CHAPTER 6

Introduction to Finite Difference Methods for Partial Differential Equations

Numerical Methods in the Hydrological Sciences: AGU Lecture Notes 1
Copyright 2004 by the American Geophysical Union
doi: 10.1029/001LN8
(Version 2004-08-02)

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:

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

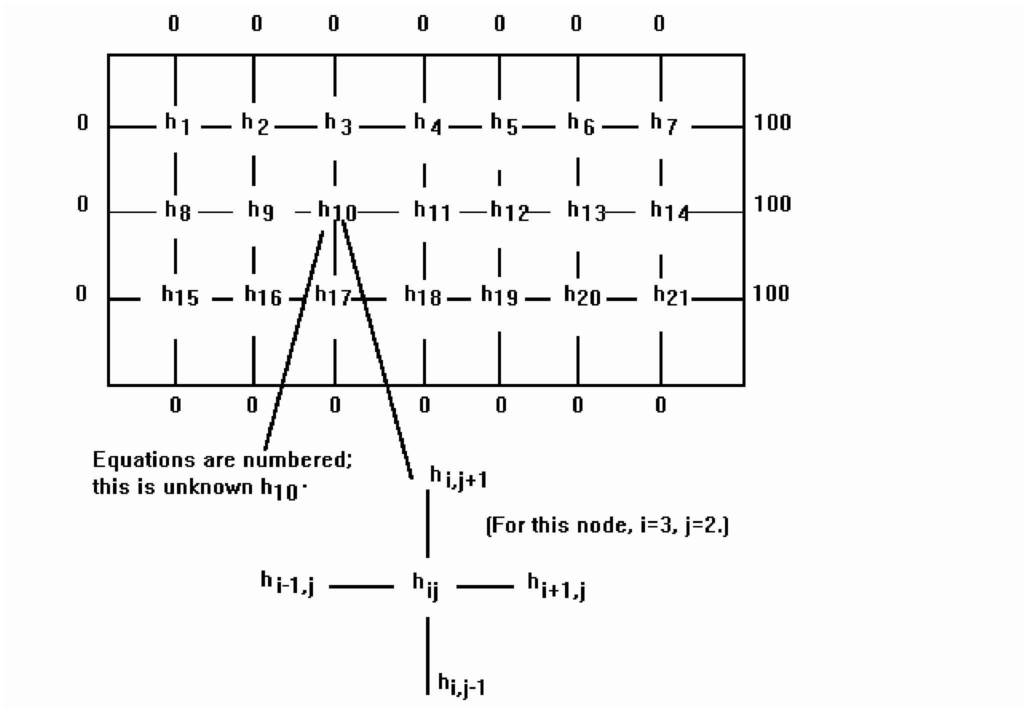

0 0 0 0 0 0 0
0
0
0
h1 h2 h3 h4 h5 h6 h7
h8 h9 h10 h11 h12 h13 h14
h15 h16 h17 h18 h19 h20 h21
100
100
100
0 0 0 0 0 0 0
Equations are numbered; this is unknown h10.
hi,j+1
(For this node, i=3, j=2.)
hi-1,j
hij
hi+1,j
hi,j-1

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:

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

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

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

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)
```

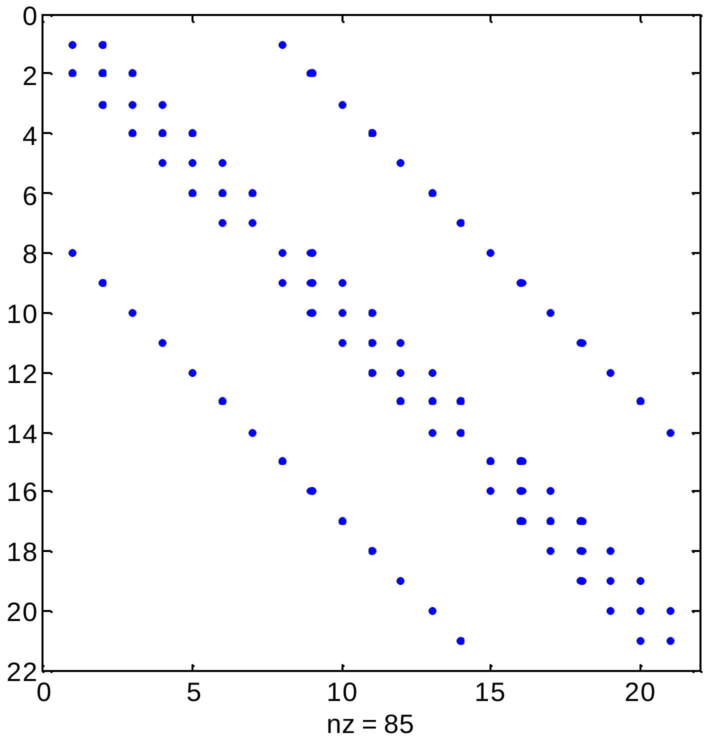


Figure 6.2. The sturcture of matrix A. The number of non-zero entries in the matrix is nz.

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,

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

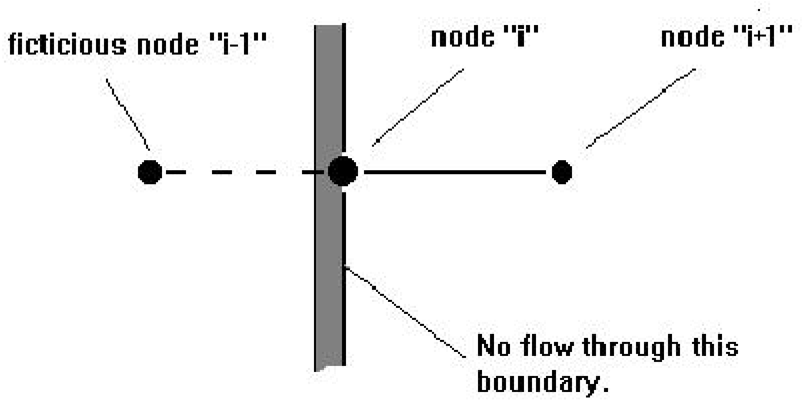


Figure 6.3. Nodes in the vicinity of a no-flow boundary.

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 $h_{i\text{-}1}$ must equal h_{i+1}. Consequently, when $h_{i\text{-}1}$ 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, Δx_1, Δy_1	h_2, T_2, Δx_2, Δy_2	h_3, T_3, Δx_3, Δy_3		
h_6, T_6, Δx_6, Δy_6	h_7, T_7, Δx_7, Δy_7			
h_{11}, T_{11}, Δx_{11}, Δy_{11}				

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

$$\frac{\left[\dfrac{2T_2T_3}{\Delta x_3 T_2 + \Delta x_2 T_3}\left(h_3 - h_2\right) - \dfrac{2T_1T_2}{\Delta x_2 T_1 + \Delta x_1 T_2}\left(h_2 - h_1\right)\right]}{\left(\dfrac{\Delta x_1 + 2\Delta x_2 + \Delta x_3}{2}\right)}$$

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

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

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

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.

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

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.

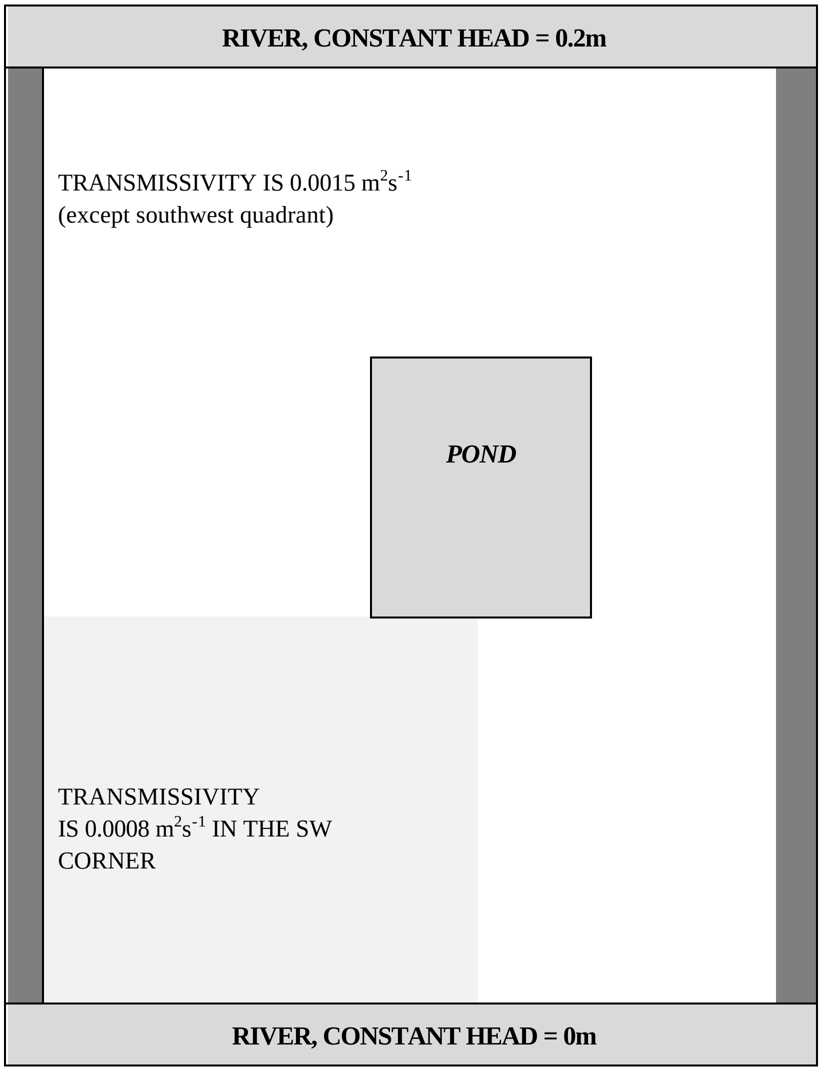

RIVER, CONSTANT HEAD = 0.2m
TRANSMISSIVITY IS 0.0015 m^2s^{-1}
(except southwest quadrant)
POND
TRANSMISSIVITY IS 0.0008 m^2s^{-1} IN THE SW CORNER
RIVER, CONSTANT HEAD = 0m

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.

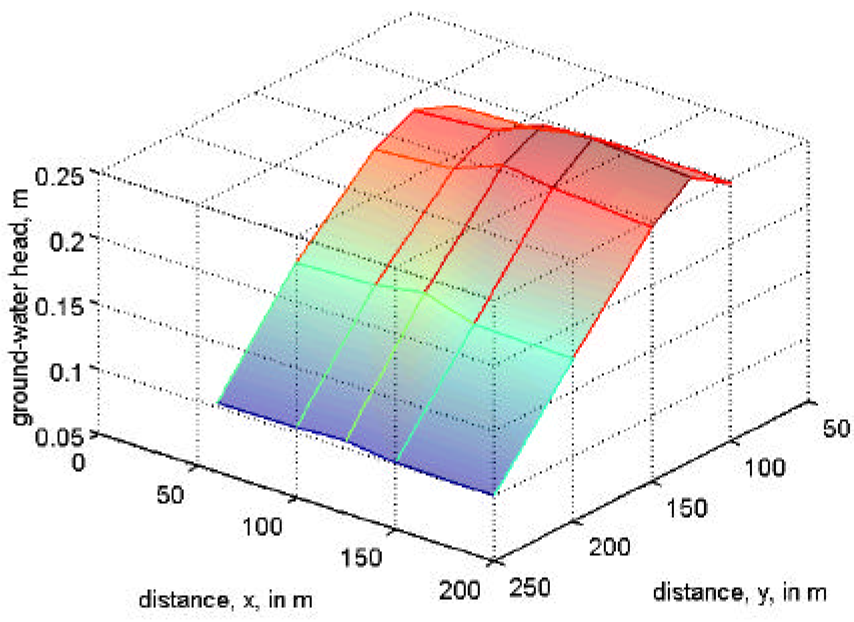


Figure 6.6. Meshplot of solution to example problem.

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

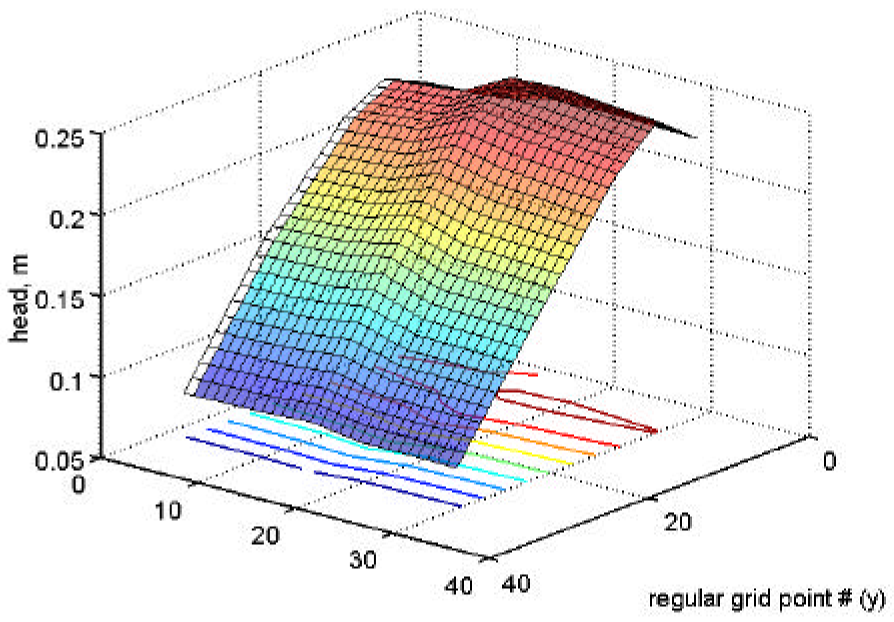


Figure 6.7. Data interpolated to a regular grid can be contoured.

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).

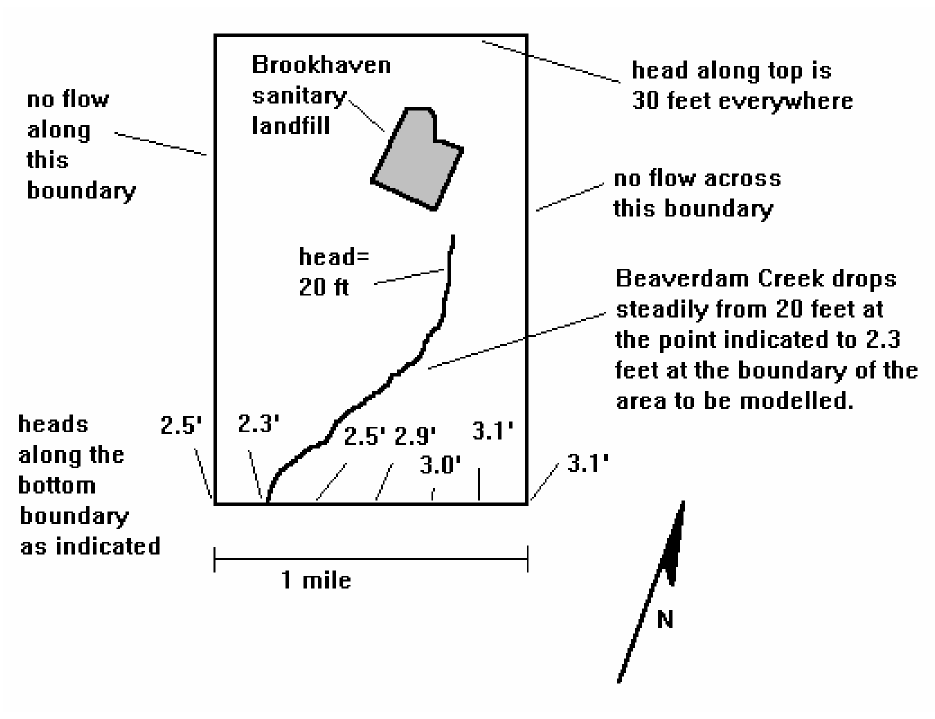

Brookhaven sanitary landfill
head along top is 30 feet everywhere
no flow along this boundary
no flow across this boundary
head= 20 ft
Beaverdam Creek drops steadily from 20 feet at the point indicated to 2.3 feet at the boundary of the area to be modelled.
heads along the bottom boundary as indicated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

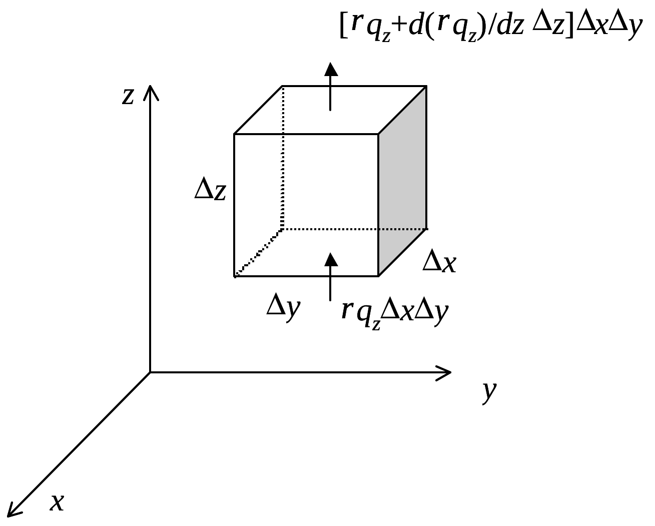
Harbaugh, A.W., Banta, E.R., Hill, M.C., and McDonald, M.G., MODFLOW-2000, the U.S. Geological Survey modular ground-water model -- User guide to modularization concepts and the Ground-Water Flow Process: U.S. Geological Survey Open-File Report 00-92, 121 pp., 2000.

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):

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

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

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

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

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

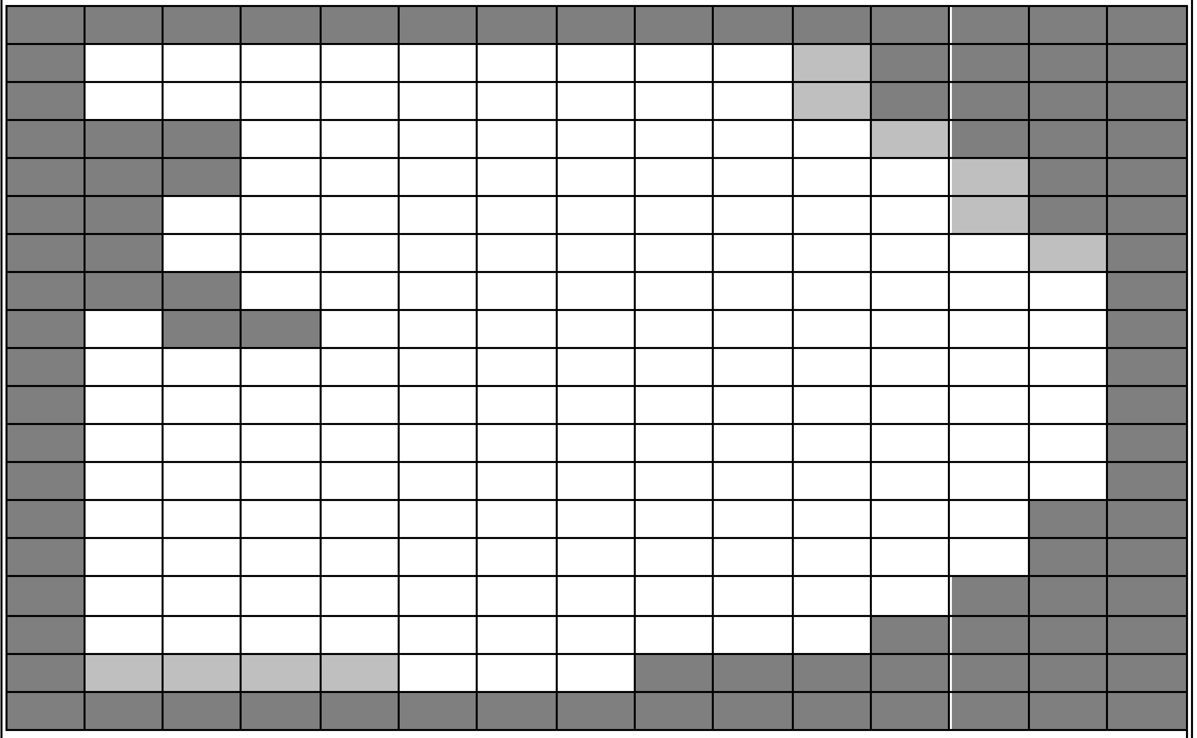
so again we find that the Laplace equation describes the steady flow of groundwater through a horizontal aquifer.

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).

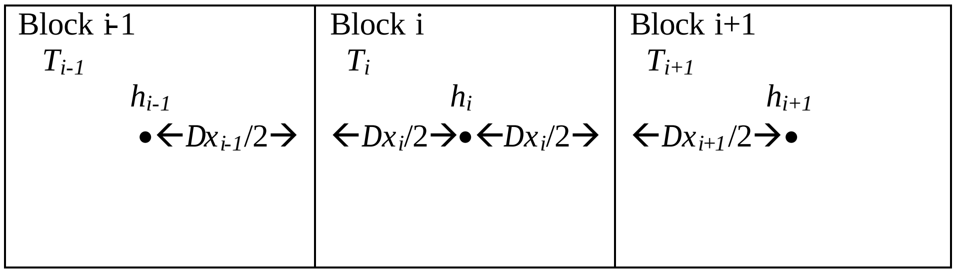


⬜ Cells where heads are to be computed
▨ Cells where heads are specified
⬛ Inactive cells – no flow

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.

CHAPTER 7

Iterative Solution of Systems of Equations

Numerical Methods in the Hydrological Sciences: AGU Lecture Notes 1
Copyright 2004 by the American Geophysical Union
doi: 10.1029/001LN9
(Version 2004-08-02)

7. Iterative Solution of Systems of Equations

7.1. Introduction

The methods for solving partial differential equations presented in Chapter 6 relied on Gaussian elimination (conveniently implemented with *MATLAB*'s "\" command) to solve the system of equations obtained when finite differences are used to approximate the derivatives. Gaussian elimination is not the only way to solve a system of equations. Iterative methods provide an alternative that can be advantageous. For example, when programming in one of the popular languages (FORTRAN, PASCAL, C), it is much easier to code the "loop" calculations needed for typical iterative solutions than it is to code Gaussian elimination. The matrix approach used in Chapter 6 is convienent for *MATLAB* but is not the way the calculations generally are done when working with a "non-matrix" language. [Consult the text by Wang and Anderson (1982) for more detail.] Also, some of the very large matrices that arise in complex hydrological problems are not solved directly very easily – iterative methods are actually preferred to Gaussian elimination. The basic idea of iterative solutions to equations was already presented in Chapter 2. In this chapter, we explore how iterative methods can be used to solve systems of equations.

7.2. Fixed-point iteration revisited

Recall that "fixed-point iteration" is one option for solving nonlinear equations [Chapter 2.6]. The equation is written in the form

$$x = g(x)$$

and the iteration formula is simply

$$x_{k+1} = g(x_k).$$

Convergence of the iteration is not guaranteed in general and the convergence criterion typically used is a check on whether x_{k+1} is "sufficiently close" to x_k.

Of course a linear equation is just a special case of a nonlinear equation. Although iteration doesn't make good sense computationally for a single linear equation, we can look at this case to motivate the actual use that interests us – solution of systems of linear equations. So, let's take the simple example

$$3x = 6.$$

To develop an iteration formula, we must "split" the 3. For example, we can say that $3x=2x+x$, subtract x from both sides of the equation, divide by 2, and have the iteration formula

$$x_{k+1} = \frac{-x_k}{2} + 3.$$

The first several steps in the iteration are

$$x_1 = \frac{-x_0}{2} + 3$$

$$x_2 = \frac{-x_1}{2}+3=\frac{-1}{2}\left(\frac{-x_0}{2}+3\right)+3=\left(\frac{-1}{2}\right)^2 x_0+3\left(1-\frac{1}{2}\right)$$

$$x_3 = \frac{-x_2}{2}+3=\left(\frac{-1}{2}\right)^3 x_0+3\left(1-\frac{1}{2}+\left(\frac{1}{2}\right)^2\right)$$

:

:

$$x_{k+1}=\left(\frac{-1}{2}\right)^{k+1} x_0+3\left(1-\frac{1}{2}+\left(\frac{1}{2}\right)^2-\left(\frac{1}{2}\right)^3+\ldots+\left(\frac{1}{2}\right)^k\right).$$

We can see from inspection of the general equation above that the iteration converges to the right answer. The first term goes to zero as $k\to\infty$ (i.e., 1/2 to the k^{th} power approaches zero). The term in brackets is the geometric series. The limit of this series is [1/(1+1/2)]=2/3, so in the limit, x_k approaches 3*(2/3)=2.

There are other ways – in fact, an infinite number of ways – we can "split" 3 in the original equation. The general case can be represented as

$$x_{k+1}=\frac{-(3-a)}{a}x_k+\frac{6}{a}=\left[\frac{-(3-a)}{a}\right]^{k+1}x_0+\left(\frac{6}{a}\right)\left(1-\frac{3-a}{a}+\left(\frac{3-a}{a}\right)^2-\ldots+\left(\frac{3-a}{a}\right)^k\right).$$

We can see the requirement for convergence of the method. The term $-(3-a)/a$ must be less than one in magnitude. (Otherwise, as the term is raised to higher and higher powers, the iteration diverges.) This condition requires that a >3/2. Is there a "best" value for a? It should be clear from an examination of the steps above that the smaller our "iteration number", the faster the effect of the initial guess goes to zero and the faster the terms in the series goes to zero and hence the faster this series converges. Therefore, we want to have the absolute value of the "iteration number" $(3-a)/a$ be as small as possible. We can look at values for the iteration number as a varies from 3/2 to 10.

