CHAPTER 6

Introduction to Finite Difference Methods for Partial Differential Equations

Numerical Methods in the Hydrological Sciences
Special Publication Series 57
Copyright 2005 by the American Geophysical Union
doi:10.1029/057SP11

6. Introduction to Finite Difference Methods for Partial Differential Equations

6.1. Classification of partial differential equations

There is a "standard" classification of partial differential equations (PDE's). There is nothing sacred about this classification, and there is actually nothing essential in it that we will use. Nevertheless, the terminology is used widely so it pays to have a passing knowledge of the taxonomic jargon. Consider the following to be your introduction to some terminology.

Consider the second-order PDE with constant coefficients A, B, and C. ("D" represents some arbitrary function and is irrelevant to the classification.)

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) = 0.$$

The equation is:

elliptic, if B^2-4AC <0;

parabolic, if B^2-4AC=0;

hyperbolic, if B^2-4AC>0.

Three "general" equations from mechanics that often appear in the hydrological sciences are 1) Laplace's equation, 2) the heat equation, and 3) the wave equation. These equations are, respectively, elliptic, parabolic, and hyperbolic. Consider the definition of the classes above and convince yourself that the classifications are as indicated by completing Table 6.1. Also think about the one-dimensional forms of the Navier-Stokes and the advection dispersion equations listed in Table 6.1. The latter equations are presented using dimensionless variables. Note that the classification isn't particularly informative about these equations unless the second-order terms (the ones involving the Reynolds number, **R**, and Peclet number, **Pe**) dominate. That is, the classification "works" only for low Reynolds numbers and low Peclet numbers. (If you are unfamiliar with these equations or the dimensionless parameters, consult a text on fluid dynamics.)

6.2. Finite difference approximations for derivatives

The method known as the "finite-difference method" uses approximations to the partial derivatives in equations to reduce PDE's to a set of algebraic equations. Various approximations to derivatives are listed in Table 3.1. In particular, recall the approximations to first and second derivatives listed below (cf. equations 3.2 through 3.5).

Table 6.1. *Some equations that arise frequently in hydrological problems.*

EQUATION	A	B	C	B^2-4AC	CLASSIFICATION
Laplace's equation: $\frac{\P^2 u}{\P x^2} + \frac{\P^2 u}{\P y^2} = 0$	1	0	1	-4	Elliptic
Heat equation: $\frac{\P u}{\P t} = \frac{\P^2 u}{\P x^2}$					
Wave equation: $\frac{\P^2 u}{\P t^2} = \frac{\P^2 u}{\P x^2}$					
Navier-Stokes: $\frac{\P\ u}{\P\ T} + u\frac{\P\ u}{\P\ X} = -\frac{1}{\mathbf{F}} - \frac{P_0}{r\ U_0^2}\frac{\P\ p}{\P\ X} + \frac{1}{\mathbf{R}}\frac{\P^2 u}{\P\ X^2}$					
Advection-dispersion: $\frac{\P\ c}{\P\ T} + \frac{\P\ c}{\P\ X} = \frac{1}{\mathbf{Pe}}\frac{\P^2 c}{\P\ X^2}$					

The forward-difference approximation to a first partial derivative with respect to x at point x_i, y_j:

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i, y_j)}{\Delta x} + O(\Delta x)$$
$$= \frac{f_{i+1,j} - f_{i,j}}{\Delta x} + O(\Delta x)$$

The backward-difference approximation to a first derivative:

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i, y_j) - f(x_i - \Delta x, y_j)}{\Delta x} + O(\Delta x)$$
$$= \frac{f_{i,j} - f_{i-1,j}}{\Delta x} + O(\Delta x)$$

The central-difference approximation to a first derivative:

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{\Delta x} + O(\Delta x)^2$$
$$= \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x} + O(\Delta x)^2$$

A central-difference approximation to the second derivative:

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x_i + \Delta x, y_j) - 2\, f(x_i, y_j) + f(x_i - \Delta x, y_j)}{(\Delta x)^2} + O(\Delta x)^2$$

$$= \frac{f_{i+1,j} - 2 f_{i,j} + f_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2$$

Similar equations give the approximations for partial derivatives with respect to *y*.

Substitution of these approximations for the derivatives in a PDE leads to a set of algebraic equations. The numerical solution to the PDE is recovered by solving the algebraic equations.

6.3. Example: The Laplace equation

Steady-state flow of groundwater in a homogeneous, isotropic, constant-thickness, horizontal aquifer is governed by the Laplace equation [Box 6.1]. The equation, in terms of groundwater head, *h*, is

$$\frac{\P\, ^2 h}{\P\, x^2} + \frac{\P\, ^2 h}{\P\, y^2} = 0$$

Consider the Laplace equation applied to a rectangular section of aquifer with Dirichlet boundary conditions (which just means that the heads on the boundaries are specified). Let the aquifer be 400m long in the *x* direction and 200m long in the *y* direction. Apply a grid with Δx=50m and Δy=50m and use *i* and *j* to index the position on the grid superimposed on the domain. The heads on the boundary are fixed at 100m along the right side of the boundary and 0 elsewhere (Figure 6.1).

The substitution of central-difference approximations to the derivatives in the Laplace equation for a general node (i,j) leads to:

$$\frac{h_{i+1,j} - 2h_{ij} + h_{i-1,j}}{(\Delta x)^2} + \frac{h_{i,j+1} - 2h_{ij} + h_{i,j-1}}{(\Delta y)^2} = 0 \qquad (6.1)$$

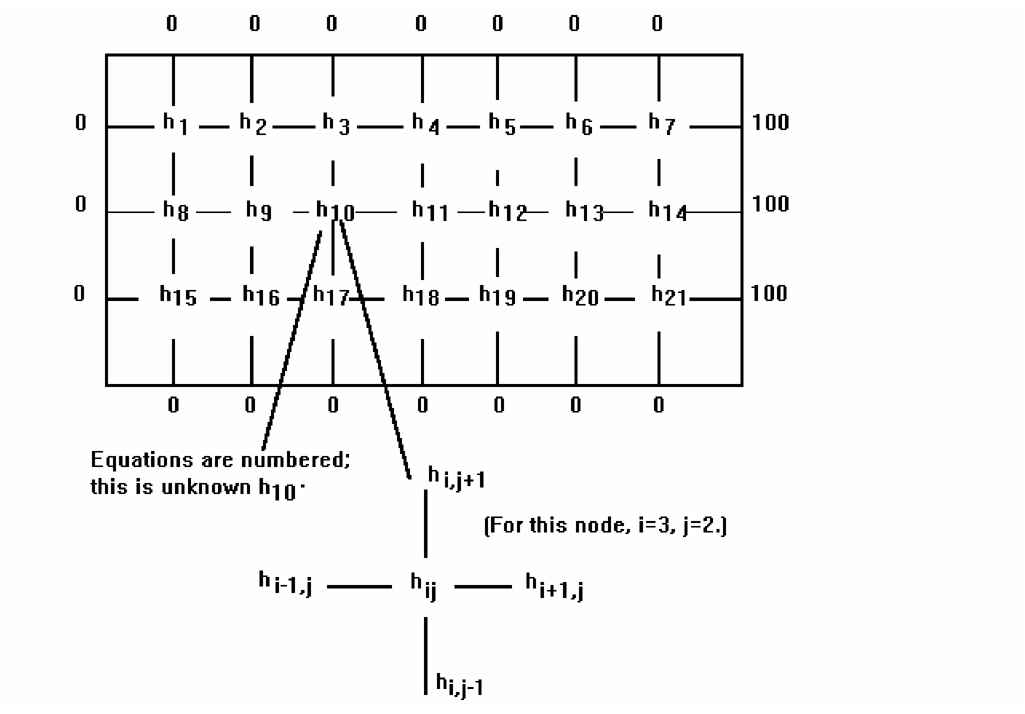

0 0 0 0 0 0 0
0 h1 h2 h3 h4 h5 h6 h7 100
0 h8 h9 h10 h11 h12 h13 h14 100
0 h15 h16 h17 h18 h19 h20 h21 100
0 0 0 0 0 0 0
Equations are numbered; this is unknown h10.
hi,j+1
(For this node, i=3, j=2.)
hi-1,j
hij
hi+1,j
hi,j-1

Figure 6.1. Finite difference grid for the Laplace equation.

