH(:,[1,end])=0;  FH([1,end],:)=0;
    FQ=n*dy*dx;
    [Phi,Q,Qx,Qy]=fdm2(x,y,kx,ky,FH,FQ);
    contour(xm,ym,Phi); hold on

    %Track particles
    por=0.35; DZ=50;
    t=[60 365 3650 25*365 100*365];
    [XP,YP,TP]=fdmpath(x,y,DZ,Q,Qx,Qy,por,t,'...p...p...p');
```

9 Examples

Some examples have already been given and an unlimited number of other ones can be given. Below one that shows the how detail can be provide and how radial and flat flow can be done easily. It demonstrates the use of layers and the input of wells and walls at a specific location. The detail can be seen in the figure by zooming in near the sheet piling (deep wall) of this building pit (preferably horizontally zoom first until the x an dy scale are about equal and the apply regular zoom. (see zoom options under the right mouse button after selecting a figure in Matlab).

```
%% Circular building pit with partially penetrating wells inside sheet piling
% The flow near the extraction and the tip of the sheet piling is very
% detailed. There is a semi-confined top layer and an aquifer.
% try to zoom in near the sheet piling. (horizontally zoom in first until
% the vertical and horizontal scales are more or less equal, the use
% regular zoom to zoom furhter (zoom options are under the right mouse
% button)
% To change to a regualr (not radial cross section) remove 'radial' form
% the call to fdm2 and replace 'radial'in fdmpath to 1 (i.e. DZ=1, a cross
% section of 1 m thickness). Also change the times for the particle track
% markers, because that will be much reduced compared to radial flow).
clear all; close all
layers={
    'clay' 0 -5  0.02           % material, top, bottom k
    'sand' -5 -50  20
    'clay' -50 -60 0.01
    'sand' -60 -200  30
    };
xW    =[19.9 20  ]; yW    =[ 0 -15]; kW=0.0001;     % dimension and props of sheet piling
xWells=[19.8 19.9]; yWells=[-6 -11]; FHWells=-6.7;  % locaton of wells, their fixed heads

x=[0:2:18, 18:0.2:22 19:0.1:21, 22:2:40, 40:10:100, 100:25:250, 250:50:500,
500:100:1000];
L=[-5 -50 -60 -200];
y=[0 -0.01 L, L+0.01, -5:-0.1:-7, -7:-0.5:-14, -15:-0.1:-16, -16:-0.5:-19.5, -19.5:-0.1:-
20.5, -20.5:-0.5:-25, -25:-5:-50];

% house keeping makes sure that points in vectors x and y are unique and ordered
[x,y,xm,ym,dx,dy,Nx,Ny]=modelsize(x,y);

kx=zeros(Ny,Nx);
for i=1:size(layers,1);
    kx(ym<=layers{i,2}&ym>layers{i,3},:)=layers{i,end};    % layer conductivities
end
kx(ym<yW(1) & ym>yW(2), xm>xW(1) & xm<xW(2))=kW;  ky=kx;   % put sheet piling in k matrix
FH=NaN*ones(Ny,Nx); FH(1,:)=0.0;                           % semi pervious top layer
FH(ym<yWells(1) & ym>yWells(2), xm>xWells(1) & xm<xWells(2))=FHWells; % wells in FH
FQ=zeros(Ny,Nx);                                           % no given flows
%[Phi,Q,Qx,Qy]=fdm2(x,y,kx,kx,FH,FQ,'radial');             % radial computation
[Phi,Q,Qx,Qy]=fdm2(x,y,kx,kx,FH,FQ);                       % flat computation
contour(xm,ym,Phi,-5:0.2:0,'b'); hold on                   % head contours
contour(x(2:end-1),y,Psi(Qx),20,'r');                      % 20 stream lines
for i=1:size(layers,1)                   % plot line above an below each layer
    plot([x(1) x(end)],[layers{i,2},layers{i,2}]);
end

% check water balance
sum(sum(Q(ym<yWells(1) & ym>yWells(2), xm>xWells(1) & xm<xWells(2))))  % extraction
sum(sum(Q(1,:)))                    % infiltration through top of model
sum(sum(Q))                         % overall water balance
Phi(ym<-5 & ym>-6,1)                % head below building pit
title('Half cross section through building pit with sheet pilings');
xlabel('x [m]'); ylabel('z [m]');

%particle tracking
por=0.35;
```

```
t=-365*(0:5:50);
%[XP,YP,TP]=fdmpath(x,y,'radial',Q,Qx,Qy,por,t,'...p');  % radial
[XP,YP,TP]=fdmpath(x,y,1,Q,Qx,Qy,por,t,'...p');          % flat
```

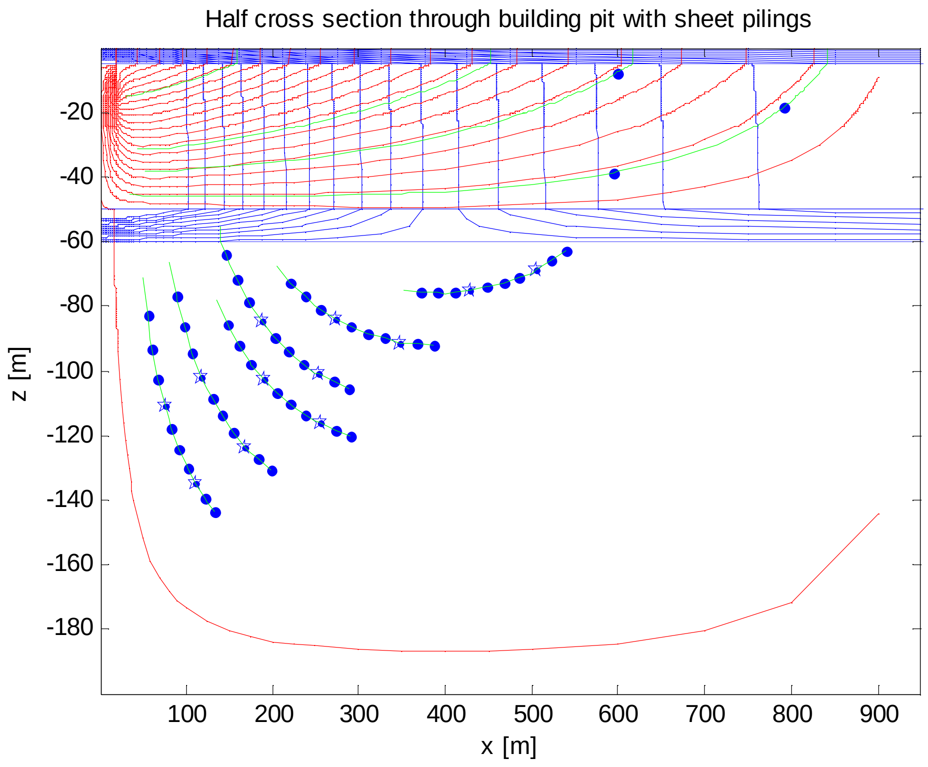


Figure 20: Cross section (flat) with heads, streamlines and some particle tracks, obtained by clicking on the figure when fdmpath is running (backward traces as times were negative, see input above). There is great detail near the sheet piling where all the streamlines converge, which can only be seen by zooming in.

10 Literature

McDonald, M.G. & A.W. Harbaugh (1988) A Modular three-dimensional finite-difference ground-water flow model. Series: Techniques of Water-Resources Investigations on the United States Geological Survey. Book 6, Chapter A1.

The Mathworks (2006) Matlab refernce.