```
alpha=3/2:1/6:10;
iter=abs((3-alpha)./alpha);
plot(alpha,iter);
xlabel('value of the splitting parameter alpha')
ylabel('magnitude of the iteration number')
```

The graph that results from the *MATLAB* operations above (Fig. 7.1) shows that there is a "best" value. As we would have expected for this simple (simple-minded?) problem, 3 is the best splitting parameter because then no iteration at all is required! The example is nonetheless instructive because one can see that the progress of the iteration – how fast the process converges – depends on the choice for splitting. To make good use of information on whether a split is "good", we need to know something about the magnitude of the iteration number. We will return to this notion directly, but first, let's look at solutions for matrix-vector equations.

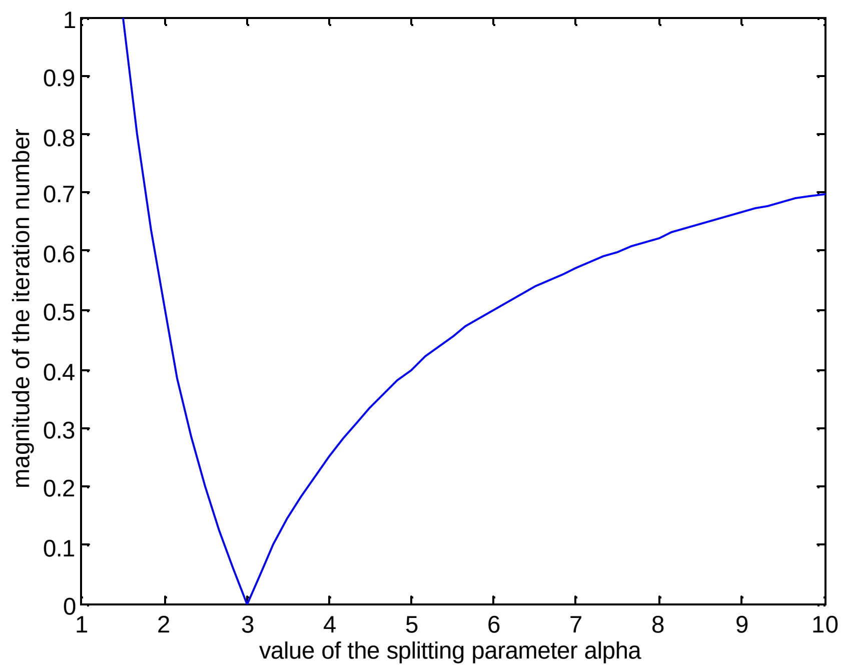


Figure 7.1. Splitting parameter for the example 3*x*=6.

7.3. Iterative solution of a system of equations

The notions above generalize to matrix-vector equations pretty well. Take a system of equations given by

$$Ax = b.$$

Split the matrix *A* into *D*+*B*. Rewrite the equation as follows,

$$Dx = -Bx + b$$

$$x = -D^{-1}Bx + D^{-1}b$$

which leads to the iteration formula

$$x_{k+1} = -D^{-1}Bx_k + D^{-1}b\,.$$

General iteration formulas have this form. The specifics differ because the choice of how to split the matrix differs. The simplest iteration (or at least one very simple one) uses the diagonal elements of matix *A* to form *D*. This method uses the sum of the part of *A* that is below the diagonal (*L*) and the part of *A* that is above the diagonal (*U*) as the *B* matrix. The iteration formula is therefore

$$x_{k+1} = -D^{-1}\left(L+U\right)x_k + D^{-1}b = \left[Q\right]^{k+1}x_0 + \left(I+Q+Q^2+\ldots+Q^k\right)D^{-1}b,$$

where $Q=D^{-1}(L+U)$. Note that because *D* is a diagonal matrix, D^{-1} is trivial to compute – it is just the inverse of the individual diagonal elements of *D*. This is a feature sought in iteration methods because computational ease is one of the reasons for using iterative methods. This simple iteration formula is known as *Jacobi* iteration. For finite-difference approximations to the Laplace equation [Chapter 6.3], this method "works", that is, it converges. Note that this means that $Q^k \rightarrow 0$ as $k \rightarrow \infty$. But how fast will it converge? And are there better methods? How can we judge these methods? For the single linear equation that we used as an example in the first section of these notes, it turned out that the magnitude of the "iteration number" was important. But now we have an iteration matrix, Q, rather than an iteration number. We need to examine the notion of the "magnitude" of a matrix before we proceed.

7.4. Vector and matrix norms

You are already familiar with the idea of magnitude applied to a vector. If the x component of velocity is 10 m s^{-1} and the y component is 10 m s^{-1}, you know that the magnitude of the velocity is about 14.14 m s^{-1}. You arrive at this answer by squaring the lengths of the x and y components, adding, and taking the square root. This measure of the magnitude of a vector is known as the L_2 norm. A "norm" is just the generalization of a measure of size.

The norm of a vector x is an associated, non-negative number, $\|x\|$, that satisfies the following requirements:

1) $\|x\| > 0$ for $x \neq 0$; $\|0\| = 0$

2) $\|cx\| = |c|\|x\|$

3) $\|x + y\| \leq \|x\| + \|y\|$

Here are the three most popular ways of assigning a vector norm.

i) $\|x\|_\infty = \max\limits_i |x_i|$ (the maximum component of the vector)

ii) $\|x\|_1 = |x_1| + |x_2| + ... + |x_n|$ (the sum of the lengths of all of the componenents)

iii) $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$ (the "standard" norm with which you are familiar).

These are often referred to as the L_∞, L_1, and L_2 norms, respectively. These are the most commonly used forms of the more general L_p norm, $\|x\|_p = [x_1^p + x_2^p + x_3^p + \circ\circ\circ + x_n^p]^{1/p}$ [Box 7.1].

The norm of a square matrix A is a non-negative number, $\|A\|$, satisfying

1) $\|A\| > 0$ *for* $A \neq 0$; $\|Z\| = 0, \text{iff } Z = 0$

2) $\|cA\| = |c|\|A\|$

3) $\|A + B\| \leq \|A\| + \|B\|$

4) $\|AB\| \leq \|A\|\|B\|$

Because in many problems we have both matrices and vectors, it is convienent to introduce the norm of a matrix in such a way that it will be consistent with a vector norm. A matrix norm is said to be compatible with a given vector norm if

$\|Ax\| \le \|A\|\|x\|$.

A matrix norm constructed to be compatible is said to be subordinate to the given vector norm.

For the three "popular" vector norms given above, the subordinate matrix norms are:

i) $\|A\|_\infty = \max\limits_i \sum\limits_{k=1}^{n} |a_{ik}|$, (the maximum sum of row elements)

ii) $\|A\|_1 = \max\limits_k \sum\limits_{i=1}^{n} |a_{ik}|$, (the maximum sum of column elements)

iii) $\|A\|_2 = \sqrt{l_1}$, where l_1 is the largest eigenvalue of the matrix $A'A$.

Next consider how matrix-vector norms are useful in examining iteration. Here are three useful theorems about matrix norms (given without proof).

Theorem 1. In order that $A^m \to 0$ as $m \to \infty$, a necessary and sufficient condition is that the eigenvalues of A be less than unity in magnitude.

Theorem 2. No eigenvalue of a matrix exceeds any of its norms in modulus. (This means that, from Theorem 1, sequential powers of a matrix will converge to zero if any matrix norm is <1.)

Theorem 3. In order that the series $I + A + A^2 + ... + A^m + ...$ converge, it is necessary and sufficient that $A^m \to 0$. In this case, the sum of the series converges to $(I\text{-}A)^{-1}$.

Armed with this knowledge, we go back to Jacobi iteration,

$$x_{k+1} = -D^{-1}(L+U)x_k + D^{-1}b = [Q]^{k+1} x_0 + \left(I + Q + Q^2 + ... + Q^k\right) D^{-1}b,$$

from which we see that the method converges if and only if $\|Q\| < 1$. Note that, in direct analogy with the trivial example that we used for motivation – iterative solution of a single linear equation – the rate of convergence of the matrix iteration problem is determined by the "size" of the iteration matrix. The smaller the norm of Q, the faster will the method converge. The aim of improved methods for solving linear equations iteratively is to reduce the norm of the iteration matrix. In particular, because it is the maximum eigenvalue of the iteration matrix that controls convergence, "good" iterative methods will have iteration matrices with small eigenvalues.

7.5. Iterative methods for finite difference equations

Recall the example problem from Chapter 6.3 (for the Laplace equation).

```
M_diag=sparse(1:21,1:21,-4,21,21);
L_diag1=sparse(2:21,1:20,1,21,21);
L_diag2=sparse(8:21,1:14,1,21,21);
L_diag1(8,7)=0; L_diag1(15,14)=0;
A=M_diag+L_diag1+L_diag2+L_diag1'+L_diag2';
```

Remember that these steps give the blocked matrix with -4's on the main diagonal and 1's on a few off-diagonals.

```
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     0     0
     1     0     0     0     0     0     0    -4     1
     0     1     0     0     0     0     0     1    -4
```

The right-hand vector for this example problem has three values of -100.

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

Now rather than use the *MATLAB* backslash command to get the solution, suppose we use the Jacobi iteration approach. The matrix D is just what we defined as `M_diag` in the *MATLAB* statements above. The $L+U$ portion is everything else; we can define them in *MATLAB* as

```
L=L_diag1+L_diag2; U=L'; LnU=L+U;
```

To get D^{-1}, just write `Dinv=inv(M_diag)`. The iteration matrix is `Q=-Dinv*LnU`. We solve the equations using a loop that ends when the fractional change in h is less than 10^{-3}.

```
convcrit=1e9;
h_old=ones(21,1);
kount=0;
while convcrit>1e-3
   kount=kount+1;
   h=Q*h_old+Dinv*b;
   convcrit=max(abs(h-h_old)./h);
   h_old=h;
end
```

It takes 40 iterations to reduce the relative change in the iterative values to $< 10^{-3}$. Compare the results from this iteration (below) with the results in Chapter 6.3 obtained with the "\" command.

```
output=[h(1:7) h(8:14) h(15:21)]

output =
    0.3519    0.4973    0.3519
    0.9112    1.2865    0.9112
    2.0076    2.8287    2.0076
    4.2929    6.0153    4.2929
    9.1505   12.6501    9.1505
   19.6612   26.2865   19.6612
   43.2090   53.1759   43.2090
```

What about the rate of convergence? A good iteration method will have a norm that is "small". The closer to unity the norm, the slower will be the rate of convergence. For the Jacobi iteration matrix for the example problem, `abs(max(eig(Q)))` gives 0.815. For the small example problem this isn't horrendous, but as the size of the problem gets bigger (the mesh spacing for a fixed area gets finer), the norm of the Jacobi iteration matrix gets close to one very quickly. The method always converges, but progress can be agonizingly slow.

The method known as "successive over-relaxation" (SOR) is the simplest of the useful iterative methods for solving systems of linear equations. The iteration formula is

$$x_{k+1} = S(w)\,x_k + \left(w^{-1}D + L\right)^{-1} b,$$

where the iteration matrix, S, is

$$S(w) = -(w^{-1}D + L)^{-1}\left(U + (1 - w^{-1})D\right).$$

The Greek letter w in the iteration formula is an iteration parameter, chosen to accelerate convergence. The identification of an optimal value for w is not easy for complex problems. Theoretically, the optimal value of w for the Laplace equation is

$$w_{opt} = \frac{2}{1 + \sqrt{1 - r^2(Q)}},$$

where $r(Q)$ is the magnitude of the largest eigenvalue of the Jacobi iteration matrix.

For the example problem, we can calculate w_{opt} using 0.815 for $r(Q)$, a value that can be found using the *MATLAB* `normest` command, `wopt=2/(1+sqrt(1-(normest(Q))^2))`, which gives a value of about 1.27. We can apply the SOR method to the example problem:

```
y=inv(D*(1/wopt)+L);
S=-y*(U+(1-(1/wopt))*D);
convcrit=1e9;
h_old=ones(21,1);
kount=0;
while convcrit>1e-3
   kount=kount+1;
   h=S*h_old+y*b;
   convcrit=max(abs(h-h_old)./h);
   h_old=h;
end
```

For the SOR method with w_{opt} =0.815, the convergence criterion is satisfied after 14 iterations. The results are,

```
output=[h(1:7) h(8:14) h(15:21)]

output =
    0.3529    0.4988    0.3530
    0.9131    1.2893    0.9132
    2.0103    2.8323    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
```

How "big" is the iteration matrix for the SOR method? `abs(max(eig(full(s))))` gives `ans = 0.2671`, which is considerably smaller than the norm of the Jacobi matrix. We can look at the norm as a function of w to get a feel for how the SOR acceleration parameter works.

```
w=1:0.025:2;
```

```
for i=1:max(size(w))
   y=inv(D*(1/w(i))+L);
   S=-y*(U+(1-(1/w(i)))*D);
   rho(i)=abs(max(eig(full(S))));
end
plot(w,rho)
xlabel('SOR iteration parameter')
ylabel('Magnitude of maximum eigenvalue of the iteration matrix')
```

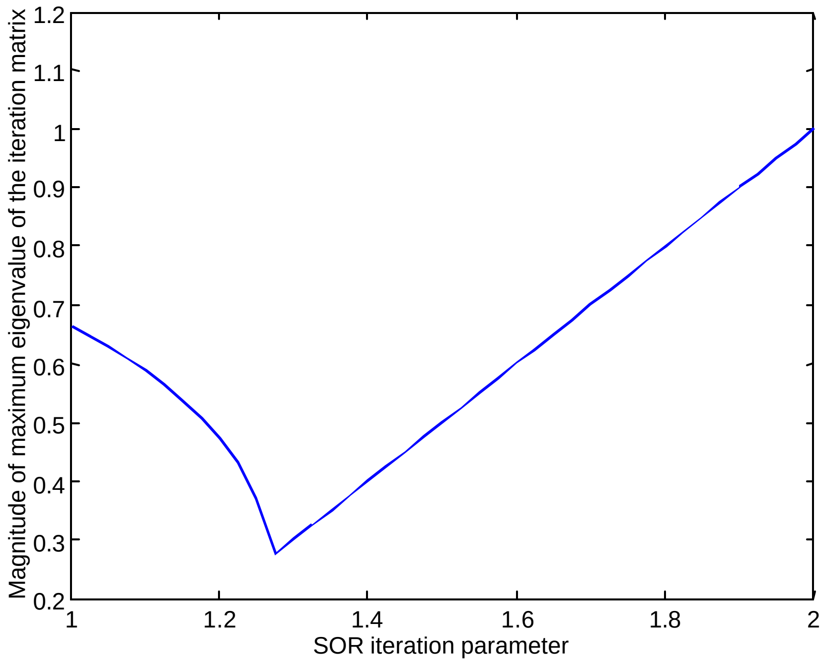


Figure 7.2. Splitting parameter for the SOR example.

Compare this figure with the one we drew for our simple-minded, one-linear-equation example. The analogy isn't perfect by any means, but the outcome is the same – there is an optimum value of an iteration parameter that speeds convergence of the iterative procedure.

The big trick is, of course, to choose a good iteration parameter. Sometimes it is sensible to calculate the maximum eigenvalue of the Jacobi matrix, but that procedure is in no case the "universal" solution. If you need to pursue this topic further, you will have to consult a text on numerical solution of systems of linear equations. Although the text is a bit "long in the tooth" now, you might try Remson et al. (1971) for a start.

The iterative methods that are used most widely today are more efficient (smaller matrix norms) than the very simple methods presented above (including SOR). One of these methods is called the Strongly Implicit Procedure (SIP). MODFLOW [Box 6.2] uses SIP to solve the groundwater flow equations (McDonald and Harbaugh, 1988). The SIP method uses a handful of iteration parameters; choosing them is an art, although the performance of the method does not depend critically on having exact values.

Another relatively new method is the conjugate-gradient method (e.g., Hill, 1990). The conjugate-gradient technique is not only powerful, but the acceleration parameter is calculated as part of the solution so no special tricks are needed to make it work efficiently. The conjugate-gradient method is really an iterative method for finding the minimum of a nonlinear function. Its use in solving sparse linear systems stems from recognizing that the minimum of

$$f(x) = \frac{1}{2}\left|Ax - b\right|^2$$

is unique and occurs for a value of x that satisfies the equation $Ax=b$. *MATLAB* has several built-in conjugate-gradient solvers, including `cgs`, which uses the "conjugate gradients squared" method. The syntax is `x=cgs(A,b)` where $Ax=b$ is the system to be solved (A is an $n\times n$ square matrix and b has length n). Error tolerance and maximum number of iterations can be set as described in "`help cgs`". Another *MATLAB* conjugate-gradient solver is `pcg`, which uses a preconditioned conjugate gradient method. However, for `pcg` the $n\times n$ coefficient matrix A must be positive definite and symmetric, which is often not true of the coefficient matrices in finite difference solutions to partial differential equations.

7.6. Problem

For the problem on the Brookhaven landfill from Chapter 6.7, use the Jacobi, SOR, and conjugate-gradient methods to solve the finite-difference equations. Write your own code for the Jacobi and SOR methods; use *MATLAB*'s `cgs` function for the conjugate-gradient method. Compare the number of iterations needed to achieve a solution using the different methods. (The number of iterations can be set in `cgs`.) Examine the effect of changing the value of the convergence criterion. Examine the effect of changing grid spacing. (Note that *MATLAB's* preconditioned conjugate gradients solver, `pcg`, will not work for this case because the coefficient matrix is not symmetric.)

7.7. References

Hill, M.C., Preconditioned conjugate-gradient 2(PCG2), a computer program for solving ground-water flow equations. U.S. Geological Survey Water-Resources Investigations Report 90-4048, 1990.

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

Remson, I., Hornberger, G.M., and F.J. Molz, *Numerical Methods in Subsurface Hydrology*, 389 pp., Wiley-Interscience, New York, 1971.

Wang, H.F. and M.P. Anderson, *Introduction to Groundwater Modeling: Finite Difference and Finite Element Methods*, 237 pp., W.H. Freeman, San Francisco, 1982.

Box 7.1. L_p norms and circles

The general equation for the L_p norm is

$$\|x\|_p = [x_1^p + x_2^p + x_3^p + \circ\circ\circ + x_n^p]^{1/p}$$

As noted in the text, p=2 gives the "standard" distance; p=1 gives the sum of absolute values; and $p=\infty$ gives the maximum component. Other values of p don't have as intuitive a meaning. One interesting way to think of these norms is in terms of the associated circles. In two dimensions, the line of constant value of a norm is a circle. The L_2 (Euclidean) norm produces the circle we are used to. But other norms also correspond to "circles." We can view these "circles" using the following m-file.