Let Δd represent both Dx and Dy, which we specified were equal at 50m. Equation (6.1) can then be written:

$$\left(\frac{1}{(\Delta d)^2}\right)\left(h_{i+1,j} + h_{i-1,j} + h_{i,j+1} + h_{i,j-1} - 4h_{ij}\right) = 0 \qquad (6.2)$$

Now consider the equation for node 1 of Figure 6.1. It is clear from the problem that we have 21 unknowns (7 "i" nodes times 3 "j" nodes). We number the nodes and use a single subscript on h to designate which unknown we are considering (Figure 6.1). Equation (6.2) for node 1 indicates

$$\frac{1}{2500}(h_2 + 0 + h_8 + 0 - 4h_1) = 0. \qquad (6.3)$$

We can write an equation like this for each of the 21 nodes to get a system of 21 equations in 21 unknowns. The first of these equations (for h_1) shows a -4 as the coefficient on the first unknown and ones for the second and eighth unknown. Write the equation for the second node. Your result should show a -4 coefficient for the second unknown and ones for the first, third, and ninth. If we write all of the equations and put them in matrix-vector form, the result is a set of simultaneous linear equations.

Solving the example problem in MATLAB

We have already seen how to solve systems of equations in *MATLAB* (Chapter 2.9). All that must be done is to specify the matrix and right-hand vector. One way that comes immediately to mind is to write an m-file that loops through the 21x21 elements of the matrix. Such a file might look like this. (If you haven't already used all of the *MATLAB* logical operations, "= =" indicates "equal to", "~=" indicates "not equal to".)

```
A=zeros(21,21);         %preallocate space for the matrix
for k=1:21              % 21 rows
      for n=1:21        % 21 columns
            A(k,n)=0;
            if k==n A(k,n)=-4;end
            if k==n-1 & n~=8 & n~=15 A(k,n)=1;end
            if k==n+1 & k~=8 & k~=15 A(k,n)=1;end
            if k==n-7 A(k,n)=1;end
            if k==n+7 A(k,n)=1;end
      end
end  ;
```

The result in *MATLAB* is (for the first 9 rows and columns)

```
A(1:9,1:9)

ans =
    -4     1     0     0     0     0     0     1     0
     1    -4     1     0     0     0     0     0     1
     0     1    -4     1     0     0     0     0     0
     0     0     1    -4     1     0     0     0     0
     0     0     0     1    -4     1     0     0     0
     0     0     0     0     1    -4     1     0     0
     0     0     0     0     0     1    -4     0     0
     1     0     0     0     0     0     0    -4     1
     0     1     0     0     0     0     0     1    -4
```

which is the desired result.

There are very good reasons to avoid this "brute-force" method for setting up finite-difference matrices. First, *MATLAB* is very efficient at dealing with vector operations and not very efficient at dealing with "for loops". (If you do want to perform loop calculations such as those shown above, make sure to preallocate space for the matrix, i.e., to use `A=zeros(21,21)` before starting the loop.) More importantly, the matrices can become large rather quickly. For the simple example above, we had 21 unknowns leading to a matrix with 21 rows and 21 columns, designated a 21x21 matrix. In this case the matrix has 441 entries. But if we want to halve the grid spacing in the example problem, we wind up with 15x7=105 unknowns and our matrix is 105x105, and has 11025 entries. And this isn't a large problem at all. How do we recover from the difficulty of working with large numbers of matrix entries? If we look at the matrix for our example problem, we get a clue. *Most of the entries in the matrix are zero*. Matrices that arise in finite-difference (and finite-element) methods are *sparse**;*** they have many zero elements. If we can take advantage of the sparseness and store only non-zero entries, we save tremendously. *MATLAB* does this through a series of sparse matrix commands as indicated below[1].

```
help sparse
 SPARSE     Build sparse matrix from nonzeros and indices.
      S = SPARSE(...) is the built-in function which generates matrices
      in MATLAB's sparse storage organization.  It can be called with
      1, 2, 3, 5 or 6 arguments.

      S = SPARSE(X) converts a sparse or full matrix to sparse form by
      squeezing out any zero elements.

      S = SPARSE(i,j,s,m,n,nzmax) uses the rows of [i,j,s] to generate
      an m-by-n sparse matrix with space allocated for nzmax nonzeros.
      The two integer index vectors, i and j, and the real or complex
      entries vector, s, all have the same length, nnz, which is the
      number of nonzeros in the resulting sparse matrix S .

      There are several simplifications of this six argument call.

      S = SPARSE(i,j,s,m,n) uses nzmax = length(s).

      S = SPARSE(i,j,s) uses m = max(i) and n = max(j).

      S = SPARSE(m,n) abbreviates SPARSE([],[],[],m,n,0).  This
      generates the ultimate sparse matrix, an m-by-n all zero matrix.

      The argument s and one of the arguments i or j may be scalars,
      in which case they are expanded so that the first three arguments
      all have the same length.

      For example, this dissects and then reassembles a sparse matrix:

                 [i,j,s] = find(S);
                 [m,n] = size(S);
                 S = sparse(i,j,s,m,n);

      So does this, if the last row and column have nonzero entries:

                 [i,j,s] = find(S);
                        S = sparse(i,j,s);

      All of MATLAB's built-in arithmetic, logical and indexing operations
      can be applied to sparse matrices, or to mixtures of sparse and
      full matrices.  Operations on sparse matrices return sparse matrices
      and operations on full matrices return full matrices.  In most cases,
      operations on mixtures of sparse and full matrices return full
      matrices.  The exceptions include situations where the result of
      a mixed operation is structurally sparse, eg.  A .* S is at least
      as sparse as S .  Some operations, such as S >= 0, generate
```

[1] Reprinted with permission from The Mathworks, Inc.

```
      "Big Sparse", or "BS", matrices -- matrices with sparse storage
      organization but few zero elements.

      See also FULL, FIND and the sparfun directory.
```

Let's use the sparse matrix commands to build the matrix for the example problem. First we want to have a 21x21 square matrix with values of –4 on the main diagonal (the entries running from the top left to the bottom right of the matrix).

```
M_diag=sparse(1:21,1:21,-4,21,21);
```

Next we build the subdiagonal, the line of entries directly below the main diagonal.

```
L1_diag=sparse(2:21,1:20,1,21,21);
```

Finally, we build the diagonal that is 7 elements down from the main diagonal. These are the entries that arise from the line of nodes below the one being considered, e.g., h_8 occurs in the first equation (equation 6.3 and Figure 6.1).

```
L2_diag=sparse(8:21,1:14,1,21,21);
```

Note that we do not have to build the super diagonals, the ones above the main diagonal, explicitly --- they are just the transpose of the subdiagonals. The matrix that we want (actually, we will have to make a minor modification; see below) is just the sum of the appropriate diagonal matrices.

```
A=M_diag+L1_diag+L2_diag+L1_diag'+L2_diag';
```

How big is our sparse matrix? `size(A)`

```
ans =

          21    21
```

What does the matrix look like? `A(1:9,1:2)`

```
ans =
   (1,1)       -4
   (2,1)        1
   (8,1)        1
   (1,2)        1
   (2,2)       -4
   (3,2)        1
   (9,2)        1
```