```
function y=isoLp(p)
x=[];y=[];
x=0:0.01:1;
y=(1-x.^p).^(1/p);
plot(x,y,-x,y,'b',x,-y,'b',-x,-y,'b')
axis('square')
```

The figure below shows the resulting "circles" for four values of p.

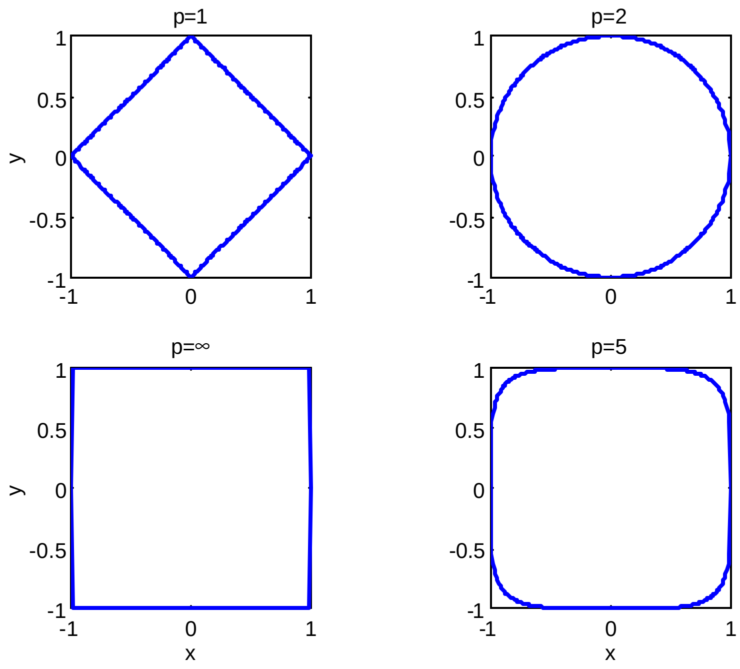


Figure B7.1. "Circles" associated with the L_p norm for several values of p.

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).

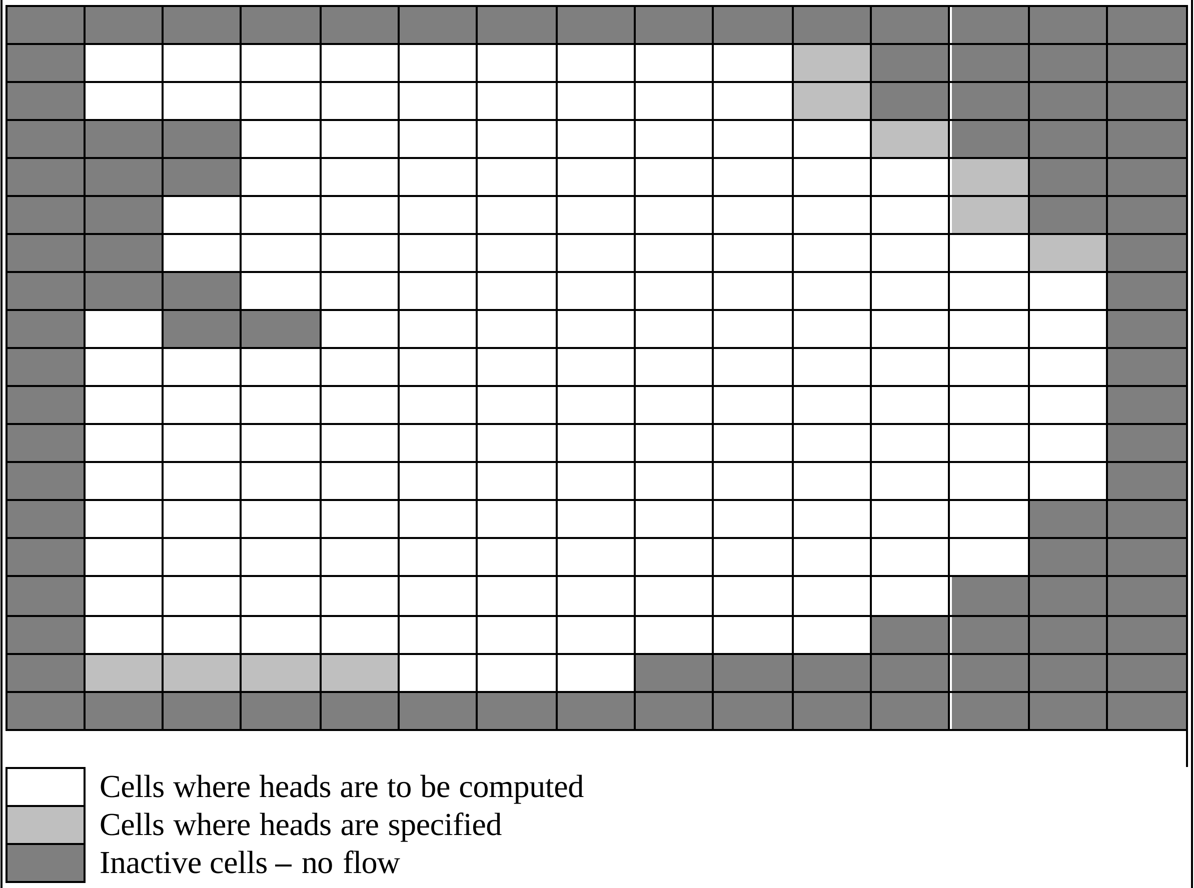

Cells where heads are to be computed
Cells where heads are specified
Inactive cells – no flow

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

CHAPTER 8
Finite Difference Solutions for Transient Problems

Numerical Methods in the Hydrological Sciences: AGU Lecture Notes 1
Copyright 2004 by the American Geophysical Union
doi: 10.1029/001LN10
(Version 2004-08-02)

8. Finite Difference Solutions for Transient Problems

8.1. Background

Hydrological problems often involve time as a variable. The propagation of flood waves through river systems, the decline in water levels in aquifers as water is pumped from wells, the speading of a contaminant through a waterway or aquifer, and the drying of surface layers of soil by evapotranspiration are examples of problems where the variablesof interest – river stage, ground-water head, or soil moisture – vary temporally.

A particularly important type of equation used to describe *transient* (i.e., time-varying) problems in hydrology is the *heat equation* (Chapter 6.1). This equation describes not only heat flow in the natural environment (e.g., see Domenico and Schwartz 1998), but such things as diffusion of chemicals in sediments (e.g., Boudreau 1997), the movement of flood waves in channels using the diffusion analogy (e.g., Bras 1990), and the change in groundwater head in an aquifer (e.g., Fetter 2001).

8.2. Unidirectional flow in an aquifer

To illustrate the numerical methods used to solve the heat equation, consider a horizontal aquifer with constant formation properties, T (transmissivity) and S (storativity)[1]. The head in the aquifer is taken to vary with only one spatial coordinate, x. The equation that describes the variation of head in time and space is then

$$\frac{\P h}{\P t} = \frac{T}{S}\frac{\P^2 h}{\P x^2}. \tag{8.1}$$

Further suppose that the aquifer is bounded by constant-head streams at x=0 and 2 km and that at some initial time the head in the aquifer is elevated at the center of the aquifer relative to the streams because of a recharge event. The head distribution is given by

$$h = 0.1x, \qquad \text{for } 0 \le x \le 1000 \text{ m}$$
$$h = 0.1(2000 - x), \quad \text{for } 1000 \text{ m} \le x \le 2000 \text{ m.}$$

The parameter T/S for the aquifer is 151.5 $m^2\, s^{-1}$. The problem is to determine heads in the aquifer as a function of time assuming that recharge has ceased.

8.3. A forward difference (or *explicit*) approximation

Finite difference approximations for transient problems involve differencing both time and space derivatives. We will adopt the convention of using a "*j*" superscript to denote grid points in time and an "*i*" subscript to denote space grid points. Thus h_i^j denotes head at spatial grid point "*i*" and time grid point "*j*".

[1] For transient conditions, the derivation of the conservation equation shown in Box 6.1 must be adjusted to accommodate changes in time within the control volume. This leads to inclusion of the time derivative of head multiplied by a storage coefficient. See a text on hydrogeology, e.g., Fetter (2001), for more details.

One finite-difference approximation to equation (8.1) is formed by using the heads at time j in the approximation of the right-hand side:

$$\frac{h_i^{j+1} - h_i^j}{\Delta t} = \frac{T}{S}\left(\frac{h_{i+1}^j - 2h_i^j + h_{i-1}^j}{(\Delta x)^2}\right). \tag{8.2}$$

The way that we solve time-varying problems is to start at an initial time when conditions are known and then calculate conditions one time step into the future. Conditions at this step in the future are then "known" and one can proceed to calculate conditions at $2\Delta t$. And so forth. Thus, in general terms, conditions at time "j" are known and conditions at time "j+1" are to be calculated. Because the heads on the right side of equation (8.2) are at the "j" time, the approximation to the time derivative is a forward-difference approximation.

The unknowns in equation (8.2) form a vector $\left[h_1^{j+1}\ h_2^{j+1}\ h_3^{j+1} \ldots\ h_n^{j+1}\right]'$. We can write equation (8.2) in matrix-vector form as

$$h^{j+1} = Wh^j \tag{8.3}$$

where W has diagonal entries equal to $(1-2\left(\frac{T\Delta t}{S(\Delta x)^2}\right))$ and sub- and super-diagonal entires equal to $\frac{T\Delta t}{S(\Delta x)^2}$. The method is *explicit* because each component equation of the set of equations (8.3) is independent of the others. The equations are not linked directly, so no solution of a system of equations is required. Equation (8.3) requires only matrix multiplication.

Let's look at how the solution works. We choose a spacing in the x direction of 250 m and a time step of 206 s $\left[= {(\Delta x)^2 S}/{2T}\right]$.

```
% Explicit solution to example one-D groundwater problem
% Set parameter values and grid spacing
dx=250; x=250:dx:1750; ToverS=151.5; dt=dx^2/(2*ToverS);
h_old=[0.1*x(1:4) 0.1*(2000-x(5:7))]';  %Initial head values
alpha=ToverS*dt/dx^2;   %Transmissivity divided by storage coefficient
M_diag=sparse(1:7,1:7,1-2*alpha,7,7);  %Diagonal of W matrix
L_diag=sparse(2:7,1:6,alpha,7,7);  %Off-diagonal of W matrix
W=M_diag+L_diag+L_diag';   %W matrix
hh=zeros(7,10);  %Pre-allocate matrix for solution
for j=1:10   %Do ten time steps
      h_new=W*h_old;          % The explicit solution
      hh(:,j)=h_new;
      h_old=h_new;
end
hh'
ans =
   25.0000   50.0000   75.0000   75.0000   75.0000   50.0000   25.0000
   25.0000   50.0000   62.5000   75.0000   62.5000   50.0000   25.0000
   25.0000   43.7500   62.5000   62.5000   62.5000   43.7500   25.0000
   21.8750   43.7500   53.1250   62.5000   53.1250   43.7500   21.8750
   21.8750   37.5000   53.1250   53.1250   53.1250   37.5000   21.8750
   18.7500   37.5000   45.3125   53.1250   45.3125   37.5000   18.7500
   18.7500   32.0312   45.3125   45.3125   45.3125   32.0312   18.7500
   16.0156   32.0312   38.6719   45.3125   38.6719   32.0312   16.0156
   16.0156   27.3438   38.6719   38.6719   38.6719   27.3438   16.0156
   13.6719   27.3438   33.0078   38.6719   33.0078   27.3438   13.6719
```

We can do the comparison with the analytical solution as well (Figure 8.1) and show that the approximate nature of the numerical solution is evident with the relatively coarse spatial grid spacing used in this example.[2]

```
% Analytical solution is an infinite Fourier series; see, e.g.,
% Carslaw and Jaeger 1959
time=0:dt:10*dt; fdtime=time(2:11);  % set time vector for solution
x1=250; x2=750;  % choose values of x for solution
n=0:25;   % use 25 terms in Fourier series solution
h250=zeros(1,11); h750=zeros(1,11);  % preallocate solution vectors
% compute the exact solution for 11 times
for j=1:11
     h250(j)=800*sum((ones(size(n))./(pi^2*(2*n+1).^2)).*cos((2*n+1).* …
        pi*(x1/1000-1)/2).*exp(0.3738*(2*n+1).^2*time(j)/1000));
     h750(j)=800*sum((ones(size(n))./(pi^2*(2*n+1).^2)).*cos((2*n+1).*…
        pi*(x2/1000-1)/2).*exp(-0.3738*(2*n+1).^2*time(j)/1000));
end
plot(fdtime,hh(1,:),'or',fdtime,hh(3,:),'or',time,h250,'b',time,h750,'b')
xlabel('Time, seconds')
ylabel('Heads at 250m (lower curve) and 750m (upper curve)')
```

[2] The analytical solution is in the form of an infinite Fourier series. See Carslaw and Jaeger (1959).

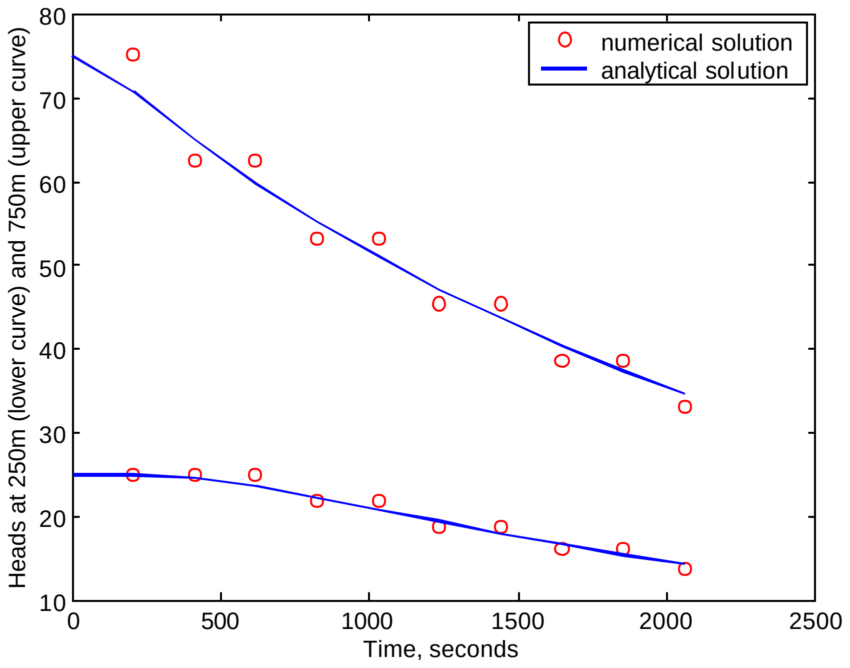


Figure 8.1. Heads calculated using explicit method with the analytical solution.

8.4. Stability problems with the explicit method

The explicit method is attractive because of its simplicity. To be useful, however, the time steps in the explicit method must be kept small. To illustrate the problem, let's look at the solution above for a Δt double that of our original choice, i.e. $\Delta t = (\Delta x)^2 S/T$.

```
% set time and space steps and parameters
dx=250; x=250:dx:1750; ToverS=151.5; dt=dx^2/(ToverS);
alpha=ToverS*dt/dx^2;
% place initial heads in vector h_old
h_old=[0.1*x(1:4) 0.1*(2000-x(5:7))]';
% construct the finite-difference matrix
M_diag=sparse(1:7,1:7,1-2*alpha,7,7);
L_diag=sparse(2:7,1:6,alpha,7,7);
W=M_diag+L_diag+L_diag';
% preallocate solution matrix
hh=zeros(7,10);
for j=1:10                % solve the equations for 10 time steps
      h_new=W*h_old;
      hh(:,j)=h_new;
      h_old=h_new;
end
hh(1:4,:)'

ans =
          25          50          75          50
          25          50          25         100
          25           0         125         -50
         -25         150        -175         300
         175        -350         625        -650
        -525        1150       -1625        1900
        1675       -3300        4675       -5150
       -4975        9650      -13125       14500
       14625      -27750       37275      -40750
      -42375       79650     -105775      115300
```

Obviously, something is drastically wrong! The calculated heads are oscillating with absolute values getting ever larger with simulation time. This is a classic example of instability.

The way to study stability problems is to look at how errors (in computation there will *always* be small round-off errors) propagate through the solution. If the true solution to the finite-difference equations is given by H, the best that we can calculate is $H+e$, where e is the error. For one of the individual equations of (8.3), the true computation should be

$$H_i^{j+1} = rH_{i-1}^j + (1-2r)H_i^j + rH_{i-1}^j \tag{8.4}$$

where $r = \dfrac{T\Delta t}{S\left(\Delta x\right)^2}$. Our computation is imperfect, however, and is actually given by the equation

$$H_i^{j+1} + e_i^{j+1} = r\left(H_{i-1}^j + e_{i-1}^j\right) + (1-2\,r)\left(H_i^j + e_i^j\right) + r\left(H_{i+1}^j + e_{i+1}^j\right). \tag{8.5}$$

Subtracting (8.4) from (8.5), we find that the errors are governed by exactly the same equation as are the true heads:

$$e_i^{j+1} = r\left(e_{i-1}^j\right) + (1-2r)\left(e_i^j\right) + r\left(e_{i+1}^j\right). \tag{8.6}$$

A rigorous analysis of errors can be made in a number of ways. It turns out that we can get the right answer using a very simple analysis. We want a way to calculate a bound on the error at time $j+1$. Looking at equation (8.6), we ask, "What is the *worst* thing that could happen?" (It turns out that in computation, the worst thing always does happen.) Clearly, the error at time $j+1$ will be largest if e at the i-1 and i+1 spatial nodes have the opposite sign to the error at node i. Without loss of generality, we can assume that the errors are also equal in magnitude. In that case (8.6) becomes:

$$e_i^{j+1} = \left(1-4r\right)e_i^j. \tag{8.7}$$

Noting that this equation would serve as a bound for all values of j, we find that, after m time steps, the error is related to any initial error by

$$e_i^{j+1} = \left(1-4r\right)^m e_i^0. \tag{8.8}$$

We see that, if (1-4r) > 1 in magnitude, the error will grow as a power of the number of time steps. Such unbounded growth in error is what we mean by instability. The stability criterion for the explicit method is read directly from (8.8).

$$\left|1-4r\right| \le 1, \quad \text{or}$$
$$\mathrm{r} \le \frac{1}{2}.$$

The stability restriction renders the explicit method cumbersome for many applications. For any approximation to be reasonably good, Δx must be kept small. Because Δt must be scaled by $(\Delta x)^2$, each time step is very small and a large number of computational steps may be required. If the stability restriction on the time step could be removed, Δt could be selected independently of Δx and the computation could be much more efficient. A modification to the finite-difference approximation accomplishes just that aim.

8.5. A backward difference (or *implicit*) approximation

To show how to avoid the rather strict stability criterion for explicit methods, let us revisit the finite difference equation (8.2) and see what happens if we use the unknowns at the j+1 time step to approximate the spatial derivative.

$$\frac{h_i^{j+1}-h_i^j}{\Delta t}=\frac{T}{S}\left(\frac{h_{i+1}^{j+1}-2h_i^{j+1}+h_{i-1}^{j+1}}{\left(\Delta x\right)^2}\right). \tag{8.9}$$

With this approximation, we are "looking backward" in time from the time step at which we take the spatial derivative, so the finite difference approximation in time is a backward difference. With this equation, if we place the "unknowns" (all heads at the j+1 time step) on the left-hand side of the equation and all the "knowns" (the heads at time j) on the right-hand side, we get

$$Ah^{j+1}=h^j, \tag{8.10}$$

where the diagonal elements of A are (1+2r) and the sub- and super-diagonal elements are -r. The solution is implicit because a system of matrix-vector equations must be solved.

How about the stability? The same type of analysis that we used for the explicit equation gives the equation for the error as

$$-re_{i-1}^{j+1}+(1+2r)e_i^{j+1}-re_{i+1}^{j+1}=e_i^j.$$

The absolute worst that can happen in this case is for the errors at time j+1 for the i-1 and i+1 nodes to be equal to the error for node i. Then the error at the j+1 time step would be equal to the error at the j time step. No explosive growth. No instability. Even under the worst conditions. Thus, the implicit method is unconditionally stable for the ground-water equation. We can use any time steps that we want. Of course, large time steps will not result in a very accurate solution to the equation, but the solution will be stable.

How does the implicit method work in practice? The modification to the *MATLAB* code for the explicit method isn't very difficult.

```
% set time and space steps and parameters
dx=250; x=250:dx:1750; Tover S=151.5; dt=dx^2/(2*ToverS);
alpha=ToverS*dt/dx^2;
```

```
% place initial heads in vector h_old
h_old=[0.1*x(1:4) 0.1*(2000-x(5:7))]';
% construct the finite-difference matrix
M_diag=sparse(1:7,1:7,1+2*alpha,7,7);
L_diag=sparse(2:7,1:6,-alpha,7,7);
A=M_diag+L_diag+L_diag';
% preallocate solution matrix
hh=zeros(7,10);
for j=1:10                    % solve the equatios for 10 time steps
      h_new=A\h_old;          % solution uses the backslash command
      hh(:,j)=h_new;
      h_old=h_new;
end
hh'

ans =
   24.7423   48.9691   71.1340   85.5670   71.1340   48.9691   24.7423
   24.1471   47.1038   66.3301   75.9486   66.3301   47.1038   24.1471
   23.2616   44.7521   61.5390   68.7438   61.5390   44.7521   23.2616
   22.1756   42.1791   57.0369   62.8903   57.0369   42.1791   22.1756
   20.9760   39.5530   52.8777   57.8840   52.8777   39.5530   20.9760
   19.7308   36.9713   49.0482   53.4661   49.0482   36.9713   19.7308
   18.4872   34.4872   45.5190   49.4926   45.5190   34.4872   18.4872
   17.2755   32.1276   42.2606   45.8766   42.2606   32.1276   17.2755
   16.1137   29.9039   39.2467   42.5616   39.2467   29.9039   16.1137
   15.0115   27.8186   36.4551   39.5083   36.4551   27.8186   15.0115
```

The numerical solution is "smoother" for the implicit method than for the explict method (compare Figure 8.2 with Figure 8.1).

To illustrate that the solution is stable for larger values of r, rerun the example with r=1, the value that led to explosive instability with the explicit method. As the results on the next page show, the calculated heads "behave" even when r exceeds 0.5. It is the ability to take relatively large time steps that leads to the preference for the use of implicit methods in finite-difference solutions of ground-water-flow problems.

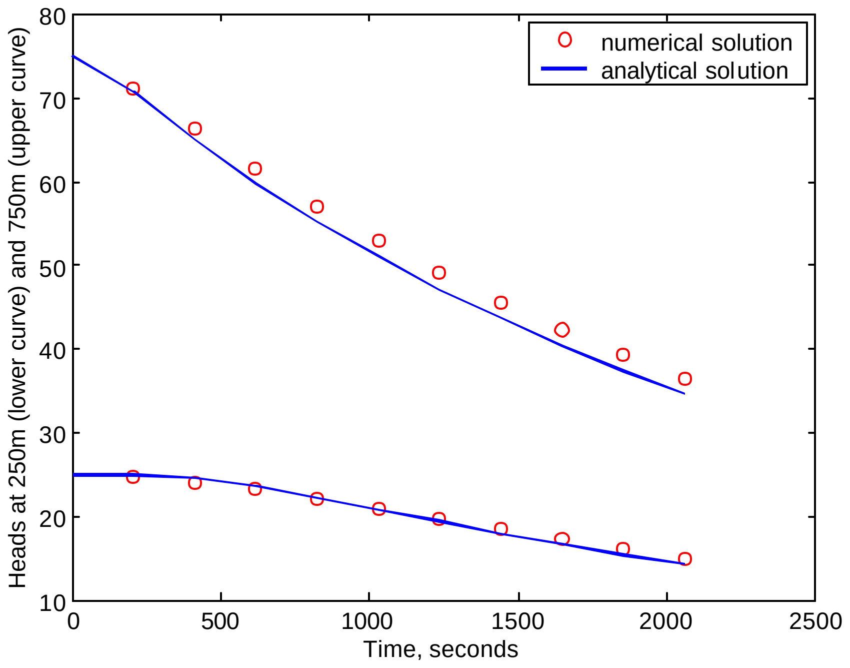


Figure 8.2. Heads calculated using implicit method with the analytical solution.

```
dx=250; x=250:dx:1750; ToverS=151.5; dt=dx^2/(ToverS);
h_old=[0.1*x(1:4) 0.1*(2000-x(5:7))]';
alpha=ToverS*dt/dx^2;
M_diag=sparse(1:7,1:7,1+2*alpha,7,7);
L_diag=sparse(2:7,1:6,-alpha,7,7);
A=M_diag+L_diag+L_diag';
hh=zeros(7,10);
for j=1:10
      h_new=A\h_old;
      hh(:,j)=h_new;
      h_old=h_new;
end
hh'

ans =
   23.9362   46.8085   66.4894   77.6596   66.4894   46.8085   23.9362
   22.0349   42.1684   57.6618   64.3278   57.6618   42.1684   22.0349
   19.7685   37.2706   49.8750   54.6926   49.8750   37.2706   19.7685
   17.4690   32.6386   43.1762   47.0150   43.1762   32.6386   17.4690
   15.3076   28.4537   37.4150   40.6150   37.4150   28.4537   15.3076
   13.3526   24.7502   32.4444   35.1679   32.4444   24.7502   13.3526
   11.6191   21.5048   28.1451   30.4860   28.1451   21.5048   11.6191
   10.0979   18.6745   24.4207   26.4425   24.4207   18.6745   10.0979
    8.7699   16.2120   21.1915   22.9418   21.1915   16.2120    8.7699
    7.6140   14.0721   18.3904   19.9075   18.3904   14.0721    7.6140
```

8.6. The g method

We can "mix and match" the explicit and implicit methods. That is, we can approximate the spatial derivative as *g* times the finite-difference approximation for the spatial derivative at the *j*+1 time step plus (1-*g*) times the approximation at time *j*. This results in a set of equations

$$Gh^{j+1} = Hh^{j},$$

where the diagonal elements of *G* are (1+2*rg*) and the sub- and super-diagonal elements are -*rg*, and the diagonal elements of *H* are [1-2*r*(1-*g*)] and the sub- and super-diagonal elements are *r*(1-*g*). When *g*=0, we recover the explicit method; if *g*=1, we get the implicit method.

For *g*=1/2, this method is known as the Crank-Nicolson method. Recall that the forward- and backward-difference approximations to a first derivative in time have truncation errors O(Δ*t*) whereas the central-difference approximation has truncation error O((Δ*t*)^2). The Crank-Nicolson approximation is, in a sense, a central-difference approximation in time. In fact, one can show that the truncation error for this case is O((Δ*t*)^2).

8.7. Flow in an unsaturated soil

Problems involving flow in an unsaturated soil generally are attacked by solving the Richards equation [Box 8.1]. Such problems are quite difficult because the relationships between matric potential, *y*, and moisture content, *q*, and between hydraulic conductivity, *K*, and moisture content are highly nonlinear. The material in the following section is meant to serve as an introduction to how such problems can be approached.

Consider the problem of vertical flow of soil moisture. The Richards equation for this problem can be written (with "z" measured vertically downward from the surface)

$$\frac{\P q}{\P t} = \frac{\P}{\P z}\left[K(q)\frac{\P y}{\P z}\right] - \frac{\P K(q)}{\P z}. \qquad (8.11)$$

There are a number of ways for expressing the dependence of *y* and *K* on *q*. One of these is as follows.

$$s = \frac{q}{n}, \quad \text{where } n \text{ is the porosity;}$$

$$y\ (s) = y_{sat} s^{-1/m}, \quad \text{where } m \text{ is a parameter;}$$

$$K(s) = K_{sat} s^{c}, \quad \text{where } c \text{ is a parameter.}$$

where K_{sat} and y_{sat} are the values of these parameters for *s*=1. Given these forms for the nonlinear relationships, equation (8.11) can be written

$$\frac{\P q}{\P t} = \frac{\P}{\P z}\left[K_{sat}\left(\frac{q}{n}\right)^{c}\left(\frac{-y_{sat}}{m}\right)\left(\frac{1}{n}\right)\left(\frac{q}{n}\right)^{-1-1/m}\frac{\P q}{\P z}\right] - \left(K_{sat}c\left(\frac{1}{n}\right)\left(\frac{q}{n}\right)^{c-1}\right)\frac{\P q}{\P z}.$$

The equation is obviously nonlinear. There are no "standard" ways to solve such equations. Suffice it to say that solution is often achieved with considerable difficulty.

Solutions to nonlinear equations can be obtained by "linearizing" the nonlinear equations and accepting the solution to the linear equations as an approximation to the solution to the nonlinear equations. Often, iterative methods are used. One non-iterative approach for solving the nonlinear finite-difference soil moisture equations is a "predictor-corrector" method. In predictor-corrector methods, the nonlinear terms are taken out of the equations by substituting known values in these terms. The basic idea behind predictor-corrector methods is to eliminate the nonlinear terms by approximating them with values at the *j*th time level, solve these equations to get a "predicted" value of the dependent variable at, say, time level *j*+1/2, use these "predicted" values to linearize the nonlinear terms in the original equation, and obtain "corrected" values by solving these equations.

To implement a predictor-corrector method for the soil-moisture equation, it is rewritten as:

$$\frac{\P^2 q}{\P z^2} = \frac{1}{y}\frac{\P q}{\P t} - \left[\frac{u}{y}\frac{\P q}{\P z} - \frac{w}{y}\right]\frac{\P q}{\P z}$$

where y, u, and w are functions of moisture content:

$$y = K\frac{dy}{dq}, \quad w = \frac{dK}{dq}, \text{ and } u = \frac{dK}{dq}\frac{dy}{dq} + K\frac{d^2y}{dq^2}.$$

Note that the y, w, and u terms are the nonlinear terms in the equation; if these were *known* and constant (i.e., not functions of q), the soil-moisture equation would be linear. The equation is solved from t^{j} to t^{j+1} in two steps. First a step (the "predictor") is made to an intermediate point half way between t^{j} and t^{j+1}. Then, using the results from this half step, the solution is advanced to time t^{j+1} (the "corrector"). The predictor equation is:

$$\frac{q_{i-1}^{j+1/2} - 2q_i^{j+1/2} + q_{i+1}^{j+1/2}}{\Delta z^2} = \frac{1}{y_i^j}\frac{q_i^{j+1/2} - q_i^j}{\Delta t/2} - \left[\frac{u_i^j}{y_i^j}\left(\frac{q_{i+1}^j - q_{i-1}^j}{2\Delta z}\right) - \frac{w_i^j}{y_i^j}\right]\left(\frac{q_{i+1}^j - q_{i-1}^j}{2\Delta z}\right)$$

and the corrector equation is:

$$\frac{1}{2}\left[\frac{q_{i-1}^{j+1} - 2q_i^{j+1} + q_{i+1}^{j+1}}{\Delta z^2} + \frac{q_{i-1}^{j} - 2q_i^{j} + q_{i+1}^{j}}{\Delta z^2}\right] =$$

$$\frac{1}{y_i^j}\frac{q_i^{j+1} - q_i^j}{\Delta t} - \left[\frac{u_i^{j+1/2}}{y_i^{j+1/2}}\left(\frac{q_{i+1}^{j+1/2} - q_{i-1}^{j+1/2}}{2\Delta z}\right) - \frac{w_i^{j+1/2}}{y_i^{j+1/2}}\right]\left(\frac{1}{2}\right)\left(\frac{q_{i+1}^{j+1} - q_{i-1}^{j+1}}{2\Delta z} + \frac{q_{i+1}^{j} - q_{i-1}^{j}}{2\Delta z}\right)$$

The predictor uses an implicit method covering a time step $\Delta t/2$, linearized by approximating y,w, and u at the *j*th time level. The corrector is a Crank-Nicholson method covering the full time step – level *j* to level *j*+1 – and linearized by approximating y, w and u using the results from the predictor, i.e., the *j*+1/2 values. (See Remson et al., 1971 for details.)

An implementation of the predictor-corrector in *MATLAB* may be helpful. Consider the problem of infiltration into an initially dry sandy loam soil (K_s=3.4 x 10^{-3} cm s^{-1}, y_s=-25 cm, n=0.25, m=5.4, c=3.4). Moisture content at the start of the infiltration event is equal to 0.10

everywhere. The moisture content at the soil surface is instantaneously raised to 0.25 and held there (e.g., by applying a pond of water in an infiltrometer). The time evolution of the moisture profile can be studied by integrating the Richards equation numerically.

```
% 3 function files precede the main code
function y=Kdpsi(param,theta)    % function "y"
Ks=param(1); c=param(2); n=param(3); psis=param(4); m=param(5);
y=(Ks*(-psis/m)*(theta/n).^(c-1-1/m))/n;
function w=dk(param,theta)       % function "w"
Ks=param(1); c=param(2); n=param(3); psis=param(4); m=param(5);
w=(Ks*c*(theta/n).^(c-1))/n;
function u=dKdpsi(param,theta)      % function "u"
Ks=param(1); c=param(2); n=param(3); psis=param(4); m=param(5);
u1c=Ks*c*(-psis/m);
u2c=Ks*psis*(1/m)*(1+1/m);
u=((u1c+u2c)*(theta/n).^(c-2-1/m))/n^2;
% main predictor-corrector loop (using the y, w, and u functions above)
Ks=3.4e-3; psis=-25; n=0.25; m=3.3; c=3.6;   %Values of parameters
param=[Ks c n psis m]; nz=30;
dz=100/nz; dt=60;                        %set time and space steps
ysat=-Ks*psis/m; wsat=Ks*c;
theta_old=ones(nz+1,1)*0.1;
theta_old(1)=0.25;
output=zeros(15,nz+1)';
time=zeros(19,1);
outkount=1;
for j=1:75                          % time loop
% predictor to advance one-half of the time step
   y=Kdpsi(param,theta_old(2:nz));
   w=dK(param,theta_old(2:nz));
   u=dKdpsi(param,theta_old(2:nz));
   bracketterm=((u./y).*(theta_old(3:nz+1)-theta_old(1:nz-1))/(2*dz)-w./y);
   mterm=-2*ones(nz-1,1)/dz^2-2*ones(nz-1,1)./(y*dt);
   M_diag=sparse(1:nz-1,1:nz-1,mterm,nz-1,nz-1);
   Lterm=ones(nz-1,1)./dz^2;
   Uterm=ones(nz-1,1)./dz^2;
   L_diag=sparse(2:nz-1,1:nz-2,Lterm(2:nz-1),nz-1,nz-1);
   U_diag=sparse(1:nz-2,2:nz-1,Uterm(1:nz-2),nz-1,nz-1);
   A=M_diag+L_diag+U_diag;
   rhs=-2*theta_old(2:nz)./(y*dt)-(bracketterm/(2*dz)).*...
      (theta_old(3:nz+1)-theta_old(1:nz-1));
   rhs(1)=rhs(1)-low(1)*theta_old(1);
   rhs(nz-1)=rhs(nz-1)-up(nz-1)*theta_old(nz+1);
   soln=A\rhs;
   thetahalf=[theta_old(1);soln;theta_old(nz+1)];
% corrector to advance to the next time
   y=Kdpsi(param,thetahalf(2:nz));
   w=dK(param,thetahalf(2:nz));
   u=dKdpsi(param,thetahalf(2:nz));
   bracketterm=((u./y).*(thetahalf(3:nz+1)-thetahalf(1:nz-1))/(2*dz)-w./y);
   Mterm=-ones(nz-1,1)/dz^2-ones(nz-1,1)./(y*dt);
   M_diag=sparse(1:nz-1,1:nz-1,d,nz-1,nz-1);
   Lterm=ones(nz-1,1)./(2*dz^2)-bracketterm./(4*dz);
   Uterm=ones(nz-1,1)./(2*dz^2)+bracketterm./(4*dz);
   L_diag=sparse(2:nz-1,1:nz-2,Lterm(2:nz-1),nz-1,nz-1);
   U_diag=sparse(1:nz-2,2:nz-1,Uterm(1:nz-2),nz-1,nz-1);
   A=M_diag+L_diag+U_diag;
   rhs=-theta_old(2:nz)./(y*dt)-(theta_old(1:nz-1)-2*theta_old(2:nz)...
      +theta_old(3:nz+1))/(2*dz^2)-bracketterm.*(theta_old(3:nz+1)...
      -theta_old(1:nz-1))/(4*dz);
   rhs(1)=rhs(1)-low(1)*theta_old(1);
   rhs(nz-1)=rhs(nz-1)-up(nz-1)*theta_old(nz+1);
   soln=A\rhs;
   theta_new=[theta_old(1);soln;theta_old(nz+1)];theta_old=theta_new;
   if rem(j,15)==0
      output(:,outkount)=theta_new;
      time(outkount)=dt*15*outkount; outkount=outkount+1;
   end
end
zz=0:-1/nz:-1; axis([0 0.26 -1 0]);
plot(output(:,1),zz,output(:,2),zz,output(:,4),zz)
xlabel('Volumetric moisture content');ylabel('Depth, meters')
time=time/60;
text(0.11,-0.6,['profiles for times ' num2str(time(1)) ' ' num2str(time(2))
' ' num2str(time(4)) ' minutes'])
```

The results of the computation (Figure 8.3) show that the moisture propagates into the soil with a fairly sharp front. In cases with a sharp front, time and space increments need to be kept small to avoid "glitches" in the solution. (Try the code above with nz=20, instead of 30, for example.) You should keep your skepticism intact when you are generating solutions to complex equations and not believe everything immediately as it comes out of the computer.

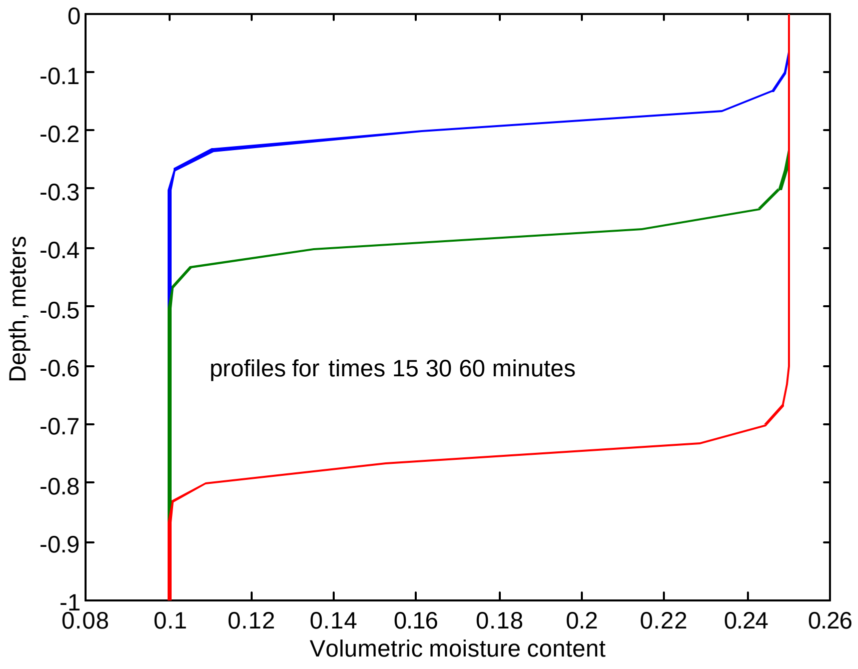


Figure 8.3 Solution to the Richards equation.

8.8. Problems

1. Heat flow in the earth is governed by the processes of conduction and convection. In regions where water is free to move, heat flow in the near surface (the top several hundred meters of the earth's surface) is strongly affected by convection and the analysis of temperature changes is quite complicated. In the arctic, however, permafrost essentially renders water motion meaningless as a heat-flow mechanism. In these areas, conduction is the primary mechanism by which heat is transported in the crust and a relatively simple analysis may be appropriate. The equation for such a problem is:

$$\frac{\P T}{\P t} = \frac{K}{r\ c}\frac{\P^2 T}{\P z^2}$$

where T is temperature, z is depth below the surface, t is time, K is thermal conductivity, c is heat capacity, and r is density.

Consider heat flow in the top 1km of the crust. The surface boundary condition is a specified temperature. The bottom boundary condition (at 1km) is that the upward heat flux, q, be equal to KG_0, where G_0 is the geothermal gradient, about 3°C per 100m. The thermal conductivity of rock and of permafrost is about 0.5 cal m^{-1} s^{-1} °C^{-1} and rc is about 0.5 cal cm^{-3} °C^{-1}.

a) Write a code, using the γ method, to solve the temperature problem for permafrost regions. Explore the effect of changing grid spacing and γ.

b) Use your code to calculate the steady-state temperature profile for a surface temperature of -15°C. Start the computation with T=0°C everywhere.

c) Starting with the steady-state temperature profile as the initial condition, calculate the temperature profile every decade under conditions of a steadily increasing surface temperature at a rate of 3.5°C per century.

d) Mann et al. (1998) suggest that global surface temperature remained relatively steady for several centuries prior to the 20th century and then the temperature rose at a rate of about 0.5°C per century. Of course surface temperature trends at any locale can depart from the global mean trend. The table below gives measured temperatures from a borehole on the north slope of Alaska in 1984. Using your code, offer an interpretation of these data. By asking you to use your code, the intention is that you should be quantitative in your answer. You will want to run your code emphasizing the top tens of meters of the temperature profile – otherwise the fine detail of the changing temperature profile may be obscured by the coarse spatial discretization. [After you have finished this problem, you may want to look at the article by Lachenbruch and Marshall (1986) which is the source for these data. For a more up-to-date discussion of the analysis of borehole temperature profiles relative to climate, see Pollack and Huang (1998). For data from boreholes around the world, see http://www.ngdc.noaa.gov/paleo/borehole/borehole.html. Finally, in case you are prone to accept the results of the analysis of borehole temperatures uncritically, see Mann and Schmitt (2003) for a discussion of difficulties associated with inferring climate change from borehole temperature profiles.]

Depth below surface, meters	Temperature, degrees C
700	11.99
600	9.00
500	5.99
400	3.04
300	-0.01
200	-3.00
150	-4.49
125	-5.24
100	-5.92
90	-6.11
80	-6.37
75	-6.46
70	-6.50
65	-6.60
60	-6.69
55	-6.71
50	-6.75
45	-6.76
40	-6.78
35	-6.73

2. Explore the solution of the Richards equation for unsaturated flow. How are results sensitive to the time and space steps? How does the infiltration front progress for various soil types? (The values below are from Bras, 1990.)

SOIL TYPE	K_{sat} (cm s^{-1})	y_{sat} (cm)	n	m	c
Clay	3.4 x 10-5	-90	0.45	0.44	7.5
Silty loam	3.4 x 10-4	-45	0.35	1.2	4.7
Sand	8.6 x 10-3	-15	0.2	5.4	3.4

8.9. References

Boudreau, BP., *Diagenetic Models and their Implementation*, 414 pp., Springer-Verlag, Berlin, 1997.

Bras, R.L., *Hydrology*, 643 pp., Addison-Wesley, Reading, MA, 1990.

Carslaw, H.S. and J.C. Jaeger, *Conduction of Heat in Solids*, 2nd ed., 510 pp, Clarendon Press, Oxford, 1959.

Domenico, P.A. and F.W. Schwartz, *Physical and Chemical Hydrogeology*, 2nd ed., 506 pp., Wiley, New York, 1998.

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

Lachenbruch, A.H. and B.V. Marshall, Changing climate: geothermal evidence from permafrost in the Alaskan Arctic, *Science, 234*: 689-696, 1986.

Mann, ME, Bradley, RS, and MK Huges, Global-scale temperature patterns and climate over the past six centuries. *Nature, 392*: 779-787. 1998.

Mann ME and GA Schmidt, Ground vs. surface air temperature trends: Implications for borehole surface temperature reconstructions, *Geophys. Res. Let. 30*: art. no. 1607, 2003.

Pollack, HN and S. Huang, Underground temperatures reveal changing climate, *Geotimes, 43*:16-19, 1998.

Remson, I., Hornberger, G.M., and F.J. Molz, *Numerical Methods in Subsurface Hydrology*. 389pp., Wiley-Interscience, New York, 1971.

Box 8.1. Equation for flow in an unsaturated soil

Water in the subsurface can be conveniently divided into *groundwater*, or water under pressure > 0, and *soil moisture*, or water under pressure < 0. That is, generally speaking, soil moisture is water above the water table (the surface defined by $p = 0$) that is under "tension" (under pressure less than atmospheric).

Water in the unsaturated zone, like water in the saturated zone, moves down a gradient in *head*. For soil moisture, the total head, h, is taken as the sum of the head due to gravity, $-z$ (for the z axis directed downward), and capillary pressure head or matric head, y.

$$h = y - z.$$

Matric head is a function of the moisture content. That is, in a very dry soil, capillary (and other) forces hold the water very strongly (high y) while in a moist soil the water is held less strongly (lower y). Matric head also depends on pore size, which generally scales with grain size, so that finer-grained sediment or soil has a larger matric head at a given level of saturation than coarse-grained soils . The relationship between matric head, y, and moisture content, q, for a given soil is known as the *matric characteristic*.

Darcy's law is used to describe the flow of water in the unsaturated zone. The law has the same form as for flow in saturated soils – specific discharge is proportional to the gradient in total head – but in the unsaturated zone, the hydraulic conductivity is a function of moisture content: $K=K(q)$. Darcy's law is then written

$$q = -K(q)\frac{dh}{dl}$$

where "l" represents a distance variable in the direction of flow.

For many applications we are concerned with the flow of soil moisture in the vertical, or "z" direction. In this case Darcy's law is (for z measured downward)

$$q_z = -K(q)\frac{dh}{dz} = -K(q)\frac{d}{dz}(y - z)$$

$$q_z = -K(q)\left[\frac{dy}{dz} - 1\right]$$

For the case of vertical flow of water, the appropriate continuity equation [Box 6.1] is:

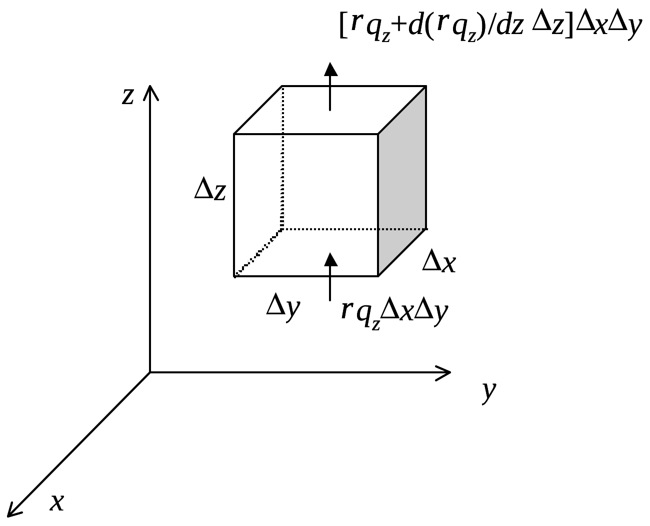
$$\frac{\partial q}{\partial t} = -\frac{\partial q_z}{\partial z}$$

The left side of this equation represents the rate of change of mass in a small control volume, and the right side is the difference between the inflow rate and the outflow rate, each expressed on a per unit volume basis. Combining the continuity equation with Darcy's law results in the *Richards equation*.

$$\frac{\P\, q}{\P\, t} = \frac{\P}{\P\, z}\left[K(q)\frac{\P\, y}{\P\, z}\right] - \frac{\P\, K(q)}{\P\, z}.$$

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):

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

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

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

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,

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

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

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

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.

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

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.

$$\frac{\partial^2 h}{\partial x^2}+\frac{\partial^2 h}{\partial y^2}+\frac{\partial^2 h}{\partial z^2}=0 \qquad \text{(B6.1.7)}$$

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

$$b\frac{\partial q_x}{\partial x}+b\frac{\partial q_x}{\partial x}=0 \qquad \text{(B6.1.8)}$$

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

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

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:

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

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

$$\frac{\partial^2 h}{\partial x^2}+\frac{\partial^2 h}{\partial y^2}=0 \qquad \text{(B6.1.11)}$$

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.

CHAPTER 9

Finite Difference Methods for Transport Equations

Numerical Methods in the Hydrological Sciences: AGU Lecture Notes 1
Copyright 2004 by the American Geophysical Union
doi: 10.1029/001LN11
(Version 2004-08-02)

9. *Finite Difference Methods for Transport Equations*

9.1. Background

In Chapter 8, we considered problems related to time-dependent diffusion of quantities such as heat, solutes, and head. In those problems, diffusion was the only process responsible for the flux or transport of the quantity of interest. However, a solute in a moving fluid not only will be transported by diffusion, but will also be advected with the fluid. In this case both transport processes – advection and diffusion – must be included in the equation for the time-dependent change in concentration,

$$\frac{\partial c}{\partial t}+u\frac{\partial c}{\partial x}=D\frac{\partial^2 c}{\partial x^2} \qquad (9.1)$$

where *u* is the advection velocity and *D* is the diffusion or dispersion coefficient. (Vertical and transverse variations in flow velocity increase rates of mixing compared to molecular or turbulent diffusion. The increased rate of mixing is commonly parameterized by a dispersion coefficient. See, e.g, Fischer et al., 1979 or Hemond and Fechner-Levy, 2000.) Equation (9.1) is commonly referred to as the advection-diffusion or advection-dispersion equation (or simply the transport equation). Here we'll refer to (9.1) as the advection-dispersion equation.

9.2. Numerical dispersion

When considering numerical solutions to the advection- dispersion equation, it is helpful to non-dimensionalize equation (9.1) using the dimensionless parameters $\hat{t}$ = *tu/L*, and $\hat{x}$ = *x*/*L*,

$$\frac{\partial c}{\partial \hat{t}}+\frac{\partial c}{\partial \hat{x}}=\frac{1}{\mathbf{Pe}}\frac{\partial^2 c}{\partial \hat{x}^2} \qquad (9.2)$$

where *L* is a length scale. The dimensionless coefficient on the right-hand-side of the equation is the Peclet number, **Pe** = *uL*/*D.* The Peclet number represents the ratio of the time scale for advection to the time scale for diffusion or dispersion. When the Peclet number is large, the term on the right can be neglected, i.e., dispersion is relatively unimportant. When the Peclet number is small, transport is dominantly by dispersion.

To illustrate one of the difficulties associated with solving equations having the form of the advection-dispersion equation numerically, we first consider the equation for large Peclet numbers. In this case the dispersion term is negligible compared to the advection term and the equation we want to solve is:

$$\frac{\partial c}{\partial \hat{t}}+\frac{\partial c}{\partial \hat{x}}=\frac{\partial c}{\partial t}+u\frac{\partial c}{\partial x}=0.$$

Let *c* be the concentration of some tracer in a stream flowing at velocity *u*. The problem is to describe the downstream transport of the tracer when it is injected at location *x* = 0 such that the concentration in the stream at the injection point is held at 1 unit. That is, initially *c* is equal to zero everywhere, then *c* = 1 at *x* = 0 for all subsequent time. We can see intuitively what the solution to the problem is: the concentration "front" (c = 1 on the upstream side and c = 0 on the downstream side) propagates downstream at speed u.

The advection equation provides an easy demonstration that apparently sensible finite-difference schemes do not always work. Take the straightforward explicit representation of the (dimensional version of the) advection equation:

$$c_i^{j+1} = c_i^j - \left(\frac{u\Delta t}{2\Delta x}\right)\left(c_{i+1}^j - c_{i-1}^j\right) \tag{9.3}$$

This benign-looking method is actually *unconditionally* unstable! That is, there are no values of the time and space steps, no matter how small, that will make this method stable. You can do the simple "worst-case" stability analysis (Chapter 8.3) to show this.

The Lax scheme often is used to solve equations in which the advective term is dominant. To use this scheme, we replace the c_i^j on the right-hand side of the finite-difference equation above with $\left(c_{i+1}^j + c_{i-1}^j\right)/2$.

$$c_i^{j+1} = \left(\frac{c_{i+1}^j + c_{i-1}^j}{2}\right) - \left(\frac{u\Delta t}{2\Delta x}\right)\left(c_{i+1}^j - c_{i-1}^j\right), \text{ or,}$$

$$c_i^{j+1} = \left(\frac{1}{2} + \frac{u\Delta t}{2\Delta x}\right)c_{i-1}^j + \left(\frac{1}{2} - \frac{u\Delta t}{2\Delta x}\right)c_{i+1}^j$$

Now if you use our simple stability analysis procedure, you will find that the Lax scheme is stable if the following stability criterion (referred to as the Courant condition) is met:

$$\frac{|u|\Delta t}{\Delta x} \le 1.$$

Let's look at the solution to our problem using the Lax scheme.