You see that the only entries listed are non zero. The *MATLAB* "full" command can be used to convert a sparse matrix to the regular form.

```
full(A(1:9,1:9))
```

```
ans =
    -4     1     0     0     0     0     0     1     0
     1    -4     1     0     0     0     0     0     1
     0     1    -4     1     0     0     0     0     0
     0     0     1    -4     1     0     0     0     0
     0     0     0     1    -4     1     0     0     0
     0     0     0     0     1    -4     1     0     0
     0     0     0     0     0     1    -4     1     0
     1     0     0     0     0     0     1    -4     1
     0     1     0     0     0     0     0     1    -4
```

The matrix is not quite right yet. The matrix should be "blocked" with the limits of the "blocks" corresponding with the ends of the rows of the finite-difference grid. (Figure 6.2 shows that the matrix is composed of three 7x7 "blocks" that are identical.) The blocks arise because there is no "one" in the position to the right of the rightmost point on the row (e.g., variable 8 does not appear in the equation for node 7). Likewise, there is no "one" in the position to the left of the leftmost position in a row (e.g., variable 14 does not appear in the equation for node 15). Thus, we need to set several elements in our sparse matrix to zero.

```
A(7,8)=0;A(8,7)=0;A(14,15)=0;A(15,14)=0;
```

We can use the *MATLAB* "spy" command to look at the structure of matrix A graphically (Figure 6.2).

```
 spy(A)
```

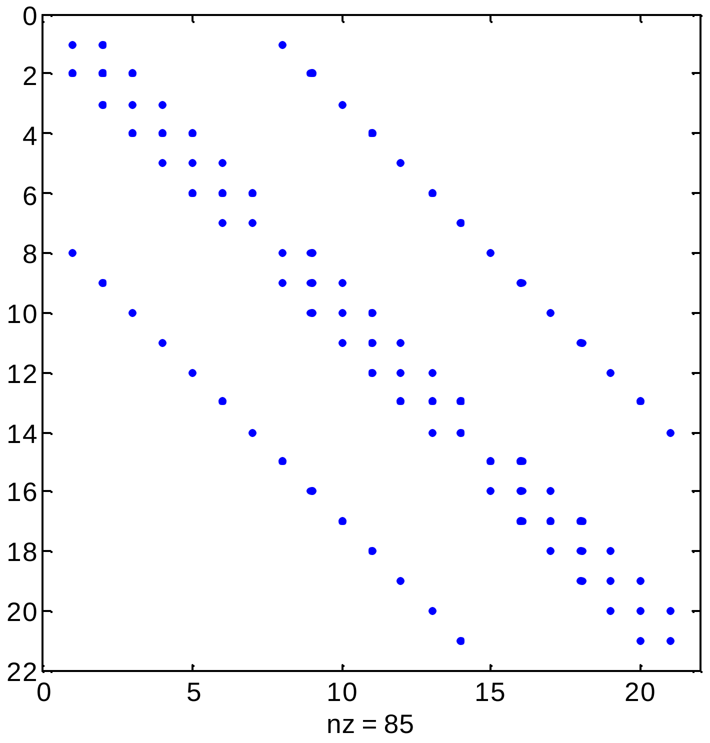


Figure 6.2. The sturcture of matrix A. The number of non-zero entries in the matrix is nz.

The final step, solving the equations, can be done using methods presented in Chapter 2. For example, we can set the right-hand vector using known quantities (the heads on the boundaries) and obtain the solution easily using the *MATLAB* backslash command

```
b=zeros(21,1);b(7)=-100;b(14)=-100;b(21)=-100 ;
```

```
h=A\b;
output=[h(1:7) h(8:14) h(15:21)]

output =
    0.3530    0.4989    0.3530
    0.9132    1.2894    0.9132
    2.0103    2.8324    2.0103
    4.2957    6.0194    4.2957
    9.1532   12.6538    9.1532
   19.6632   26.2894   19.6632
   43.2101   53.1774   43.2101
```

You might want to prepare an m-file, generalizing the statements above, to solve the sample problem for user-specified grid spacing and look at the effect of decreasing grid spacing on the answers.

6.4. Example problem: The "Tóth" problem

As an illustration of how to generalize the approach outlined above, consider the m-file below to solve the Laplace equation for a two-dimensional, vertical slice of an aquifer with a top boundary specified by undulating values of ground-water head. The undulations are to represent the effects of topography. Tóth (1963) investigated the problem in a now classic paper.

The boundary conditions for the left and right sides and for the bottom of the domain are not fixed heads. Rather, we assume that there is no flow across these boundaries. In this case the derivative of *h* with respect to the spatial coordinate is zero. We handle the condition by using the central difference approximation to the derivative at the boundary node. Consider the approximation for a boundary node "i" in a grid (Figure 6.3).

Equation (3.4), the central-difference approximation for the first derivative, requires values at "i-1" as well as at "i+1". That is,

$$\frac{\partial h}{\partial x} \approx \frac{\left(h_{i+1} - h_{i-1}\right)}{2\Delta x}$$

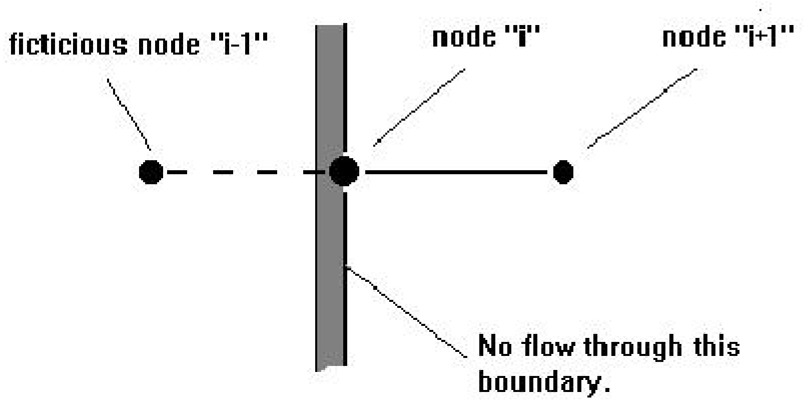


Figure 6.3. Nodes in the vicinity of a no-flow boundary.

In reality, we don't have a node at "i-1", but we insert a fictitious node just to allow us to treat the boundary condition. For no flow, the derivative above must be zero and this implies that h_{i-1} must equal h_{i+1}. Consequently, when h_{i-1} occurs in the finite-difference equations, we set it to h_{i+1}. In effect, this means that the coefficients on the "h_{i+1}'s" in the Tóth problem must be set to 2 instead of 1. (The best way for you to digest this is to study the code and think about the appropriate boundary conditions.)

```
% This is a finite-difference solution for Toth's problem.
% Code by George Hornberger and Jeff Raffensperger
%
% First set the grid size.
%
J=input('Number of nodes in the x direction? (Try 75 if in doubt.)')
K=input('Number of nodes in the y direction? (Try 25.) ')
W=input('Number of hill-valley waves? (Try 3.) ')
amp=input('Amplitude of hill-valley waves (0-10)? (Try 3.) ')
%
% The number of nodal points is J*K.
%
n=J*K;
%
% Finite-difference equations generate SPARSE matrices;
% that is, most entries are zero.
% MATLAB takes advantage of this sparsity by NOT storing the entire matrix.
%
% Set the coefficient matrix for the Laplace equation.
%
M_DIAG=sparse(1:n,1:n,-4,n,n);
L1_DIAG=sparse(2:n,1:n-1,1,n,n);
L2_DIAG=sparse(J+1:n,1:n-J,1,n,n);
A=L2_DIAG+L1_DIAG+M_DIAG+L1_DIAG'+L2_DIAG';
%
% Next set the vector of "knowns" and modify the coefficient matrix
% to account for the boundary conditions.
% The heads at the top are set to grade linearly from 20 to 0.2,
% with W sine waves of amplitude A superimposed.
%
dh=20/J;
hT=20:-dh:0.2;
%
for i=1:J
      hT(i)=hT(i)+amp*sin(2.*pi*((i-1)/(J/W)));
end
%
% Let's look at the topography of the water table...
%
dJ=0:1:J-1;
dK=0:1:K-1;
figure(1)
plot(dJ,hT)
%
% Next set the right-hand vector, adjust the coefficients on the no-flow
boundaries, and fix the matrix entries at the edges of the "blocks".
%
rhs=zeros(n,1);
for j=1:J                     % Bottom and top boundaries
    rhs((K-1)*J+j)=-hT(j);
    A(j,j+J)=2;
end
for k=1:K                     % Right-hand boundary
    A(k*J,k*J-1)=2;
end
for j=1:J:K*J                 % Left-hand boundary
    A(j,j+1)=2;
    if j>1 & j<K*J            % Block the matrix
        A(j,j-1)=0;
        A(j-1,j)=0;
    end
end
%
% The finite difference equations are in the form A*h=rhs, where
% "A" is the coefficient matrix, "h" is the vector of unknown heads,
% and "rhs" is the vector of known quantities.
%
% The MATLAB "\" function solves the system of equations.
%
h=A\rhs;
%
% The unknown heads can be put back into the gridded form to view
% in our x-y orientation.
%
hh=reshape(h,J,K);
%
% Add in the known heads along the top boundary and rotate the grid
% into the upright position.
%
hh=[hh hT'];
```

```
hh=hh';
%
% Next we can contour the heads.
figure(2)
contour(hh,12)
axis equal
%
% The MATLAB gradient function, allows us to calculate the flow
% directions.
%
[px,py]=gradient(hh);
%
% We can plot the flow vectors on the contour plot.
% (Note: the "1" in the "quiver" command regulates the size of the
% arrows.)
%
hold on;quiver(-px,-py,1);hold off;
```