```
% Lax solution for the advection problem
c=zeros(1,21);       % initial conditions
c(1)=1;            % boundary condition
u=1; dx=2; dt=2;        % set velocity and grid steps
% note: solution for dt=dx/u=2 should preserve sharp front
for i=1:20
   c(2:20)=(0.5+u*dt/(2*dx))*c(1:19)+(0.5-u*dt/(2*dx))*c(3:21);
   cc(i,1:21)=c;
end
% repeat for dt=1.5
c=zeros(1,21);
c(1)=1;
dt=1.5;
for i=1:20
      c(2:20)=(0.5+u*dt/(2*dx))*c(1:19)+(0.5-u*dt/(2*dx))*c(3:21);
   cc2(i,1:21)=c;
end
% plot the profile at t=24 (12 steps for dt=2; 16 for dt=1.5)
z=0:dx:40;
```

```
plot(z,cc(12,:),'b',z,cc2(16,:),'r')
xlabel('Distance from injection point')
ylabel('Reduced concentration')
legend('dt=2','dt=1.5')
```

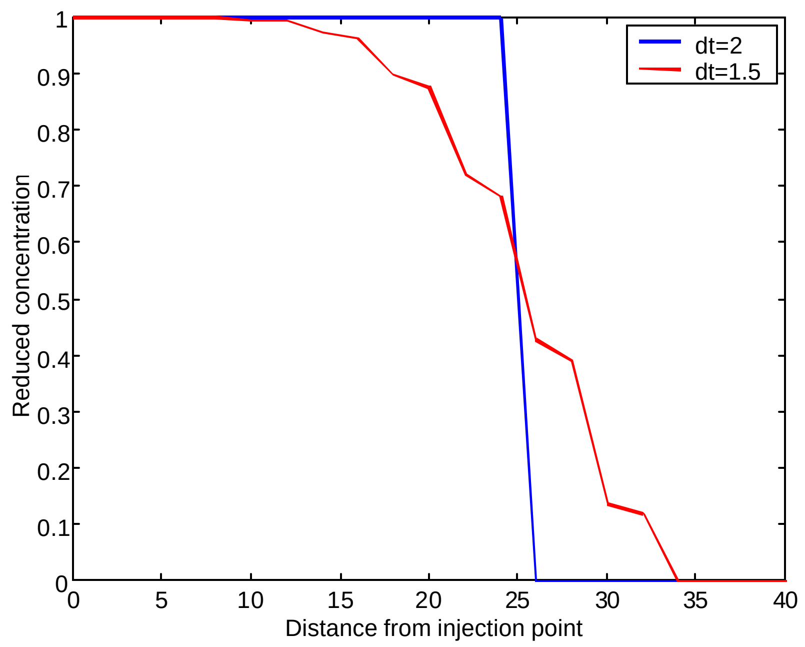


Figure 9.1. Lax solution for a simple advection problem for two time steps

Curiously, the solution for Δt = 2 is better than the solution for Δt = 1.5. What is going on? It turns out that the change we made to ensure stability was not totally benign. Let's consider our modification of equation (9.3) more carefully. The forward-in-time, central-in-space approximation that is unconditionally unstable is:

$$\frac{c_i^{j+1}}{\Delta t} = \frac{c_i^j}{\Delta t} - \left(\frac{u}{2\Delta x}\right)\left(c_{i+1}^j - c_{i-1}^j\right)$$

To achieve stability, we replaced the first term on the right-hand side with an average of the concentrations at the surrounding nodes. It turns out that this is equivalent to adding the term $\frac{1}{2}\frac{(\Delta x)^2}{\Delta t}\left(\frac{c_{i+1}^j - 2c_i^j + c_{i-1}^j}{(\Delta x)^2}\right)$ to the right-hand side of the equation. But note that the term that we had to add is a numerical approximation to a dispersion term equal to $\frac{1}{2}\frac{(\Delta x)^2}{\Delta t}\frac{\partial^2 c}{\partial x^2}$. That is, we introduced a *numerical* dispersion term.

Of course, the dispersion term in a differential equation is not exactly represented by the numerical approximation – recall that we truncated a Taylor series to get the approximation to the second derivative [Chapter 3.2]. The actual differential equation that we are approximating in using the Lax scheme can be written

$$\frac{\partial c}{\partial t}+u\frac{\partial c}{\partial x}=\frac{1}{2}u\Delta x\left(\frac{1}{\mathbf{Cr}}-\mathbf{Cr}\right)\frac{\partial^2 c}{\partial x^2}+\cdots$$

where $\mathbf{Cr}$ is the Courant number ($\mathbf{Cr}=u\Delta t/\Delta x$) and the ellipses (…) represent higher order terms in the equation. Now it is clear why the solution for Δt = 2 is good – the Courant number is exactly unity so the dispersion term is knocked out of the equation above. The solution for Δt = 2 "moves" the front exactly one grid space in a time interval and all of the truncation errors magically cancel. In general, however, the only way we have to control numerical dispersion errors in solving transport problems is to use small time and space grid sizes. (There are some special ways to use finite-differences to control numerical dispersion – for example, the "method of characteristics" – but we will not consider these here.)

9.3. The advection-dispersion equation

With this bit of background on numerical dispersion, we return to the full advection-disperion equation.

$$\frac{\partial c}{\partial \hat{t}}+\frac{\partial c}{\partial \hat{x}}=\frac{1}{\mathbf{Pe}}\frac{\partial^2 c}{\partial \hat{x}^2}$$

A Crank-Nicolson approximation for this equation is:

$$\frac{c_i^{j+1}-c_i^j}{\Delta t}+\frac{1}{2}\left[\frac{c_{i+1}^{j+1}-c_{i-1}^{j+1}}{2\Delta x}+\frac{c_{i+1}^{j}-c_{i-1}^{j}}{2\Delta x}\right]$$
$$=\left(\frac{1}{\mathbf{Pe}}\right)\left(\frac{1}{2}\right)\left(\frac{c_{i+1}^{j+1}-2c_i^{j+1}+c_{i-1}^{j+1}}{\Delta x^2}+\frac{c_{i+1}^{j}-2c_i^{j}+c_{i-1}^{j}}{\Delta x^2}\right).$$

(Note that the Lax method is unstable when the dispersion term is included in the equation.) Let's look at a *MATLAB* implementation of this solution for a couple of grid spacings and compare the solution to an analytical solution for a semi-infinite domain. (The comparison will be valid as long as the concentration at the downstream end of the domain that we are simulating remains very close to the zero background concentration.)

```
% adv_disp.m
% Crank-Nicolson solution to advection-dispersion equation
% Solution for a Peclet number of 30, a distance of 4 dimensionless
% units and a total time of 1.5.
Pe=30; length=4; endtime=1.5;
%
% First do the solution for dx=0.2 (21 nodes) and dt=0.25.
nnodes=21; nsoln=nnodes-2; dx=length/(nnodes-1); dt=0.25;
ntimes=endtime/dt;
x1=0:dx:length;
cold=zeros(nnodes,1); cold(1)=1;  %set initial and boundary conditions
Mterm=1/dt+1/(Pe*dx^2);
Uterm=1/(4*dx)-1/(2*Pe*dx^2);
Lterm=-1/(4*dx)-1/(2*Pe*dx^2);
M_diag=sparse(1:nsoln,1:nsoln,Mterm,nsoln,nsoln);
U_diag=sparse(1:nsoln-1,2:nsoln,Uterm,nsoln,nsoln);
L_diag=sparse(2:nsoln,1:nsoln-1,Lterm,nsoln,nsoln);
```

```
A=M_diag+U_diag+L_diag;   %construct the left-hand-side matrix
Mterm=1/dt-1/(Pe*dx^2);
Uterm=-1/(4*dx)+1/(2*Pe*dx^2);
Lterm=1/(4*dx)+1/(2*Pe*dx^2);
M_diag=sparse(1:nsoln,1:nsoln,Mterm,nsoln,nsoln);
U_diag=sparse(1:nsoln-1,2:nsoln,Uterm,nsoln,nsoln);
L_diag=sparse(2:nsoln,1:nsoln-1,Lterm,nsoln,nsoln);
rhsmatrix=M_diag+U_diag+L_diag;   %construct the right-hand-side matrix
for j=1:ntimes
   rhs=rhsmatrix*cold(2:nnodes-1);   %calculate rhs vector
      %adjust rhs vector for boundary condition at x=0
   rhs(1)=rhs(1)+2*(1/(4*dx)+1/(2*Pe*dx^2))*cold(1);
   cnew=A\rhs;   %solve the equation
   cnew=[cold(1);cnew;cold(nnodes)];
   cold=cnew;
end
c1=cnew;
%
% now do the solution for 81 nodes and for dt=0.1
%
nnodes=81; nsoln=nnodes-2; dx=length/(nnodes-1); dt=0.1;
ntimes=endtime/dt;
x2=0:dx:length;
cold=zeros(nnodes,1); cold(1)=1;
Mterm=1/dt+1/(Pe*dx^2);
Uterm=1/(4*dx)-1/(2*Pe*dx^2);
Lterm=-1/(4*dx)-1/(2*Pe*dx^2);
M_diag=sparse(1:nsoln,1:nsoln,Mterm,nsoln,nsoln);
U_diag=sparse(1:nsoln-1,2:nsoln,Uterm,nsoln,nsoln);
L_diag=sparse(2:nsoln,1:nsoln-1,Lterm,nsoln,nsoln);
A=M_diag+U_diag+L_diag;
Mterm=1/dt-1/(Pe*dx^2);
Uterm=-1/(4*dx)+1/(2*Pe*dx^2);
Lterm=1/(4*dx)+1/(2*Pe*dx^2);
M_diag=sparse(1:nsoln,1:nsoln,Mterm,nsoln,nsoln);
U_diag=sparse(1:nsoln-1,2:nsoln,Uterm,nsoln,nsoln);
L_diag=sparse(2:nsoln,1:nsoln-1,Lterm,nsoln,nsoln);
rhsmatrix=M_diag+U_diag+L_diag;
for j=1:ntimes
   rhs=rhsmatrix*cold(2:nnodes-1);
   rhs(1)=rhs(1)+2*(1/(4*dx)+1/(2*Pe*dx^2))*cold(1);
   cnew=A\rhs;
   cnew=[cold(1);cnew;cold(nnodes)];
   cold=cnew;
end
c2=cnew;
%
% The analytical solution.
xanalyt=0:0.02:4;
arg1=(xanalyt-endtime)./(2*sqrt(endtime/Pe));
arg2=(xanalyt+endtime)./(2*sqrt(endtime/Pe));
o=ones(size(xanalyt));
canalyt=0.5*(o-erf(arg1)+exp(1/Pe)*(o-erf(arg2)));
plot(x1,c1,'bo-',x2,c2,'r+-',xanalyt,canalyt,'g')
xlabel('Dimensionless distance')
ylabel('Reduced concentration')
legend('coarse spacing','finer spacing','analytical solution')
```

The results of the computation (Figure 9.2) indicate that the effects of numerical dispersion are easily noticeable for the coarse spacing but appear to be greatly diminished for the finer spacing. One way to check for numerical dispersion is to halve the grid spacing and see if the answer remains the same. Note also the "overshoot" near the front for the coarse spacing. This, like the more subtle effects of numerical dispersion, is a feature that also is seen quite often in numerical solutions for problems in which advection is important.

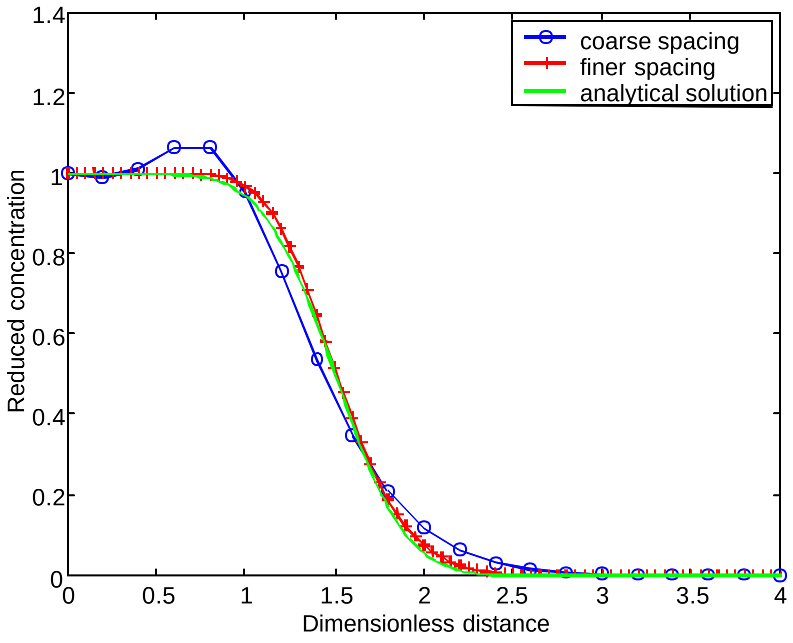


Figure 9.2. Results for numerical solution of the advection-dispersion equation.

9.4. Transport of reactive solutes

Often, we are concerned with solutes that interact with the mineral grains – of the aquifer if we are dealing with groundwater transport, of the streambed material if we are dealing with streamflow. To illustrate some of the embellishments of the methods that we have considered so far when applied to such transport problems, we will look at the transport of silica colloid through sand (Saiers et al., 1994). The interaction between the colloidal particles and the sand grains can be described using first-order kinetics. The transport equations are:

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = D\frac{\partial^2 c}{\partial x^2} - k_f\, c + k_b\, s$$

$$\frac{\partial s}{\partial t} = k_f c - k_b\, s$$

where s is the amount of adsorbed constituent (equal to the bulk density of the aquifer material divided by the porosity times the adsorbed concentration expressed in mass of constituent per mass of mineral grains), and k_f and k_b are "forward" and "backward" rate coefficients, respectively, for the transfer of colloidal mass between the aqueous (c) and the adsorbed (s) phases. A fully implicit finite-difference approximation for these equations is:

$$\frac{c_i^{j+1}-c_i^j}{\Delta t}+u\frac{c_{i+1}^{j+1}-c_{i-1}^{j+1}}{2\Delta x}=D\frac{c_{i+1}^{j+1}-2c_i^{j+1}+c_{i-1}^{j+1}}{\Delta x^2}-k_f c_i^{j+1}+k_b s_i^{j+1}$$
$$\frac{s_i^{j+1}-s_i^j}{\Delta t}=k_f c_i^{j+1}-k_b s_i^{j+1}$$

The way to picture the formulation of (and hence the solution to) this problem in matrix-vector notation is to note that we have twice as many unknowns now – the values of the c_i^{j+1} for the "regular" advection-dispersion problem and the s_i^{j+1} as well. To gain a solution, we again number the *c*'s sequentially, c_1 through c_n. Now, rather than doing the same for the *s*'s, we let the first *s* be c_{n+1}, the second *s* be c_{n+2}, and so on so that the n^{th} *s* is c_{2n}. Our vector of unknowns is then

$$c=[c_1,c_2,.......,c_n,c_{n+1},c_{n+2},........,c_{2n}]'$$

where the second *n* elements are the unknown values of the sorbed concentration.

How does the coefficient matrix look for our approach? Consider the matrix to be blocked into $n\times n$ submatrices. The upper part of the matrix is for the equations for the aqueous concentrations. The diagonal elements in the upper left block (M in matrix below) are the coefficients on c_i^{j+1}: $1/\Delta t+2D/\Delta x^2+k_f$. The upper diagonal elements (U) are the coefficients on c_{i+1}^{j+1}: $u/2\Delta x-D/\Delta x^2$. Likewise the lower diagonal (L) contains the coefficients on the c_{i-1}^{j+1}: $-u/2\Delta x-D/\Delta x^2$. The matrix block in the upper right (r) contains the coefficients on the *s*'s in the equations for the aqueous concentrations. The diagonal elements (for the *block*) are the coefficients on s_i^{j+1}, $-k_b$.

$$\left|\begin{array}{ccccccccccc|ccccccccccc}
M&U&&&&&&&&&&r&&&&&&&&&&\\
L&M&U&&&&&&&&&&r&&&&&&&&&\\
&L&M&U&&&&&&&&&&r&&&&&&&&\\
&&L&M&U&&&&&&&&&&r&&&&&&&\\
&&&L&M&U&&&&&&&&&&r&&&&&&\\
&&&&L&M&U&&&&&&&&&&r&&&&&\\
&&&&&L&M&U&&&&&&&&&&r&&&&\\
&&&&&&L&M&U&&&&&&&&&&r&&&\\
&&&&&&&L&M&U&&&&&&&&&&r&&\\
&&&&&&&&L&M&U&&&&&&&&&&r&\\
&&&&&&&&&L&M&&&&&&&&&&&r\\
\hline
d&&&&&&&&&&&m&&&&&&&&&&\\
&d&&&&&&&&&&&m&&&&&&&&&\\
&&d&&&&&&&&&&&m&&&&&&&&\\
&&&d&&&&&&&&&&&m&&&&&&&\\
&&&&d&&&&&&&&&&&m&&&&&&\\
&&&&&d&&&&&&&&&&&m&&&&&\\
&&&&&&d&&&&&&&&&&&m&&&&\\
&&&&&&&d&&&&&&&&&&&m&&&\\
&&&&&&&&d&&&&&&&&&&&m&&\\
&&&&&&&&&d&&&&&&&&&&&m&\\
&&&&&&&&&&d&&&&&&&&&&&m
\end{array}\right|$$

The lower half of the matrix is for the equations for the adsorbed concentrations. The diagonal elements in the right, lower block (m) are $1/\Delta t+k_b$, and the coefficients in the lower left block (d) are $-k_f$. The right-hand-side vector is composed of the appropriate "known" quantities involving the *j* time step.

The solution in which we are interested is the breakthrough curve at the end of a column of sand. Initial conditions are zero concentration everywhere. The top boundary condition is a reduced concentration of 1 during pulse injection and a reduced concentration of zero thereafter. The bottom boundary condition is subtler. Flow occurs out of the bottom and the concentration changes with time. Thus, the bottom boundary condition is neither a fixed concentration nor a specified flux. The appropriate condition is that the flux occurs only by advection, i.e. dispersion=0. This implies that $\partial^2 c/\partial x^2 = 0$ (since $D \neq 0$) or, in terms of finite differences, $c_{n+1} - 2c_n + c_{n-1} = 0$. Solving for the concentration at the fictitious "*n*+1" node gives $c_{n+1} = 2c_n - c_{n-1}$.