6.5. Solving the two-dimensional Poisson equation

One problem that arises very frequently in practice involves the solution of a two-dimensional ground-water flow problem through a horizontal aquifer. Heads are averaged over the vertical and the equation to be solved is the Poisson equation ([Box 6.1]; also see a hydrogeology text – Fetter, 2001, for example – to review the derivation of the equation and the physical meaning of the terms.) The pertinent equation is:

$$\frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(T\frac{\partial h}{\partial y}\right)=-w \qquad (6.4)$$

where w is a source term such as recharge (dimensions of length per time); a negative w would indicate a sink such as evaporation or pumping. The transmissivity, T, can vary with x and y.

A block-centered grid is often very useful for solving ground-water problems, especially for grids with unequal grid spacing (varying Δx and Δy) and for spatially varying transmissivities, T [Box 6.2]. We picture the nodes at the center of blocks that define the transmissivities and the finite-difference steps in x and y directions. For example, consider the 3x5 set of blocks shown schematically below. Values of head, transmissivity, and grid spacing for each block are indicated within the block.

$h_1, T_1, \Delta x_1, \Delta y_1$	$h_2, T_2, \Delta x_2, \Delta y_2$	$h_3, T_3, \Delta x_3, \Delta y_3$		
$h_6, T_6, \Delta x_6, \Delta y_6$	$h_7, T_7, \Delta x_7, \Delta y_7$			
$h_{11}, T_{11}, \Delta x_{11}, \Delta y_{11}$				

A finite-difference approximation for the second derivative with respect to x in the second block above can be written

$$\frac{\left[\frac{2T_2T_3}{\Delta x_3T_2+\Delta x_2T_3}\left(h_3-h_2\right)-\frac{2T_1T_2}{\Delta x_2T_1+\Delta x_1T_2}\left(h_2-h_1\right)\right]}{\left(\frac{\Delta x_1+2\Delta x_2+\Delta x_3}{2}\right)}$$

These types of equations are the basis for many of the computer codes used to solve ground-water problems (e.g., see Bredehoeft 1990 and [Box 6.3]). What is important from our standpoint is that we again obtain a set of equations that must be solved and all we need do is determine how to set up the appropriate matrix and right-hand vector if we want to use *MATLAB* to obtain a solution.

Following Bredehoeft (1990) we define

$$\frac{T_{i-1/2}}{\Delta x_{i-1/2}}=\frac{2T_iT_{i-1}}{\Delta x_{i-1}T_i+\Delta x_iT_{i-1}},\quad c^x=\frac{T_{i-1/2}}{\Delta x_{i-1/2}\Delta x},\quad c^y=\frac{T_{j-1/2}}{\Delta y_{j-1/2}\Delta y},$$

$$c^x_{i+1}=\frac{T_{i+1/2}}{\Delta x_{i+1/2}\Delta x},\quad \text{and}\quad c^y_{j+1}=\frac{T_{j+1/2}}{\Delta y_{j+1/2}\Delta y},$$

where Δx is the width of the central (i,j) block, and Δy is the height of the central block. With these definitions, we can derive the following as the finite-difference representation of the Poisson equation.

$$c^xh_{i-1}+c^x_{i+1}h_{i+1}+c^yh_{j-1}+c^y_{j+1}h_{j+1}-\left(c^x+c^x_{i+1}+c^y+c^y_{j+1}\right)h_i=-w_i$$

Note that the finite-difference matrix for a problem will have the form (for a grid with 9 nodes in the x direction):

$$\begin{pmatrix}
a_1 & b_1 & & & & & & & d_1 & \\
c_2 & a_2 & b_2 & & & & & & & d_2 \\
 & c_3 & a_3 & b_3 & & & & & & \\
 & & & \cdot & & & & & & \\
 & & & & \cdot & & & & & \\
 & & & & & \cdot & & & & \\
 & & & & & & \cdot & & & \\
 & & & & & & & \cdot & & \\
e_9 & & & & & & & & a_9 & b_9 \\
 & e_{10} & & & & & & & & a_{10}
\end{pmatrix}$$

where the "*a*'s" are the coefficients on h_i in the Bredehoeft equation, the "*b*'s" are the coefficients on h_{i+1}, the "*c*'s" are the coefficients on h_{i-1}, the "*d*'s" are the coefficients on h_{j+1}, and the "*e*'s" are the coefficients on h_{j-1}.

6.6. An example problem

A waste-disposal pond is designed to lose water to evaporation and to "trap" the contaminants in the sludge at the bottom of the pond (Figure 6.4). Unfortunately, the pond, which is about 50 m on a side, leaks (as do all such ponds!) and contaminated water is recharged to the underlying aquifer at a rate of 4 m y^{-1} (~1.25 m s^{-1}). Recharge over the rest of the area in this semi-arid environment is negligible. Streams that dissect the area bound the aquifer to the north and south. The stream to the north, 75 m from the north end of the pond, is topographically higher and, on average, is a "losing stream", with water infiltrating into the aquifer. The boundary can be approximated as having a constant head of 0.2 m above datum. The boundary to the south, some 100 m from the south end of the pond, also can be approximated by a constant-head boundary with a head of 0m. Because the undisturbed flow in this stream-aquifer system tends to be directly from north to south, we can choose east and west boundaries (away from the pond) to be "no-flow" boundaries. The west no-flow boundary is 80 m from the edge of the pond and the east no-flow boundary is 50 m from the pond. The transmissivity of the aquifer is estimated to be 0.0015 m^2 s^{-1}, except in the southwestern part of the aquifer where the transmissivity is thought to be half of that value.

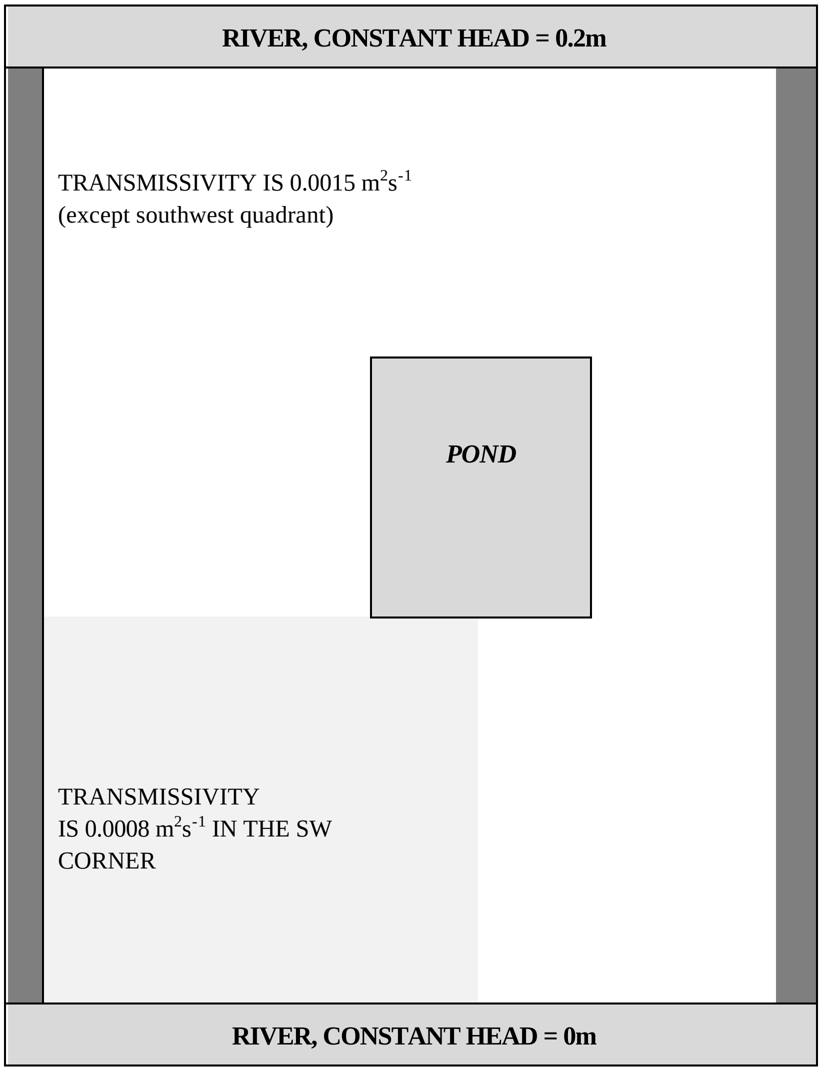

RIVER, CONSTANT HEAD = 0.2m
TRANSMISSIVITY IS 0.0015 m²s⁻¹
(except southwest quadrant)
POND
TRANSMISSIVITY
IS 0.0008 m²s⁻¹ IN THE SW
CORNER
RIVER, CONSTANT HEAD = 0m

Figure 6.4. Schematic diagram of the example problem.

For the example, we use a very simple (coarse) grid (Figure 6.5). The values in the schematic below give the values for the blocks in the grid. For the blocks representing the area of the pond, the size is 25 m x 25 m, the transmissivity is 0.0015 m^2 s^{-1}, and the recharge is 1.25x10^{-7} m s^{-1}. There are 25 cells (unknowns), numbered from "1" in the upper left, "2" in the next cell to the right, ..., "6" in the cell in the second row leftmost position, ..., and "25" in the lower right cell.

As mentioned above, the difficult part of finite-difference solutions is in setting up the matrices. Review the m-file listed below to see how one might proceed to write a code for this problem.

Δx 40 Δy 75 T1 1.5e-3	40 75 1.5e-3	25 75 T1	25 75 T1	50 75 1.5e-3
40, 25, 1.5e-3	40, 25, 1.5e-3	▨	▨	50, 25, 1.5e-3
40, 25, 1.5e-3	40, 25, 1.5e-3	▨	▨	50, 25, 1.5e-3
40, 50, T2 (8e-4)	40, 50, 8e-4	25 50 T2	25, 50, T1	50, 50, 1.5e-3
40, 50, 8e-4	40, 50, 8e-4	25 50 T2	25, 50 T1	50, 50, 1.5e-3

Figure 6.5. Schematic of the finite-difference blocks for the example problem. The dark shaded squares indicate the overlying pond and the lighter shaded cells are where the transmissivity is relatively lower.

```
% example code for poisson equation
% set the grid spacing and transmissivity matrices
[X,Y]=meshgrid(cumsum([0 40 40 25 25 50]),cumsum([0 75 25 25 50 50]));
dx=diff(X');dy=diff(Y);
dx=dx(:,1:5);dy=dy(:,1:5);dx=dx';
T=0.0015*ones(5,5);
% account for heterogeneity in the southwest corner
```

```
T(4,1:3)=0.0008;T(5,1:3)=0.0008;
% form "minus half" and "plus half" arrays for bredehoeft equations
Tjm1=[zeros(5,1),T(:,1:4)];Tim1=[zeros(1,5);T(1:4,:)];
Tjp1=[T(:,2:5), zeros(5,1)];Tip1=[T(2:5,:);zeros(1,5)];
dxim1=[zeros(1,5);dx(1:4,:)];dyjm1=[zeros(5,1),dy(:,1:4)];
dxip1=[dx(2:5,:); zeros(1,5)];dyjp1=[dy(:,1:4),zeros(5,1)];
% preallocate zero matrices for the coefficients
a=zeros(1,25);b=a;c=a;d=a;e=a;rhs=a;
% set up the coefficients
Toverdxminus=(2*T.*Tim1)./(dxim1.*T+dx.*Tim1);
Toverdxplus=(2*T.*Tip1)./(dxip1.*T+dx.*Tip1);
Toverdyminus=(2*T.*Tjm1)./(dyjm1.*T+dy.*Tjm1);
Toverdyplus=(2*T.*Tjp1)./(dyjp1.*T+dy.*Tjp1);
cx=Toverdxminus./dx;cy=Toverdyminus./dy;
cxp1=Toverdxplus./dx;cyp1=Toverdyplus./dy;
[ii,jj]=find(isnan(cx)==1);cx(ii,jj)=0;
[ii,jj]=find(isnan(cy)==1);cy(ii,jj)=0;
[ii,jj]=find(isnan(cxp1)==1);cxp1(ii,jj)=0;
[ii,jj]=find(isnan(cyp1)==1);cyp1(ii,jj)=0;
c=reshape(cx',1,25);b=reshape(cxp1',1,25);
e=reshape(cy',1,25);d=reshape(cyp1',1,25);
a=-(c+b+e+d);
a(1:5)=a(1:5)-b(1:5);
a(21:25)=a(21:25)-c(21:25);
a(1:5:21)=a(1:5:21)-d(1:5:21);
a(5:5:25)=a(5:5:25)-e(5:5:25);
% set the top boundary -- head of 0.2m
for j=1:5
      rhs(j)=rhs(j)-b(j)*0.2;
end
% set the no-flow conditions at the sides --note that the following loop
% is ok for this example, but does not work for general heterogeneity!!
for i=1:5
      k=(i-1)*5+1;
      d(k)=2*d(k);
      k=k+4;
      e(k)=2*e(k);
end
% recharge at the pond grid cells
w=-1.25e-7;
rhs(8)=w;rhs(9)=w;rhs(13)=w;rhs(14)=w;
% use the sparse matrix commands to set the full matrix and then solve the
equations
U1_diag=sparse(6:25,1:20,b(1:20),25,25);
L1_diag=sparse(1:20,6:25,c(6:25),25,25);
U2_diag=sparse(2:25,1:24,d(1:24),25,25);
L2_diag=sparse(1:24,2:25,e(2:25),25,25);
M_diag=sparse(1:25,1:25,a,25,25);
A=M_diag+U1_diag+U2_diag+L1_diag+L2_diag;
h=A'\rhs';
aa=reshape(h,5,5);
x=X(1,2:6);
y=Y(2:6,1);
mesh(x,y,aa);xlabel('distance, x, in m')
```

```
ylabel('distance, y, in m');zlabel('ground-water head, m')
pause
[xx,yy]=meshgrid(35:5:185,70:5:230);
zz=interp2(x,y,aa,xx,yy);surfc(zz);
zlabel('head, m');xlabel('regular grid point # (x)')
ylabel('regular grid point # (y)')
```

The matrix "aa" from the code above gives the heads arranged in an order to ease interpretation. You should keep in mind that the heads in matrix “aa” are for nodal points and, for this example problem, are ***not*** equally spaced. The *MATLAB* "contour" command and the related others that we used previously assume that the matrix elements are representative of equally spaced points. Thus, the output above should not be contoured directly, at least not with the *MATLAB* "contour" command. One thing we can do with unequally spaced data is use the "mesh" command (Figure 6.6) once the *x* and *y* values are specified.