```
% kinsorp.m  Implicit solution to advection-dispersion equation with
% kinetic sorption
% Solution for u=15.38cm/h, D=2.43cm^2/h, L=14.5cm, kf=0.13h^-1,
% kb=1.08h^-1, and a pulse input of 2.28 h (2.42 pore volumes).
v=15.38; length=14.5; D=2.43; kf=0.13; kb=1.08; pulse=2.28; endtime=5.0;
%
% do the solution for 49 nodes and for dt=0.02h.
%
nnodes=49; nsoln=nnodes; dx=length/(nnodes-1); dt=0.02;
ntimes=endtime/dt; npulse=pulse/dt;
ntotal=2*nsoln;
time=0:dt:endtime-dt;
time=time/0.94;  %correction specific to this data set
cbottom=zeros(size(time));
c=zeros(nnodes,1);c(1)=1;
s=zeros(nnodes,1);
cold=[c;s];
%
% Set the coefficient matrix
Mterm=1/dt+2*D/(dx^2)+kf;
Uterm=v/(2*dx)-D/(dx^2);
Lterm=-v/(2*dx)-D/(dx^2);
M_diagUL=sparse(1:nsoln,1:nsoln,Mterm,ntotal,ntotal);
U_diagUL=sparse(1:nsoln-1,2:nsoln,Uterm,ntotal,ntotal);
L_diagUL=sparse(2:nsoln,1:nsoln-1,Lterm,ntotal,ntotal);
M_diagUR=sparse(1:nsoln,nsoln+1:ntotal,-kb,ntotal,ntotal);
M_diagLR=sparse(nsoln+1:ntotal,nsoln+1:ntotal,1/dt+kb,ntotal,ntotal);
M_diagLL=sparse(nsoln+1:ntotal,1:nsoln,-kf,ntotal,ntotal);
A=M_diagUL+U_diagUL+L_diagUL+M_diagUR+M_diagLR+M_diagLL;
% Bottom boundary
A(nnodes,nnodes)=1/dt+v/dx+kf;
A(nnodes,nnodes-1)=-v/dx;
% Set the right hand side
rhs=cold./dt;
% Do the solution
for j=1:ntimes
   % top boundary
      if j<=npulse
        rhs(1)=rhs(1)+v/(2*dx)+D/(dx^2);
```