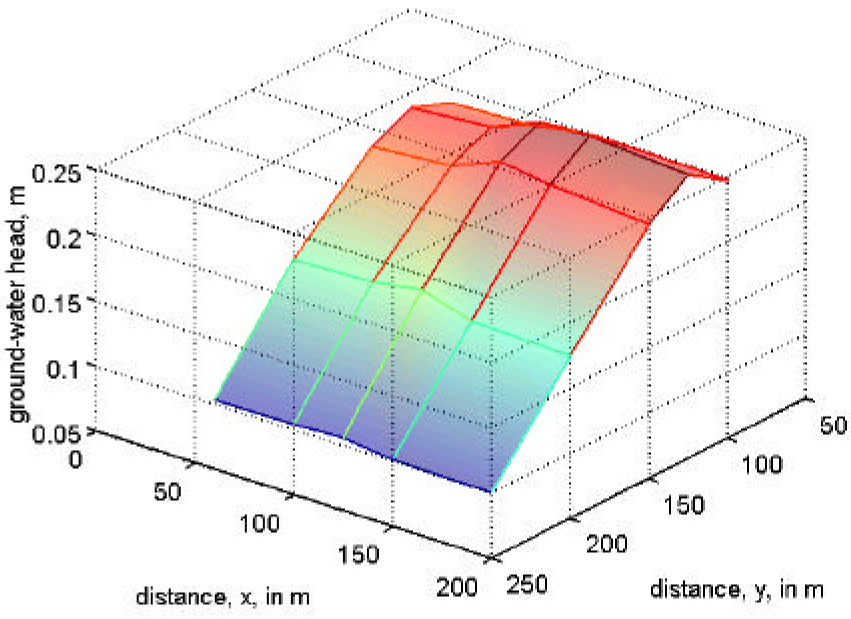


Figure 6.6. Meshplot of solution to example problem.

If we want to contour the heads, we can use the *MATLAB* `interp2` command to interpolate the unequally spaced data onto a regular grid.

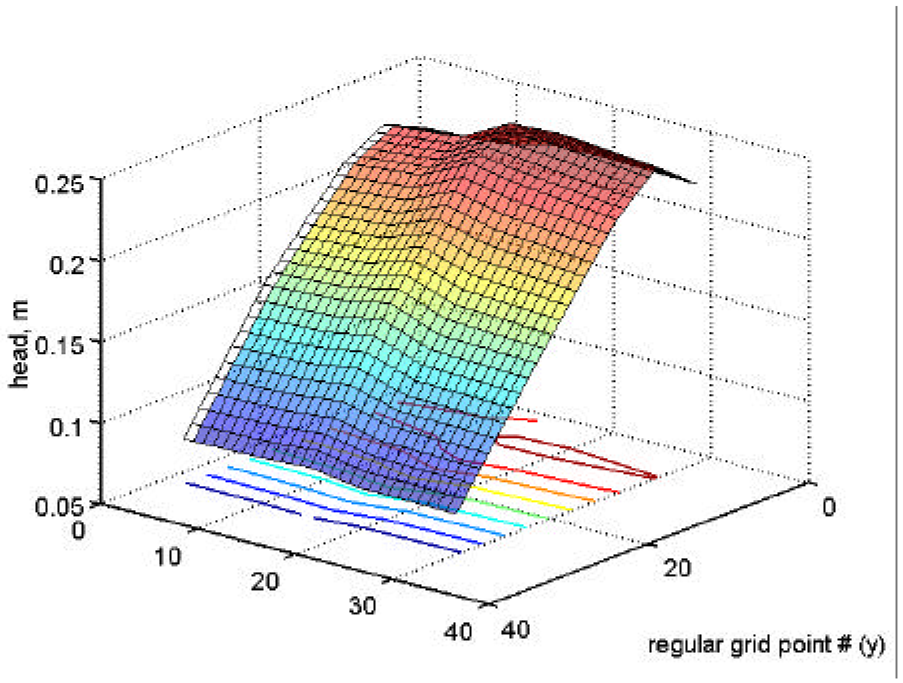


Figure 6.7. Data interpolated to a regular grid can be contoured.

6.7. Problems

The Brookhaven landfill was constructed on (Wisconsin) glacial outwash deposits. The landfill overlies the upper glacial aquifer, the lower boundary of which is the Gardiners Clay. The aquifer is composed primarily of sand and gravel with small amounts of feldspar, mica, and other minerals. See Wexler (1988) for details.

Precipitation in the area averages 47.4 inches annually. Under natural conditions, about half this amount recharges the aquifer, with the remainder going to evapotranspiration. The bulk of the recharge water flows toward the south in the upper glacial aquifer, discharging to streams and bays.

Depth to the water table ranges from 0 to 55 feet, depending on surface elevation. Saturated thickness of the upper aquifer ranges from 100 to 130 feet. The sands and gravels are quite permeable. Estimates of hydraulic conductivity range from 187 to 267 feet per day. Beaverdam Creek, which is fed almost totally by groundwater from the upper glacial aquifer, drains the area (Figure 6.8).

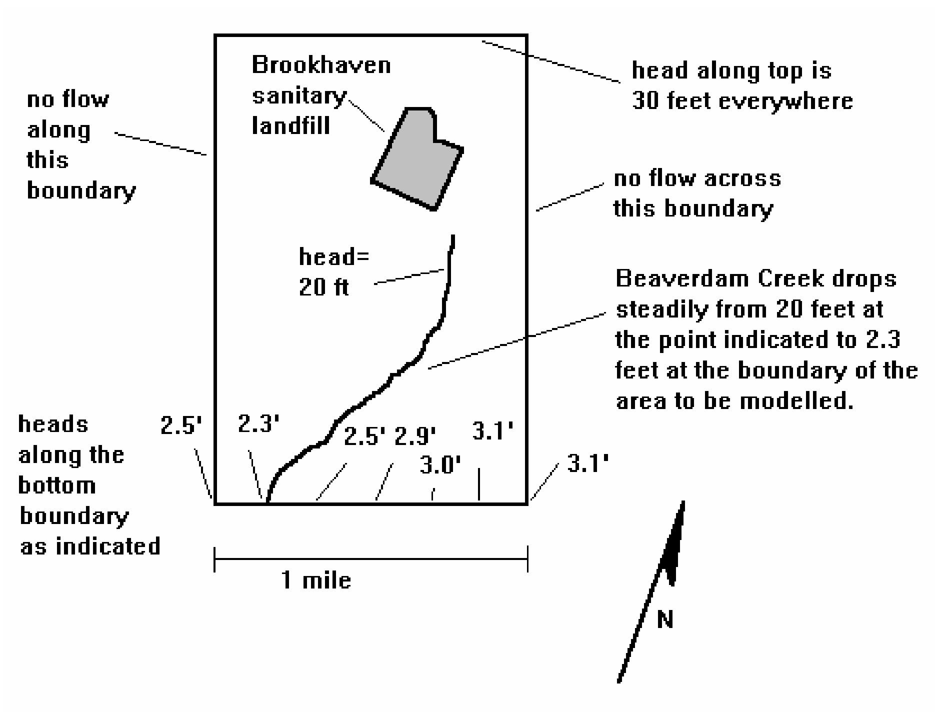

Brookhaven sanitary landfill
head along top is 30 feet everywhere
no flow along this boundary
no flow across this boundary
head= 20 ft
Beaverdam Creek drops steadily from 20 feet at the point indicated to 2.3 feet at the boundary of the area to be modelled.
heads along the bottom boundary as indicated
2.5'
2.3'
2.5'
2.9'
3.1'
3.0'
3.1'
1 mile
N

Figure 6.8. Schematic layout for the Brookhaven Landfill problem.