```
          end
     cnew=A\rhs;
     cold=cnew;
     rhs=cold./dt;
     cbottom(j)=cnew(nsoln);
end
load silica.dat;
plot(time,cbottom,'b',silica(:,1),silica(:,2),'+r')
xlabel('Dimensionless time, (pore volumes)')
ylabel('Reduced concentration')
legend('model','data')
```

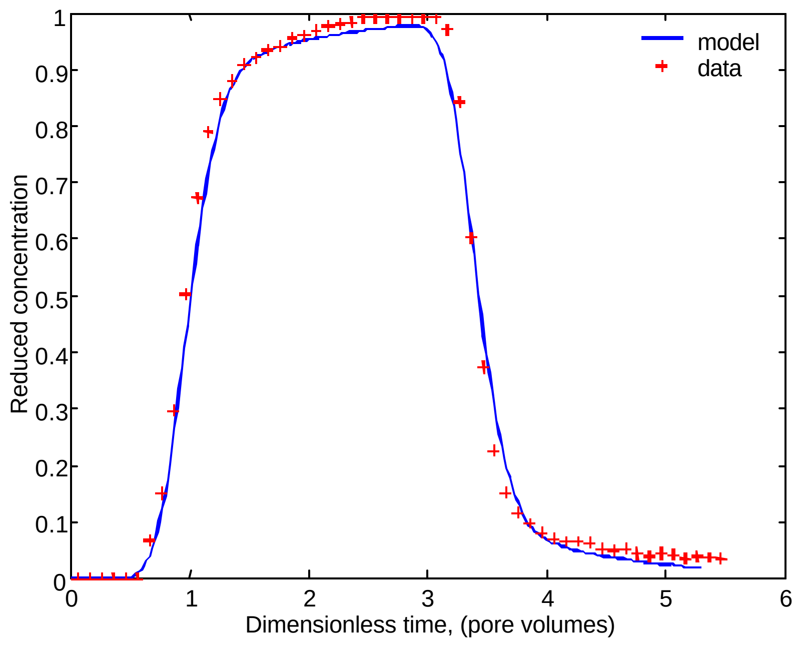


Figure 9.3. Breakthrough curve for example.

9.5. Problems

1. The transport of solute in a stream flowing over alluvium can be approached by considering the water velocity in the alluvium to be negligible with respect to that in the stream and by considering the exchange of solutes between the stream and the groundwater in the sediments to be governed by a simple first-order expression. (If you need to read up on the notion of the "transient-storage" model, you can start with the paper by Bencala and Walters, 1983.) The equations to solve (for a somewhat more simplified representation of the channel and the flow processes than that given by Bencala and Walters) are

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = D\frac{\partial^2 c}{\partial x^2} + \alpha\,(s-c)$$

$$\frac{ds}{dt} = -\alpha\,\frac{A}{A_s}(s-c)$$

where c represents concentration in the stream, s represents concentration in the substream sediments (the "hyporheic zone"), α is a first-order exchange coefficient, A is the stream cross-sectional area , and A_s is the cross-sectional area of the storage zone.

Write a code to solve the transient-storage-zone model. Apply the code to the stream at Shaver Hollow, Virginia using the data below (from Castro and Hornberger, 1991). [To get into the ballpark with the parameter values that you will need, note that Bencala and Walters obtained values of α in the 10^{-5} per second range, D on the order of 0.1 $m^2\ s^{-1}$, and A/A_s in the range of 0.3 – 1 for Uvas Creek, a small gravel-bed stream; other data indicate that A/A_s is larger for larger streams and rivers (Bencala and Walters, 1983).]

Data below are for bromide concentrations at a station 131 m downstream of the injection point. At the injection point, concentrations in the stream were maintained approximately constant at 47.8 mg L^{-1} from 1015 on 2 Aug 1988 through 0700 on 6 Aug 1988. Concentrations at the injection site were zero thereafter.

Day of August 1988	*Time*	*Bromide Concentration* (mg L^{-1})
2	1406	1.67
2	1506	4.96
2	1606	7.60
2	1704	10.38
2	1806	12.61
2	1906	14.37
2	2058	16.82
2	2214	18.92
2	2359	21.29
3	0206	23.04
3	0413	24.92
3	0613	25.92
3	0802	25.92
3	1204	28.04
3	1609	28.04
3	2016	31.55
4	0800	34.13
4	1155	31.55

4	1602	30.34
4	1955	32.81
5	0818	36.92
5	1202	34.13
5	1558	34.13
5	2004	35.50
6	0640	38.41
6	0730	36.92
6	0829	38.41
6	0956	37.31
6	1156	34.51
6	1409	27.30
6	1616	15.81
6	1815	9.89
7	0823	3.39
7	1147	2.66
7	1628	2.18
8	0825	1.64
9	0712	0.99
12	0825	0.52
20	0920	0.14

9.6. References

Bencala, K.E. and R.A. Walters, Simulation of solute transport in a mountain pool and-riffle stream: A transient storage model, *Water Resour. Res., 19*: 718-724, 1983.

Castro, N.M. and G.M. Hornberger, Surface-subsurface water interactions in an alluviated mountain stream channel, *Water Resour. Res., 27*: 1613-1621, 1991.

Fischer, H.B., E.J. List, R.C.Y. Koh, J. Imberger, and N.H. Books, *Mixing in Inland and Coastal Waters*, 483 pp., Academic Press, New York, 1979.

Hemond, H.F. and E.J. Fechner-Levy, *Chemical Fate and Transport in the Environment*, 433 pp., Academic Press, San Diego, 2000.

Hornberger, G.M., Mills, A.L., and J.S. Herman, Bacterial transport in porous media: evaluation of a model using laboratory observations, *Water Resour. Res., 28*: 915-938, 1992.

Saiers, J.E., Hornberger, G.M., and C. Harvey, Colloidal silica transport through structured, heterogeneous porous media, *J. Hydrology 163*: 271-288, 1994.

CHAPTER 10

The Finite Element Method: An Introduction

Numerical Methods in the Hydrological Sciences: AGU Lecture Notes 1
Copyright 2004 by the American Geophysical Union
doi: 10.1029/001LN12
(Version 2004-08-02)

10. The Finite Element Method: An Introduction

10.1. Background

Finite-difference methods for solving differential equations are attractive for a number of reasons. First, they are conceptually straightforward – they "make sense". Second, the equations are relatively easy to derive, at least for a regular mesh. Third, programming to obtain a solution is not too difficult.

Finite-difference methods have some serious drawbacks, however. First, it often is desirable to have an irregular mesh (unequally spaced nodes). The finite difference equations are not easy to write in general terms under this condition. Second, it is quite cumbersome to handle irregular boundaries using finite differences. And finally, anisotropy is not easy to incorporate into finite-difference methods.

A popular numerical method that overcomes the drawbacks of finite-difference methods (admittedly at the expense of some of the attractive features of the method) is the finite-element method. In the finite-element method, the solution to the differential equation is approximated as a continuous function of the independent variables, as opposed to being approximated at only a discrete number of points, the nodal points of a mesh. The presentation below starts with the method of collocation to illustrate the idea of approximation with a continuous function and then goes on to the finite-element method itself.

10.2. Collocation

The basic idea for the method of collocation is to consider some functional approximation to the solution to a differential equation and then find coefficients in the approximate function to make it "close" to the actual solution. For example, we can consider the approximating function to be a polynomial of degree n with the $(n+1)$ coefficients selected to match the boundary conditions and to fit the solution as well as possible. An illustrative example should help clarify the idea.

Consider the problem of groundwater flow between two drains (Figure 10.1). With the Dupuit assumptions[1], the equation describing the problem is (e.g., see Fetter 2001):

$$\begin{aligned} &\frac{d}{dx}\left[Kh\frac{dh}{dx}\right] = -w \\ &h(0) = h_1 \\ &h(L) = h_2 \end{aligned} \qquad (10.1)$$

where K is hydraulic conductivity and w is recharge rate. To make the problem even more concrete, let h_1=5 m, h_2=10 m, L=100 m, K=10 cm d^{-1}, and w=0.15 cm d^{-1}.

[1] The Dupuit assumptions are used to simplify the problem of flow in an unconfined aquifer. Basically, the assumptions are that the vertical component of flow in a horizontal aquifer is negligible – that the head gradient is the slope of the water table and that flow is horizontal.

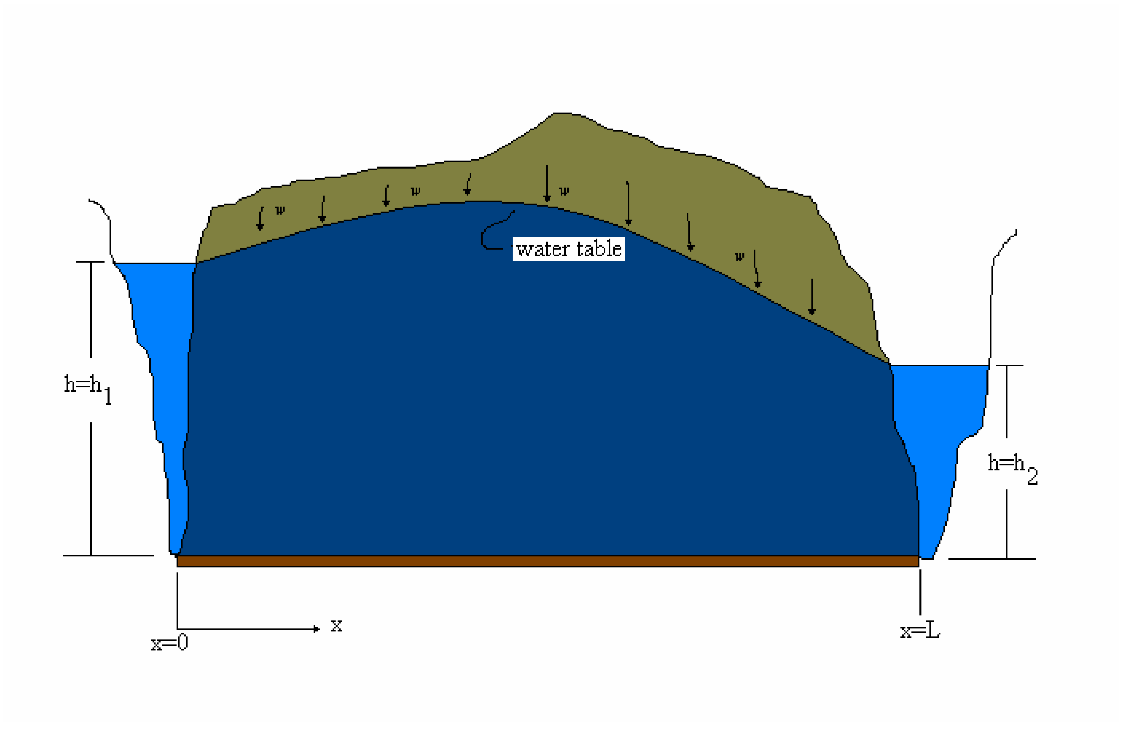


Figure 10.1. Schematic of groundwater flow between two drains.

Choose a second-degree polynomial to represent the true solution. The polynomial can be made to satisfy the boundary conditions by judicious choice of two of the coefficients.

$$\hat{h} = 5 + 0.05x + Cx(x - 100) \tag{10.2}$$

where C is a constant at our disposal to fit the approximate solution to the true solution. First, express the left side of (10.1) in terms of the trial function (10.2).

$$\hat{h} = 5 + 0.05x + Cx(x - 100) = 5 + (0.05 - 100C)x + Cx^2$$

$$\frac{d\hat{h}}{dx} = (0.05 - 100C) + 2Cx$$

$$K\hat{h}\frac{d\hat{h}}{dx} = K\left[5 + (0.05 - 100C)x + Cx^2\right]\left[(0.05 - 100C) + 2Cx\right]$$

$$= K\left\{5(0.05 - 100C) + \left[(0.05 - 100C)^2 + 10C\right]x + 3C(0.05 - 100C)x^2 + 2C^2x^3\right\}$$

$$\frac{d}{dx}\left[K\hat{h}\frac{d\hat{h}}{dx}\right] = K\left\{\left[(0.05 - 100C)^2 + 10C\right] + 6C(0.05 - 100C)x + 6C^2x^2\right\}$$

For the exact solution to the differential equation, the derivative that we just calculated is equal to -w for all values of x. Because $\hat{h}$ is approximate, there will be a *residual* between the derivative and w for at least some values of x:

$$residual = r = -w - K\{[(0.05-100C)^2 + 10C] + 6C(0.05-100C)x + 6C^2x^2\}$$

The objective of the method of collocation is to make the residual as close to zero as possible. With one free coefficient at our disposal, we can select one value of x at which to make the residual equal zero exactly. A logical choice might be at $x=50$, halfway between the drains. This choice leads to the following solution.

$$-w - K\{[(0.05-100C)^2 + 10C] + 6C(0.05-100C)50 + 6C^2(50)^2\} = 0$$
$$-w/K - \{[(0.05-100C)^2 + 10C] + 6C(0.05-100C)50 + 6C^2(50)^2\} = 0$$
$$-0.0175 - 15C + 5000C^2 = 0$$
$$C = \text{-}8.9792\text{x}10^{-4}.$$

[Note that there are two solutions for C (the equation is quadratic) but only the one that makes sense for the problem at hand is presented. Also note that reducing the residual at x=50 to zero is *not* the same as reducing the difference between approximate and actual values of h at x=50 to zero.]

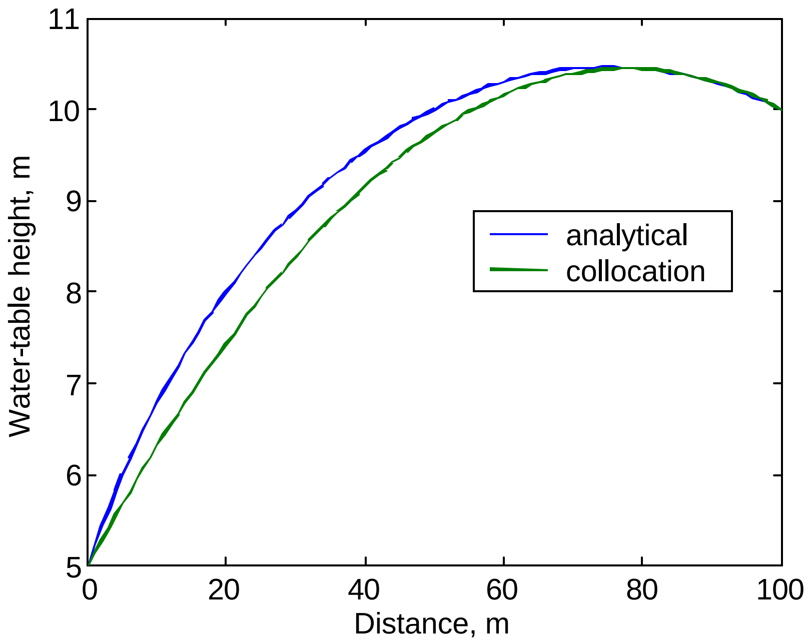


Figure 10.2. Approximate solution to the groundwater-drain problem determined using collocation compared with the analytical solution.

The approximate solution is "similar" to the analytical solution[2] (Figure 10.2), but quite possibly not as good as we might want. In the method of collocation, a better fit can be obtained by using a higher order polynomial as the trial function. For example, if we used a cubic equation, we would

[2] The *MATLAB* symbolic toolbox can be used to derive the analytical solution. Letting u be dh/dx, the problem can be represented as two differential equations, h*du/dx+u^2=-w/K and dh/dx=u. The solution is obtained by the statement: `[h,u]=dsolve('h*Du+u^2=-.015','Dh=u','h(0)=5','h(100)=10','x').`

have had two free coefficients to fit and could have chosen two points in the domain to match the differential equation exactly.

The approximate solution with the second-degree polynomial would have been much better for the example problem had the drains been only 10 m apart and the head difference between them only 0.5 m. Given the stated problem, however, increased accuracy requires a more complex approximating polynomial and, consequently, a solution to more complicated equations. The finite-element method overcomes the problems of collocation by breaking the domain into a number of elements and using simple polynomials to approximate the solution over each "small" element.

10.3. Weighted residual method

In the method of collocation applied to the example problem, the residual between the approximate solution and the right-hand side of the differential equation was minimized (set to zero) at one point in the domain. A general method for determining a good approximation is to reduce the integral of the residuals over the domain to zero. In fact, in the general method of *weighted residuals,* the residuals are multiplied by some function $W(x)$ (the weighting function) and the integral of the product is set to zero. For the example problem, we would require:

$$\int_0^{100} W(x)r(x)dx = 0 \tag{10.3}$$

The method of collocation as we applied it is a special case of the weighted residual method with the weighting function set to the Dirac delta function, a function that "selects" the collocation points ($x=50$ in the example) from the integral and sets the residual at these points to zero.

10.4. The finite element approach: Galerkin weighted residual method

Consider again the example problem of groundwater flow between drains. This time, we will simplify the problem by considering the linearized version – i.e., by replacing the term Kh with a constant transmissivity, T, which we will take to be 80 m^{2}d^{-1}.

$$\frac{d}{dx}\left[T\frac{dh}{dx}\right] = T\frac{d^2h}{dx^2} = -w$$

or,

$$\begin{aligned} &T\frac{d^2h}{dx^2} = -w = -0.15cm/day \\ &h(0) = 5m \\ &h(100) = 10m \end{aligned} \tag{10.4}$$

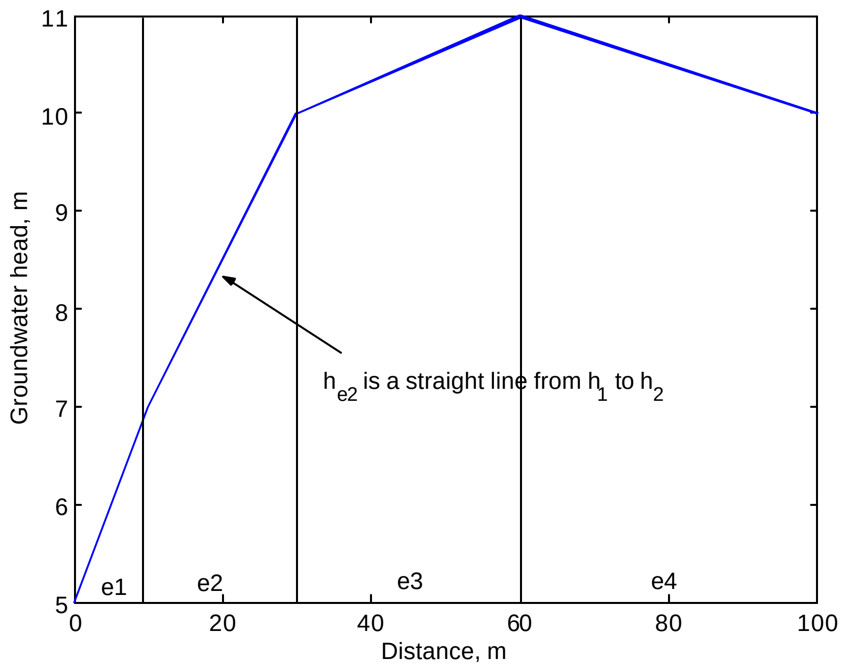


Figure 10.3. The domain is broken into four elements, e1 through e4. The groundwater head is approximated by a straight line over each element, with the endpoints denoted by the heads at the nodes, h_i.

The first step in the finite-element method is to divide the domain into *elements*, in this case four line segments (Figure 10.3). The elements are bounded by *nodes*, the endpoints of the line segments. The next step is to approximate the solution over each element. The simplest approximating polynomials are straight lines, which can be expressed as:

$$
\begin{aligned}
h_{e1} &= h_1\left(\frac{x_2 - x}{x_2 - x_1}\right) + h_2\left(\frac{x - x_1}{x_2 - x_1}\right) \\
h_{e2} &= h_2\left(\frac{x_3 - x}{x_3 - x_2}\right) + h_4\left(\frac{x - x_2}{x_3 - x_2}\right) \\
h_{e3} &= h_3\left(\frac{x_4 - x}{x_4 - x_3}\right) + h_4\left(\frac{x - x_3}{x_4 - x_3}\right) \\
h_{e4} &= h_4\left(\frac{x_5 - x}{x_5 - x_4}\right) + h_5\left(\frac{x - x_4}{x_5 - x_4}\right)
\end{aligned}
\tag{10.5}
$$

It is easy to check that equations (10.5) are straight lines connecting the heads at the two endpoints of an element.

The straight-line approximation for an element can be written in terms of general "basis functions", $x(x)$, defined for each element. The ith basis function has a value of one at node i and drops linearly to zero at the adjacent nodes (Figure 10.4). The basis functions are written explicitly as functions of x as follows.

$$x_i = \begin{cases} \dfrac{x_{i+1} - x}{x_{i+1} - x_i}, & \text{for } x_i \le x \le x_{i+1} \\ \dfrac{x - x_{i-1}}{x_i - x_{i-1}}, & \text{for } x_{i-1} \le x \le x_i \end{cases} \tag{10.6}$$

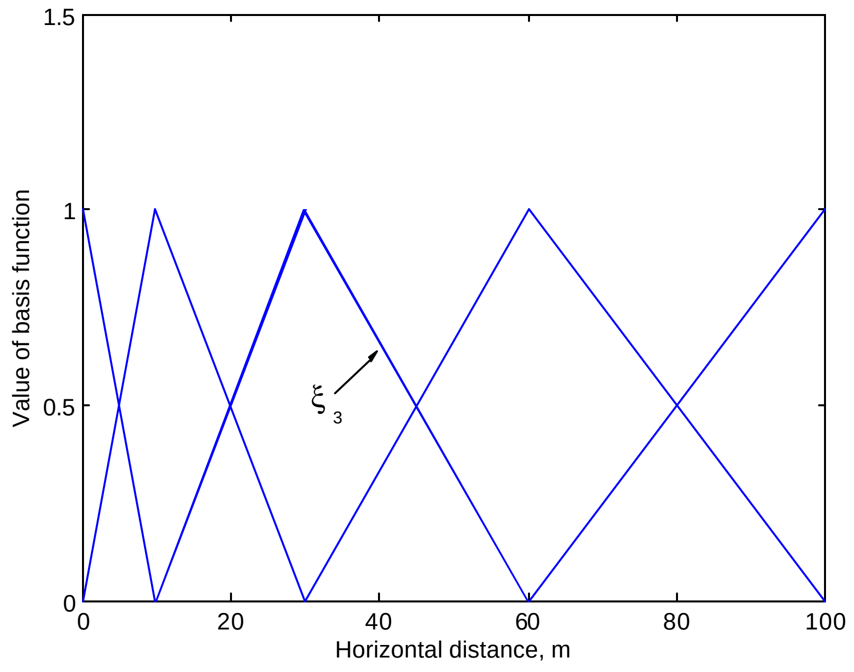


Figure 10.4. The basis functions ("chapeau functions") for the example problem. There is a basis function for each node. The function for node three is shown by the bold line.

The approximations for the groundwater heads in terms of the basis functions are:

$$\hat{h}_{ei} = h_i ?_i + h_{i+1} ?_{i+1} \tag{10.7}$$

where i represents the ith element. (Compare equation 10.7 with equation 10.5 given the definition of the x_i in equation 10.6.) The expression for h over the entire domain is then:

$$\hat{h} = \sum_k x_k h_k \tag{10.8}$$

The next step is to substitute the approximate function for h into the governing equation and form the residuals as we did in section 10.2.

$$residual = r = T\frac{d^2\hat{h}_e}{dx^2} + w$$

This residual is weighted and integrated over the domain of h as suggested in equation (10.3). The Galerkin method uses the basis functions themselves as the weighting functions. Although this choice may seem arbitrary, it turns out that it can be shown that use of the basis functions as weights is "best" in a sense. The specific form for equation 10.3 for the example problem is then:

$$\int_R x\left(T\frac{d^2\hat{h}_e}{dx^2} + w\right)dx = 0 \qquad (10.9)$$

where R is the domain over which the integration is carried out. The second derivative is eliminated from equation (10.9) by integrating by parts[3]. Let u be x and dv be $T\frac{d^2\hat{h}_e}{dx^2}dx$ (giving $v = T\frac{dh_e}{dx}$). Equation (10.9) can be written:

$$xT\frac{d\hat{h}_e}{dx}\Bigg|_0^{100} - \int_0^{100}\frac{dx}{dx}T\frac{d\hat{h}_e}{dx}dx + \int_0^{100} xwdx = 0 \qquad (10.10)$$

We started our solution to the problem by dividing the domain into a number of elements. Instead of integrating across the entire domain R – from $x=0$ to $x=100$ – it makes more sense to integrate over the elements and then sum them to get the total integral. That is, we can rewrite equation (10.10) as:

$$\sum_e\left\{x_eT_e\frac{d\hat{h}_e}{dx}\Bigg|_{x_i}^{x_{i+1}} - T_e\int_{x_i}^{x_{i+1}}\frac{dx_e}{dx}\frac{d\hat{h}_e}{dx}dx\right\} + \int_{x_i}^{x_{i+1}} xwdx = 0 \qquad (10.11)$$

where the e subscripts refer to elements. Note that in this particular example T_e is constant, but even if it were variable, it could be approximated as constant over an element and still be brought outside the integral for the element. In such a case, the coefficients derived would vary from element to element as T_e varied.

Next we evaluate equation (10.11) for a single element. First, note that the first term on the left-hand side of (10.11) is simply the total water flow into the element at x_i ($Q_i = T\frac{dh}{dx}\Big|_{x=x_i}$) minus the flow out of the element at x_{i+1} (Q_{i+1}). Now consider the second integral in equation (10.11). For the element, there are two portions of the basis functions to consider (Figure 10.5). For x_i, the second integral on the left-hand side of (10.11) reads:

$$T_e\int_{x_i}^{x_{i+1}}\frac{dx_i}{dx}\frac{d}{dx}(h_ix_i + h_{i+1}x_{i+1})dx = T_e\int_{x_i}^{x_{i+1}}(\frac{dx_i}{dx}\frac{dx_i}{dx}h_i + \frac{dx_i}{dx}\frac{dx_{i+1}}{dx}h_{i+1})dx \qquad (10.12)$$

and for x_{i+1} it is:

[3] $\int udv = uv - \int vdu.$

$$T_e \int_{x_i}^{x_{i+1}} \frac{dx_{i+1}}{dx}\frac{d}{dx}(h_i x_i + h_{i+1}x_{i+1})dx = T_e \int_{x_i}^{x_{i+1}} \left(\frac{dx_{i+1}}{dx}\frac{dx_i}{dx}h_i + \frac{dx_{i+1}}{dx}\frac{dx_{i+1}}{dx}h_{i+1}\right)dx \quad (10.13)$$

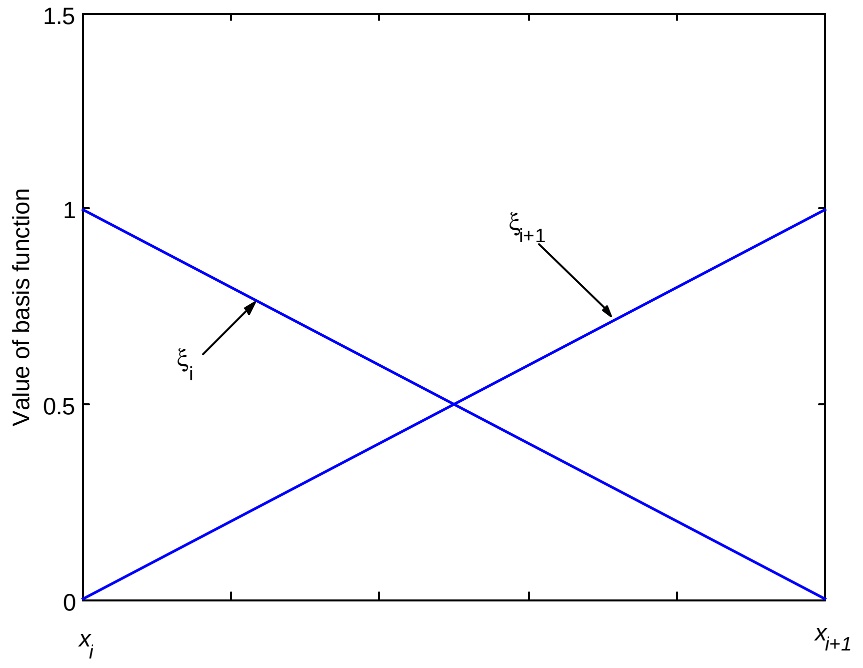


Figure 10.5. The two segments of the basis function for a general element (cf. Figure 10.4).

The integral can be evaluated easily once we find the derivatives of the basis functions with respect to x. From equation (10.6) it is clear that:

$$\frac{dx_i}{dx} = \frac{1}{(x_{i+1}-x_i)}, \quad \frac{dx_{i+1}}{dx} = \frac{-1}{(x_{i+1}-x_i)}$$

Thus, the integrands in equations (10.12) and (10.13) are not functions of x and the necessary evaluation is simply the integral of dx.

The last integral on the left-hand side of equation (10.11) is easily evaluated. Consider the integral for x_i and note that we can use equation (10.6) to make a substitution to convert the integral with respect to x to an integral with respect to x_i. In making the substitution, the limits of integration change to 0 to 1 (when x=x_i, x_i =0) and dx is replaced by $(x_{i+1}-x_i)dx_i$.

$$\begin{aligned} w\int_{x_i}^{x_{i+1}} \frac{x-x_i}{x_{i+1}-x_i}dx &= w\int_0^1 x_i\,(x_{i+1}-x_i)\,dx_i \\ &= w(x_{i+1}-x_i)\frac{x_i^2}{2}\Bigg|_0^1 \\ &= \frac{w(x_{i+1}-x_i)}{2} \end{aligned}$$

The integration for x_{i+1} is similar.

Carrying out all of the integrations, the equations corresponding to (10.11) [treating (10.12) and (10.13) separately] become:

$$
\begin{aligned}
&Q_i - Q_{i+1} - T_e \left\{ \frac{h_i}{(x_{i+1}-x_i)^2} - \frac{h_{i+1}}{(x_{i+1}-x_i)^2} \right\}(x_{i+1}-x_i) + w(x_{i+1}-x_i)/2 = 0 \\
&Q_i - Q_{i+1} - T_e \left\{ \frac{-h_i}{(x_{i+1}-x_i)^2} + \frac{h_{i+1}}{(x_{i+1}-x_i)^2} \right\}(x_{i+1}-x_i) + w(x_{i+1}-x_i)/2 = 0
\end{aligned}
\tag{10.14}
$$

Now note that when the sum is done for all elements, the outflow from one element is equal to the inflow to the next. Thus, the "*Q*'s" for all internal nodes cancel and these terms can be dropped. For the example problem, we do not have flow boundary conditions, so the *Q*'s can be dropped there as well. The equations for a single element can be written in matrix-vector form.

$$
\begin{pmatrix} 1/\Delta_e & -1/\Delta_e \\ -1/\Delta_e & 1/\Delta_e \end{pmatrix} \begin{pmatrix} h_i \\ h_{i+1} \end{pmatrix} = \begin{pmatrix} w\Delta_e/2T_e \\ w\Delta_e/2T_e \end{pmatrix} \tag{10.15}
$$

where Δ_e is the element size, in this case $(x_{i+1} - x_i)$.

The finite-element procedure is completed by assembling all of the element matrices – the leftmost matrix in equation (10.15) – into a "global" matrix and solving the resulting equations for the nodal values of head. The global matrix for our 4-element example will be 3 x 3 because the heads at each end of the domain are fixed. (For the first element, the first row and first column of the element matrix disappear because the head is known. The known value of head at node 1 is transposed to the right-hand side of the equation for the first element. Likewise for the last element, the second row and column disappear.) The contribution from the first element to the global matrix is:

$$
\begin{pmatrix} 1/(x_2-x_1) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
$$

The contribution from the second element is:

$$
\begin{pmatrix} 1/(x_3-x_2) & -1/(x_3-x_2) & 0 \\ -1/(x_3-x_2) & 1/(x_3-x_2) & 0 \\ 0 & 0 & 0 \end{pmatrix}
$$

and from the third and fourth:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1/(x_4-x_3) & -1/(x_4-x_3) \\ 0 & -1/(x_4-x_3) & 1/(x_4-x_3) \end{pmatrix}$$

and

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/(x_5-x_4) \end{pmatrix}$$

The global matrix is simply the sum of the element matrices. In the same fashion, the right-hand vector is assembled by adding contributions from each element. Also, the known values of head at the endpoints of the domain are transfered to the right-hand vector. The resulting equations follow.

$$\begin{pmatrix} 1/(x_2-x_1) + 1/(x_3-x_2) & -1/(x_3-x_2) & .0 \\ -1/(x_3-x_2) & 1/(x_3-x_2) + 1/(x_4-x_3) & -1/(x_4-x_3) \\ 0. & -1/(x_4-x_3) & 1/(x_4-x_3) + 1/(x_5-x_4) \end{pmatrix} \begin{pmatrix} h_2 \\ h_3 \\ h_4 \end{pmatrix} =$$

$$\begin{pmatrix} w(x_2-x_1)/2T_1 + w(x_3-x_2)/2T_2 + 5/(x_2-x_1) \\ w(x_3-x_2)/2T_2 + w(x_4-x_3)/2T_3 \\ w(x_4-x_3)/2T_3 + w(x_5-x_4)/2T_4 + 10/(x_5-x_4) \end{pmatrix}$$

(The boundary conditions have been placed in the first and last equations as appropriate. Alternatively, we could have kept 5 equations and simply made the first and fifth simple statements of the boundary conditions – e.g., h_1=5.)

To obtain the numerical solution, assume that transmissivity is constant at 0.8 m^{2}day^{-1}. The finite-element equations are then:

$$\begin{pmatrix} 1/10+1/20 & -1/20 & 0 \\ -1/20 & 1/20+1/30 & -1/30 \\ 0 & -1/30 & 1/30+1/40 \end{pmatrix}\begin{pmatrix} h_2 \\ h_3 \\ h_4 \end{pmatrix}=\begin{pmatrix} 0.0281+0.5 \\ .0469 \\ .0656+0.25 \end{pmatrix}$$

The numerical solution is an excellent approximation to the exact solution[4] (Figure 10.6), even with the fairly coarse mesh.

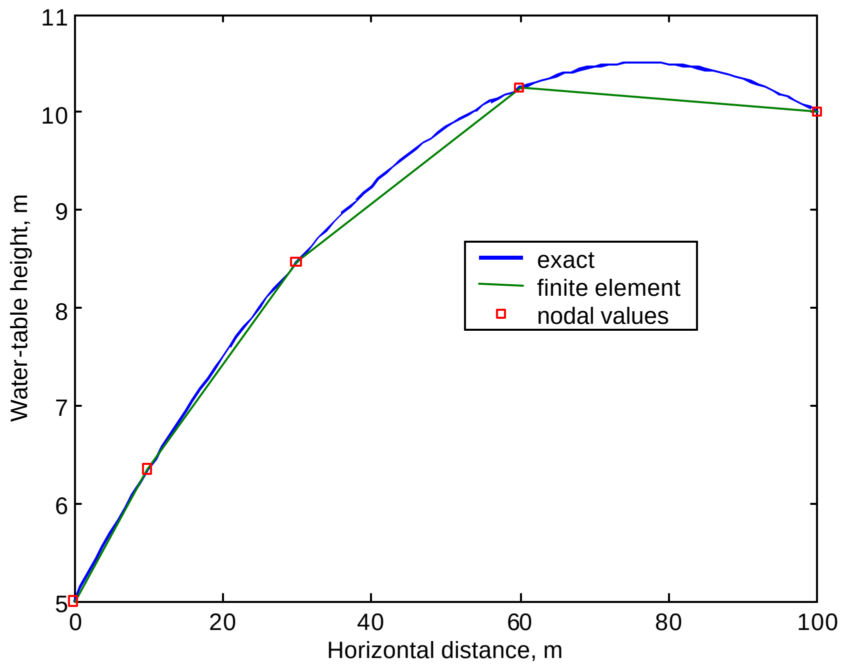


Figure 10.6. Finite element solution to the example problem.

10.5. Steady diffusion into sediment

As an example that demonstrates how variable properties, as well as unequal element size, can be accommodated in the finite-element method, consider the diffusion of oxygen into the sediment at the bottom of a lake or estuary. The equation used is a combination of Fick's law of diffusion and conservation of mass – much the same as the groundwater equation is a combination of Darcy's law and conservation of mass (see Boudreau, 1997). For steady-state conditions, the equation to solve is:

$$\frac{d}{dz}\left(fD\frac{dc}{dz}\right)+R=0. \tag{10.16}$$

[4] The *MATLAB* statement to obtain the exact solution is:

```
h=dsolve('D2h=-0.0015/0.8','h(0)=5','h(100)=10','x').
```

where f is the sediment porosity, D is the diffusion coefficient of oxygen in the pore water (corrected for tortuosity), and R is the rate of oxygen consumption by reactions in the sediment. Extension of the method to conditions other than those assumed here is straightforward [Box 10.1].

Consider a 0.005-m thick layer of sediment with z=0 at depth and z=0.005 m at the sediment surface. The flux of oxygen into the sediment from the overlying water is 0.0125 n mol cm^{-2} s^{-1}. The oxygen concentration at the 0.005m (z=0) depth is zero. The conditions are given mathematically by:

$$c(0) = 0$$

$$J = fD\left.\frac{dc}{dz}\right|_{z=0.005} = 0.0125 \text{ n mol cm}^{-2}\text{s}^{-1}$$

The diffusion coefficient, D is the product of the free-water diffusion coefficient, D_{fw} (=11.6 ×10^{-6} cm^2 s^{-1} for this problem) and the square of the sediment porosity. That is, $D = f^2 D_{fw}$. The porosity, f, varies with depth:

Depth interval (cm) below surface	z at bottom of interval (cm)	Porosity	Rate of oxygen consumption n mol cm^{-3} s^{-1}
0-0.05	0.45	0.9	0.02
0.05-0.1	0.4	0.85	0.07
0.1-0.2	0.3	0.8	0.04
0.2-0.3	0.2	0.75	0.02
0.3-0.5	0	0.7	0.01

We proceed as above to define the basis functions, do the integrations, and so forth, assuming that R and f vary from element to element but are constant within any given element. The finite-element equations are very similar to those for the groundwater problem (the equations are nearly identical.). There are some differences. First, the top boundary (the nth nodal value) condition is a *flux* condition. Look at equation (10.14). The "natural" boundary condition for the finite-element method *is* a flux condition. Thus, all that needs to be done is to place the flux on the right-hand side of the equation. Also, the porosity and diffusion coefficient vary and so are kept as part of the coefficient matrix. To simplify the notation somewhat, define l as $fD = f\left(f^2 D_{fw}\right) = f^3 D_{fw}$. The finite-element equations (i.e., the global matrix-vector equation) are:

$$\begin{bmatrix} 1 & & & & & \\ -l_1/\Delta_1 & l_1/\Delta_1 + l_2/\Delta_2 & -l_2/\Delta_2 & & & \\ & & \bullet & & & \\ & & & \bullet & & \\ & & & & \bullet & \\ & & & -l_{n-2}/\Delta_{n-2} & l_{n-2}/\Delta_{n-2} + l_{n-1}/\Delta_{n-1} & -l_{n-1}/\Delta_{n-1} \\ & & & & -l_{n-1}/\Delta_{n-1} & l_{n-1}/\Delta_{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \bullet \\ \bullet \\ \bullet \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} cbot \\ R_1\Delta_1/2 + R_2\Delta_2/2 \\ \\ \\ \\ R_{n-2}\Delta_{n-2}/2 + R_{n-1}\Delta_{n-1}/2 \\ R_{n-1}\Delta_{n-1}/2 - 0.0125 \end{bmatrix}$$

where the subscripted Δ's indicate the element lengths.

The solution in *MATLAB* is now easy to construct. The m-file below does the calculation and plots the results.