The landfill was constructed with a PVC liner, but despite this precaution, leachate has entered the aquifer. Landfill operations began in 1973 and by 1982 a plume extended several thousand feet to the southeast of the landfill.

As a consultant to the local government, your assignment is to develop a ground-water flow model for the site using the information available. The flow model is needed to interpret the directions of leachate migration. This interpretation is direct, in part, in that a flow net is generally a good guide to picturing contaminant migration. The interpretation is indirect, in part, in that the results from a flow model are required to implement a transport model. (Note: such assignments are "imprecise". You have great latitude in designing the model, but keep in mind that approximations must be made -- no model can capture even a fraction of the complexity of the "real world". The trick in developing a useful model is choosing useful approximations.) [Solve the problem first using a transmissivity two orders of magnitude smaller than suggested by the data. This will allow you to visualize what is going on more easily.]

One hint that you may find useful is how to assign a fixed head to an interior node. Suppose the general finite-difference equation for the interior node is:

$$e_i h_{i-J} + c_i h_{i-1} + b_i h_{i+1} + d_i h_{i+J} - a_i h_i = 0$$

where J is the number of columns in the finite-difference grid. Rather than change the value of h_i in all of the equations in which it appears, we can adjust just the particular equation above and carry on with the solution in an "as usual" fashion. To do this we change a_i to 1, b_i, c_i, d_i, and e_i to zero and change the 0 on the right-hand side to h_*, where h_* is the value at which we want to fix h_i. Note what this does. The values of e, c, b, and d are zero. Thus, the equation is

$$h_i = h_*$$

fixing h at this node at the desired constant value.

6.8. References

Bredehoeft, J.D., Microcomputer codes for simulating transient ground-water flow in two and three space dimensions. U.S. Geological Survey Open-file Report 90-559, 1990.

Fetter, C.W. Jr., *Applied Hydrogeology*, 598 pp., Prentice-Hall, Upper Saddle River, NJ, 2001.

McDonald, M.C., and Harbaugh, A.W., A modular three-dimensional finite-difference ground-water flow model: U.S. Geological Survey Techniques of Water-Resources Investigations, Book 6, Chap. A1, 586 pp., 1988.

Harbaugh, A.W., Banta, E.R., Hill, M.C., and McDonald, M.G., MODFLOW-2000, the U.S. Geological Survey modular ground-water model -- User guide to modularization concepts and the Ground-Water Flow Process: U.S. Geological Survey Open-File Report 00-92, 121 pp., 2000.

Tóth, J.A., A theoretical analysis of ground-water flow in small drainage basins, *J. Geophys. Res., 68*: 4795-4811, 1963.

Wexler, E.J., Ground-water flow and solute transport at a municipal landfill site on Long Island, New York. Part 1. Hydrogeology and water quality. U.S.Geological Survey Water Resources Investigations Report 86-4070, 1988.

Box 6.1. Groundwater flow equations

The basis of the equations used to describe the flow of groundwater is the conservation of mass equation, which, when applied to a fixed control volume, basically says that the rate of mass inflow minus rate of mass outflow equals rate of change of mass storage – what goes in minus what goes out equals the change in what's inside.

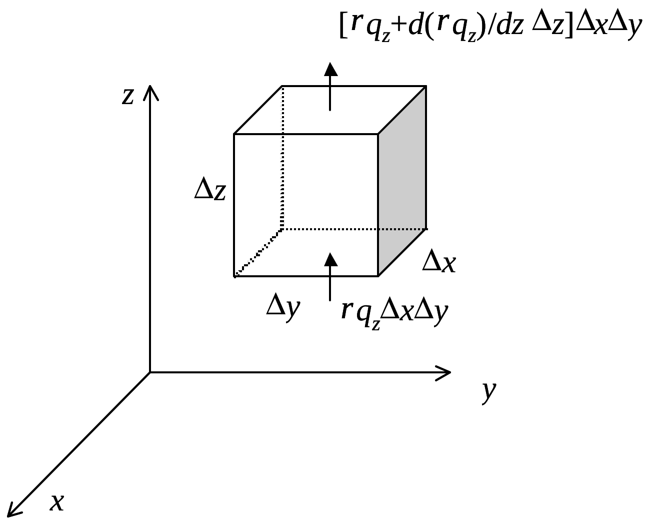

$[r q_z + d(r q_z)/dz\, \Delta z]\Delta x \Delta y$
z
Δz
Δx
Δy
$r q_z \Delta x \Delta y$
y
x

Figure B6.1.1. Control volume for deriving the conservation equation.

One way to derive the general equation of continuity for pore-fluid flow is to specify an arbitrary control volume to be a small rectangular parallelepiped in a fixed, Cartesian coordinate frame with sides of length Δx, Δy, and Δz (Fig. B6.1.1). Without any loss of generality, we take the directions designated by the arrows on the axes as positive (Fig. B6.1.1) and consider the case of positive flows. Consider first the inflow of mass into the control volume. The inflow into the parallelepiped in the z-direction is $r\, q_z \Delta x \Delta y$, where r is the density of water and q_z is the specific discharge (volumetric discharge per unit area) in the z direction. Density times the specific discharge gives the mass flux (mass per area per time) so multiplication by the area, $\Delta x \Delta y$, yields the mass inflow in the z direction. The mass flows in the x- and y-directions can be similarly calculated. Because specific discharge can change with distance, the value of q_z at the top face need not be the same as that at the bottom face. We can estimate the specific discharge at the top face using the Taylor series (Chapter 3.2). Because the distance separating the two faces (Δz) is as small as we wish to make it, we can get an acceptable approximation of the flux at the top face by just retaining the first two terms of the series (i.e., a linear extrapolation):

$$q_z(z + \Delta z) = q_z(z) + \frac{\P\, q_z}{\P z}\Delta z \qquad \text{(B6.1.1)}$$

The expression for the mass outflow in the *z* direction is then

$$\left[r\,q_z + \frac{\P\ rq_z}{\P z}\Delta z\right]\Delta x\Delta y \qquad \text{(B6.1.2)}$$

Now the expression that we need for the continuity equation is the **net** inflow of mass – the difference between the inflow and the outflow. For the *z* direction,

$$\text{net mass flow}_Z = (rq_z(z) - rq_z(z+\Delta z))\Delta x\Delta y = -\frac{\P rq_z}{\P z}\Delta x\Delta y\Delta z$$

Using similar expressions for the *x*- and *y*-directions, the total net mass inflow can be obtained:

$$\text{total net mass inflow} = -\left[\frac{\P\ rq_x}{\P x} + \frac{\P\ rq_y}{\P y} + \frac{\P\ rq_z}{\P z}\right]\Delta x\Delta y\Delta z \qquad \text{(B6.1.3)}$$

For *steady* flow there can be no change of mass in the control volume so the net inflow must be zero. If we further assume that the density is constant, equation (B6.1.3) implies that

$$\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} = 0 \qquad \text{(B6.1.4)}$$

The forces driving flow through an aquifer are due to gravity and pressure gradients. These forces, expressed on a per unit weight basis, are represented in groundwater flow equations in terms of a *head gradient*. Groundwater head is defined as pressure per unit weight plus elevation, which is the head due to gravity.

Darcy's law relates the specific discharge to the head gradient. Darcy's law states that this relationship is linear, with the constant of proportionality between specific discharge and head gradient being the *hydraulic conductivity*, *K*. If we make the assumption that the aquifer is *isotropic*, i.e., that *K* is independent of direction, Darcy's law can be written as follows.

$$\begin{aligned} q_x &= -K\frac{\partial h}{\partial x} \\ q_y &= -K\frac{\partial h}{\partial y} \\ q_z &= -K\frac{\partial h}{\partial z} \end{aligned} \qquad \text{(B6.1.5)}$$

Combining equations (B6.1.4) and (B6.1.5), we obtain an equation for steady groundwater flow.

$$\frac{\partial}{\partial x}\left(K\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(K\frac{\partial h}{\partial y}\right) + \frac{\partial}{\partial z}\left(K\frac{\partial h}{\partial z}\right) = 0 \qquad \text{(B6.1.6)}$$

For the case of a *homogeneous* aquifer, one for which K is constant, equation (B6.1.6) reduces to the Laplace equation.

$$\frac{\partial^2 h}{\partial x^2}+\frac{\partial^2 h}{\partial y^2}+\frac{\partial^2 h}{\partial z^2}=0 \tag{B6.1.7}$$

For the case of a horizontal aquifer of constant thickness, b, we assume that there is no vertical flow so equation B6.1.4 takes the form

$$b\frac{\partial q_x}{\partial x}+b\frac{\partial q_x}{\partial x}=0 \tag{B6.1.8}$$

When (B6.1.8) is combined with Darcy's law, we obtain

$$\begin{aligned}\frac{\partial}{\partial x}\left(Kb\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial x}\left(Kb\frac{\partial h}{\partial x}\right)=0\\ \frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right)=0\end{aligned} \tag{B6.1.9}$$

where T is the *transmissivity* of the aquifer. If there is recharge to the aquifer, e.g., by slow flow through an overlying confining layer, the equation is modified accordingly:

$$\frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right)=-w \tag{B6.1.10}$$

where w is the recharge rate.

If the aquifer is homogeneous, T is constant and can be brought outside the derivative in (B6.1.9). The result is

$$\frac{\partial^2 h}{\partial x^2}+\frac{\partial^2 h}{\partial y^2}=0 \tag{B6.1.11}$$

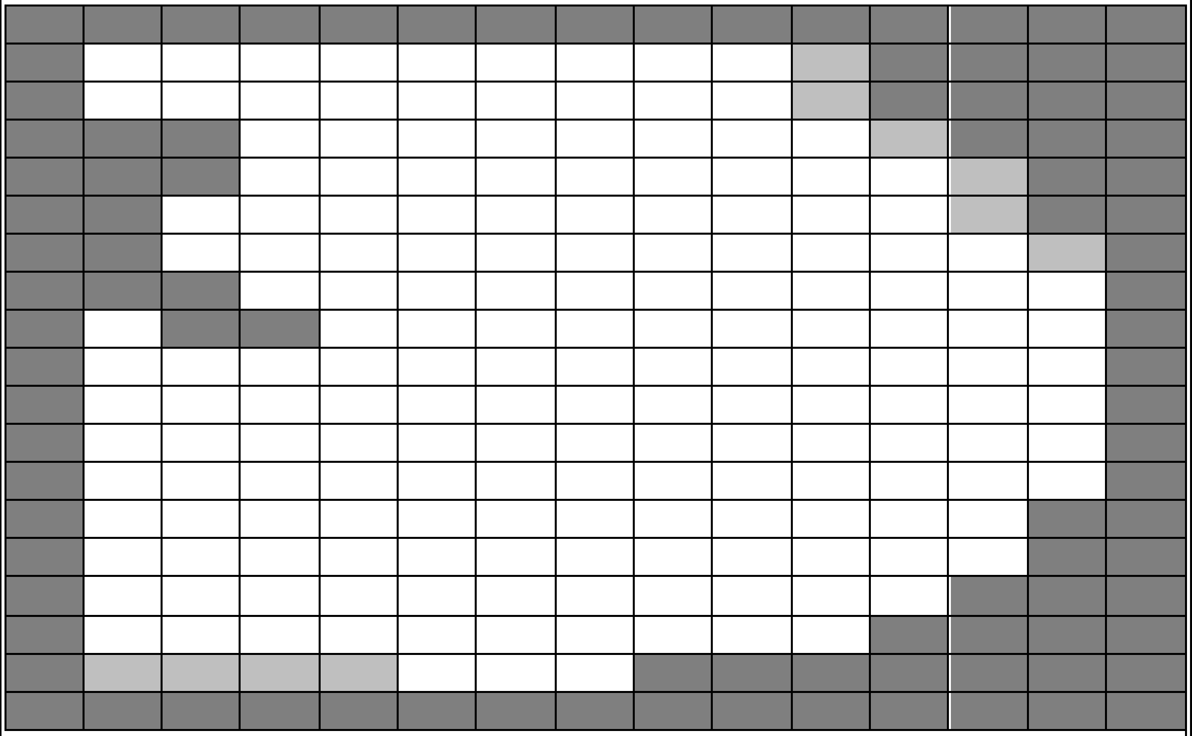
so again we find that the Laplace equation describes the steady flow of groundwater through a horizontal aquifer.

Finally, note that Darcy's law indicates that flow is down the *gradient* in head. This implies that flow lines for groundwater in an isotropic aquifer are perpendicular to lines of constant head. In *MATLAB* this means that if groundwater heads are computed and contour lines drawn, the `gradient` and `quiver` commands can be used to depict the flow.

Box 6.2. MODFLOW

The most widely used computer program in the world for simulating groundwater flow MODFLOW, a modular code developed by the U.S. Geological Survey in the early 1980's and continually improved since then (McDonald and Harbaugh, 1988; Harbaugh et al., 2000). The code along with documentation can be downloaded from the USGS Web site.

MODFLOW uses the finite difference method to solve the groundwater equations using the block centered node approach. An aquifer system is divided into rectangular blocks by a grid organized by rows, columns, and layers. Each block is called a "cell." Hydraulic properties are assigned to the cells, and boundary conditions are set by specifying cells to be constant head, no-flow, and so forth (see Figure B6.2.1).

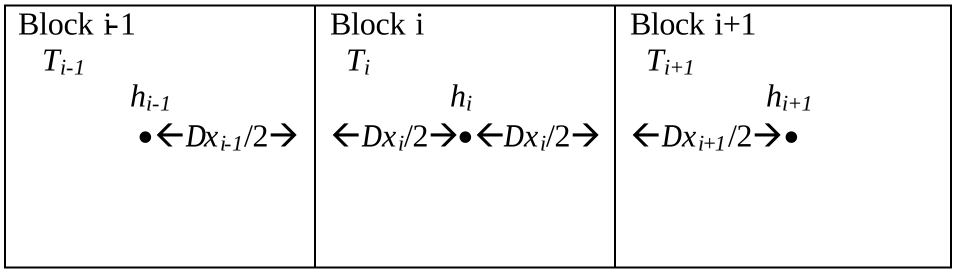


□ Cells where heads are to be computed
▨ Cells where heads are specified
■ Inactive cells – no flow

Figure B6.2.1. Example of a MODFLOW grid. Finite difference equations are written for cell-centered nodes.

Box 6.3. A block-centered finite difference approximation

For the block-centered approach, the transmissivity (and other properties) are assigned to blocks. The nodes, at which the heads are calculated, are at the center of the blocks.



The partial derivative in the *x* direction, $\frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right)$, is approximated as:

$$\frac{\left(T\frac{\partial h}{\partial x}\right)_{i+\frac{1}{2}}-\left(T\frac{\partial h}{\partial x}\right)_{i-\frac{1}{2}}}{\Delta x_i}$$

The derivatives at the halfway points are approximated in a straightforward manner. For example,

$$\left(T\frac{\partial h}{\partial x}\right)_{i+\frac{1}{2}}\approx\frac{2T_iT_{i+1}}{T_i+T_{i+1}}\frac{(h_{i+1}-h_i)}{\Delta x_i}$$

in which the transmissivity is approximated using the *harmonic mean*, $\frac{2T_iT_{i+1}}{T_i+T_{i+1}}$.

If the block sizes are unequal, the equations are modified slightly; see Bredehoeft (1990) for further details.