```
% diagen.m
% This is a finite element solution to calculate the steady state
% concentration distribution in a heterogeneous (layered) sediment.
% Set free-water diffusion coefficient
D_fw=11.6e-6;     %cm^2 s^-1
% set nodal values of z
z=[0 0.05 0.1 0.15 0.2 0.25 0.3 0.325 0.35 0.375 0.4 0.4125 0.425 0.4375
0.45 0.4625 0.475 0.4875 0.5];
dz=diff(z);
% n=number of nodes, ne=number of elements
n=length(z);
ne=length(dz);
% phi=array of element porosities; R is array of consumption rates.
i1=find(z<0.2);i2=find(z>=0.2&z<0.3);i3=find(z>=0.3&z<0.4);
i4=find(z>=0.4&z<0.45);i5=find(z>=0.45&z<0.5);
phi=zeros(size(z));
phi(i1)=0.7;phi(i2)=0.75;phi(i3)=0.8;phi(i4)=0.85;phi(i5)=0.9;
lam_e=phi.^3*D_fw;
R=zeros(size(z));
R(i1)=-0.01;R(i2)=-0.02;R(i3)=-0.04;R(i4)=-0.07;R(i5)=-0.02;
% flux=flux at top in n mol cm^-2 s^-1, cbot=constant bottom boundary
% concentration in n mol cm^-3.
flux=0.0125;
cbot=0;
% Set the coefficient matrix and the right-hand vector.
% preallocate matrix
G=zeros(n,n);
rhs=zeros(n,1);
for e=1:ne
    node1=e;node2=e+1;           %nodes for the element
    delta=z(node2)-z(node1);     %size of element
    elmat=zeros(2,2);            %initialize element matrix
    elmat(1,1)=lam_e(e)/delta;
    elmat(1,2)=-lam_e(e)/delta;
    elmat(2,1)=-lam_e(e)/delta;
    elmat(2,2)=lam_e(e)/delta;
    % assemble the global matrix by adding the element matrix
    G(node1,node1)=G(node1,node1)+elmat(1,1);
    G(node2,node2)=G(node2,node2)+elmat(2,2);
    G(node1,node2)=G(node1,node2)+elmat(1,2);
    G(node2,node1)=G(node2,node1)+elmat(2,1);
    % assemble rhs
    rhs(node1)=rhs(node1)+R(node1)*delta/2;
    rhs(node2)=rhs(node2)+R(node1)*delta/2;
end
% Change the first equation for constant concentration
% and the last to include the flux
G(1,1:n)=0; G(1,1)=1;
rhs(1,1)=cbot;
rhs(n)=rhs(n)+flux;
% The finite element equations are in the form G*c=rhs, where "G" is
% the global coefficient matrix, "c" is the vector of unknowns,
% and "rhs" is the vector of known quantities. The MATLAB "\" function
% solves the system of equations.
c=G\rhs;
% Plot the concentrations.
plot(c,z,'r+',c,z,'-')
xlabel('Oxygen concentration (n mol cm^-^3)');
ylabel('z (cm)');
```

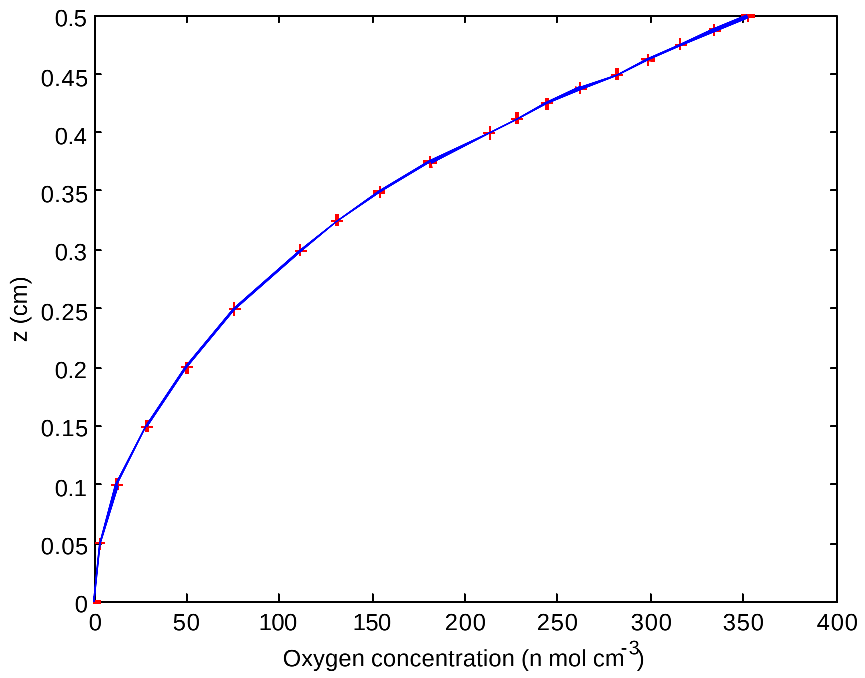


Figure 10.7. Oxygen profile calculated using the finite-element code. Note that the steady-state profile is curved because porosity (and thus diffusion coefficient) and rate of consumption vary with depth.

10.6. Problems

1. Write the finite difference equations for equation (10.1) and compare with the matrix derived using finite elements. What are the similarities and differences?

2. Berg et al. (1998) report measurements of "microprofiles" of NO_3 in freshwater sediment. In the case of nitrate, there can be "consumption" (reducing processes) or "production" (oxidizing processes). The porosity of the sediment is 0.93 and the free-water diffusion coefficient for NO_3 is 1.58×10^{-5} cm^2 s^{-1}. Given estimates of NO_3 consumption in layers of the sediment (Table 1), use a finite-element code to compute and plot the nitrate profile in the sediment. Plot your results against the measurements (Table 2) of the concentration profile. Use the surface flux boundary condition to "calibrate" the model, i.e., adjust the flux to get a good match between the computed and measured values. You might try a flux value of 0.0005 n mol cm^{-2} s^{-1} as a starting point. (Note that **negative** consumption rates represent production.)

Depth interval (cm)	NO_3 consumption rate (n mol cm^{-3} s^{-1})
0-0.16	-0.01
0.16-0.23	-0.08
0.23-0.28	0.11
0.28-0.32	0.01
0.32-0.40	0.00

Measured concentrations.

Depth	NO_3 (n mol cm^{-3})
0	69
0.05	63
0.1	57
0.15	47
0.2	30
0.25	13
0.3	2
0.35	0
0.4	0

3. The equation for early diagenesis in sediments can be modified to take account of *irrigation*, the pumping activity of tube-dwelling animals. In this case equation (10.16) is expanded to:

$$\frac{d}{dz}\left(fD\frac{dc}{dz}\right)+j\,a\left(c_0-c\right)+R=0,$$

where a is an irrigation coefficient and c_0 is the concentration at the sediment-water interface. Write a finite-element code to solve this problem and apply it to the conditions of problem 1. Examine results for $a = 1\times10^{-5}$ s^{-1} and 5×10^{-5} s^{-1}.

4. Write a finite-element code to solve the problem of heat flow in the earth, including advection of thermal energy:

$$\frac{d}{dz}\left(l\,\frac{dT}{dz}\right) - r_f c_f q \frac{dT}{dz} = 0.$$

where q is the specific discharge (units of, e.g., m s^{-1}), and r_f, and c_f, are the fluid density and heat capacity, respectively. [A code for the case of heat flow without advection, `geotherm.m`, is available as a template (see the m-file index).] Use a constant $l = 2.5$ W m^{-1} °C and a constant r_f = 1000 kg m^{-3} and c_f = 4200 J kg^{-1} °C. Solve the problem subject to the boundary conditions: T_{top}=20 °C and T_{bottom}=145 °C. Plot results for the following values of q:

q = 0.0 cm yr^{-1} (static case)
q = +0.1 cm yr^{-1} (up)
$q = +0.5$ cm yr^{-1} (up)
$q = +l.0$ cm yr^{-1} (up)
q = -0.1 cm yr^{-1} (down)
q = -0.5 cm yr^{-1} (down)
$q = -l.0$ cm yr^{-1} (down)

Comment on the implications of the results for interpretations made in Problem 1 in Chapter 8.

10.7. References

Berg, P., Risgaard-Petersen, N., and S. Rysgaard, Interpretation of measured concentration profiles in sediment pore water. *Limnol. Oceanogr. 43*: 1500-1510, 1998.

Boudreau, BP. *Diagenetic Models and their Implementation*, 414 pp., Springer-Verlag, Berlin. 1997.

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

Domenico, P.A. and F.W. Schwartz, *Physical and Chemical Hydrogeology*, 2nd edition, 506 pp., Wiley, New York, 1998.

Box 10.1. The finite element approach for transient conditions with advection

As an example of how the finite element method is extended to problems beyond the solutions for the steady state heat equation, consider the transport of a solute through a porous medium under transient conditions including advection (movement of the solute with the average velocity of the water) and dispersion (the "mixing" of solute due to differing velocities along different flow paths). Recall that we solved a form of the advection-dispersion equation using finite differences in Chapter 9. Here, we also let the solute decay according to a first-order rate law. More details about processes and the equation representing them can be found in hydrogeology texts (e.g., Fetter, 2001; Domenico and Schwartz, 1998). The equation is

$$\frac{\partial c}{\partial t} - D\frac{\partial^2 c}{\partial x^2} + u\frac{\partial c}{\partial x} + l\,c = 0$$

where c is solute concentration, t is time, x is distance, D is the dispersion coefficient, u is average velocity, and l is the decay coefficient.

The weighted residual integral for a typical element is

$$\int_{x_i}^{x_{i+1}} x\left(\frac{\partial c}{\partial t} - D\frac{\partial^2 c}{\partial x^2} + u\frac{\partial c}{\partial x} + l\,c\right)dx$$

The dispersion term is similar to the one for the steady state heat equation discussed in Chapter 10. The element matrix for the dispersion term is (cf. equation (10.15)

$$\begin{pmatrix} D/\Delta_e & -D/\Delta_e \\ -D/\Delta_e & D/\Delta_e \end{pmatrix}$$

The integrations for the other terms are straightforward. First take the advection term for x_i. Note that the element concentration can be written

$$c = c_i x_i + c_{i+1}x_{i+1} = c_i x_i + c_{i+1}(1 - x_i)$$

We can then evaluate the integral (for x_i) as follows, using once again a substitution of x_i for x in the integral and noting that $\frac{\partial c}{\partial x}dx = \frac{\partial c}{\partial x_i}dx_i$.

$$\begin{aligned}\int_{x_i}^{x_{i+1}} x_i u\frac{\partial c}{\partial x}dx &= u\int_0^1 x_i\frac{\partial c}{\partial x_i}dx_i \\ &= u\int_0^1 x_i\left(c_i - c_{i+1}\right)dx_i \\ &= u(c_i - c_{i+1})\left.\frac{x_i^{\,2}}{2}\right|_0^1 \\ &= \frac{u}{2}c_i - \frac{u}{2}c_{i+1}\end{aligned}$$

The integral for x_{i+1} is evaluated in a similar way. The element matrix for the advection term is thus

$$\begin{pmatrix} \frac{u}{2} & -\frac{u}{2} \\ \frac{u}{2} & -\frac{u}{2} \end{pmatrix}$$

The integral for the term that accounts for the decay of the solute (for x_i) is evaluated as

$$\int_{x_i}^{x_{i+1}} x_i\, l\, c_e dx = \int_o^1 l x_i \left[c_i x_i + c_{i+1}(1-x_i)\right]\Delta_e dx_i$$

$$= l\Delta_e \left(\frac{x_i^3}{3}c_i + \frac{x_i^2}{2}c_{i+1} - \frac{x_i^3}{3}c_{i+1}\right]_0^1$$

$$= l\Delta_e \left(\frac{c_i}{3} + \frac{c_{i+1}}{6}\right)$$

The evaluation for x_{I+1} is similar. Thus the element matrix for the decay term is:

$$\begin{pmatrix} \frac{l\Delta_e}{3} & \frac{l\Delta_e}{6} \\ \frac{l\Delta_e}{6} & \frac{l\Delta_e}{3} \end{pmatrix}$$

The time derivative also must be integrated. The order of differentiation and integration can be interchanged, so the weighted residual to be evaluated is:

$$\frac{\partial}{\partial t}\int_{x_i}^{x_{i+1}} x\, c_e dx$$

Note that the integral is essentially the same as the term evaluated for the decay term. Thus the element matrix equation for the time derivative term is

$$\begin{pmatrix} \frac{\Delta_e}{3} & \frac{\Delta_e}{6} \\ \frac{\Delta_e}{6} & \frac{\Delta_e}{3} \end{pmatrix}\begin{pmatrix} \frac{\partial c_i}{\partial t} \\ \frac{\partial c_{i+1}}{\partial t} \end{pmatrix}$$

The time derivative term is replaced with an implicit finite difference approximation. The part of the approximation containing the known concentration at time t^j is placed on the right-hand side of the equation. The part of the approximation containing the unknown concentrations at time t^{j+1} is added to the left-hand side of the equation. The element matrices for the left and right sides are the same because of the sign change when the knowns are moved to the right-hand side:

$$\frac{\Delta_e}{3\Delta t}$$

The global matrix is assembled and the equations are solved one time step at a time as for the finite difference method. The code below shows how the solution can be implemented in *MATLAB*.

```
% ad_w_decay.m
% This is a finite element solution to calculate transport of
% a dissolved substance undergoing decay.
% dc/dt+u(dc/dx)=D(d2c/dx2)-(lambda)c

% Assume a 100-meter reach with constant concentration boundary
% conditions
% set nodal values of z and delta_t
z=0:1:100;
dz=diff(z);
delta_t=0.2;       % second
% n=number of nodes, e=number of elements
n_nodes=length(z);
n_elements=length(dz);
% set parameter values
u=0.2;       %m/s
D=0.1;       %m^2/s
lambda=0.01;  % per second
% conc at x=0 is one and at x=100 is zero
c0=1;c100=0;
% Set the coefficient matrix.
% preallocate matrix for left and rh sides
% The time derivative is done using a finite difference with
% an implicit approximation.
G=zeros(n_nodes,n_nodes);
R=zeros(n_nodes,n_nodes);
for e=1:n_elements
    node1=e;node2=e+1;          %nodes for the element
    delta=z(node2)-z(node1);    %size of element
    elmat=zeros(2,2);           %initialize element matrix
    rmat=zeros(2,2);            %initialize rhs matrix
    elmat(1,1)=D/delta-u/2+lambda*delta/3+delta/(3*delta_t);
    elmat(1,2)=-D/delta+u/2+lambda*delta/6+delta/(6*delta_t);
    elmat(2,1)=-D/delta-u/2+lambda*delta/6+delta/(6*delta_t);
    elmat(2,2)=D/delta+u/2+lambda*delta/3+delta/(3*delta_t);
    rmat(1,1)=delta/(3*delta_t);
    rmat(1,2)=delta/(6*delta_t);
    rmat(2,1)=delta/(6*delta_t);
    rmat(2,2)=delta/(3*delta_t);
    % assemble the global matrix by adding the element matrix
         G(node1,node1)=G(node1,node1)+elmat(1,1);
         G(node2,node2)=G(node2,node2)+elmat(2,2);
         G(node1,node2)=G(node1,node2)+elmat(1,2);
         G(node2,node1)=G(node2,node1)+elmat(2,1);
    % assemble the matrix for the rhs
         R(node1,node1)=R(node1,node1)+rmat(1,1);
         R(node2,node2)=R(node2,node2)+rmat(2,2);
         R(node1,node2)=R(node1,node2)+rmat(1,2);
         R(node2,node1)=R(node2,node1)+rmat(2,1);
end
```

```
% set the first and last equation for a fixed concentration boundary
% conditions
G(1,:)=0;G(1,1)=1; G(n_nodes,:)=0; G(n_nodes,n_nodes)=1;
R(1,:)=0;R(1,1)=1; R(n_nodes,:)=0; R(n_nodes,n_nodes)=1;
%
% Set the vector of "knowns" using specified concentration values
rhs=zeros(n_nodes,1);
rhs(1)=c0;
rhs(n_nodes)=c100;
endtime=300;       % seconds
% time loop
dz=dz';
time=0;j=1;k=1;
c_old=zeros(n_nodes,1);c_old(1)=rhs(1);c_old(n_nodes)=rhs(n_nodes);
while time<endtime
    rhs=R*c_old;
    c_new=G\rhs;
    if mod(j,300)==0
        conc(:,k)=c_new;
        k=k+1;
    end
    c_old=c_new;
    j=j+1;
    time=time+delta_t;
end
% Plot the profiles.
plot(z,conc)
ylabel('concentration'); xlabel('z (m)');
```

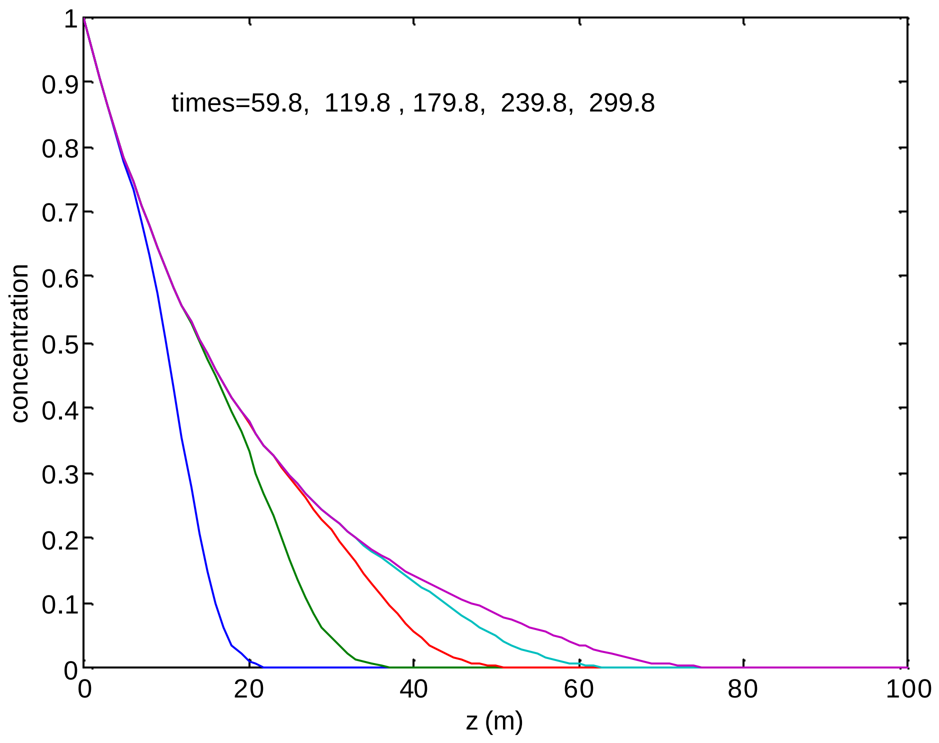


Figure B10.1.1 Finite element solution to advection-dispersion equation.