CHAPTER 5

Numerical Methods for Solving Higher-Order and Boundary-Value Ordinary Differential Equations

Numerical Methods in the Hydrological Sciences
Special Publication Series 57

doi:10.1029/057SP10

5. Numerical Methods for Solving Higher-Order and Boundary-Value Ordinary Differential Equations

5.1. Introduction

Many of the ordinary differential equations encountered in the hydrological sciences are second or higher-order differential equations, including ones for which boundary conditions at two boundaries are specified rather than at just one. Higher-order differential equations have the form

$$\frac{d^n y}{dx^n} = f(x, y, y', \ldots, y^{(n-1)}) \tag{5.1}$$

Hydrological examples include:

1. Steady, uniform flow in one dimension through an unconfined aquifer receiving recharge at rate w. In this case, conservation of mass, Darcy's law, and the Dupuit assumption result in a second-order differential equation for h, the height of the water table: $\frac{d}{dx}\left[Kh \frac{dh}{dx} \right] = -w.$

2. The Theim equation for the drawdown associated with a steady, radial flow to a well in a confined, homogeneous, isotropic aquifer: $\frac{d^2h}{dx^2} + \frac{1}{r}\frac{dh}{dx} = 0.$

To solve second-order equations like these, we must specify two initial or boundary conditions; an n^{th}-order ordinary differential equation requires n initial or boundary conditions.

5.2. Solving higher-order initial value problems

To begin, we'll assume that the given conditions are initial conditions, i.e., they are all specified at the same endpoint of the domain. The initial conditions for a second-order equation, would have the form

$$y_0 = y(x = x_0) = a; \quad y_0' = y'(x = x_0) = b$$

For an n^{th}-order ODE, n-1 derivatives of y at x_0 would be specified.

There are several methods for solving second-order equations, including Taylor series and Runge-Kutta methods. One straightforward general method for solving second and higher-order initial value problems is to transform the equation into a system of simultaneous first-order ordinary differential equations. Then the methods covered in the previous chapter for solving first-order equations, including the *MATLAB* programs `ode23` and `ode45`, can be used to solve the system of equations.

The general approach to reducing an n^{th} order ordinary differential equation to a set of n simultaneous first order equations is to let

$$
\begin{aligned}
y_1 &= y \\
y_2 &= y_1' = y' \\
y_3 &= y_2' = y'' \\
&\vdots \\
y_n &= y_{n-1}' = y^{(n-1)}
\end{aligned}
$$

Equation (5.1) can then be written

$$
\begin{aligned}
y_n' &= f(x, y_1, y_2, \ldots, y_n) \\
y_{n-1}' &= y_n \\
&\vdots \\
y_2' &= y_3 \\
y_1' &= y_2
\end{aligned}
\qquad (5.2)
$$

with appropriate initial conditions. Recalling that the initial values of y and (n-1) of its derivatives are known at $x=x_0$, we set

$$y_1(x_0) = y(x_0),\ y_2(x_0) = y_0'(x_0), \ldots, y_n(x_0) = y^{(n-1)}(x_0).$$

Equations (5.2) are in a vector form that can be used in the *MATLAB* `ode` commands to solve for the y_i's, given the vector of initial conditions.

5.3. Example: Bessel equation

Consider the second-order ordinary differential equation

$$x^2 y'' + xy' + (x^2 - n^2) y = 0$$

which can be written equivalently in the form of equation (5.1) as

$$y'' = -\frac{1}{x} y' - \frac{(x^2 - n^2)}{x^2} y \cdot \qquad (5.3)$$

This is known as the Bessel equation. This equation arises commonly in physical problems, including problems in fluid flow, elasticity, and electrical field theory. A solution to the equation is used to generate the *MATLAB* logo. Solutions to equation (5.3) are known as Bessel functions of order n. For this example, we'll take n = 0, with initial conditions $y(x=0)$ = 1and $y'(x=0)$ = 0.

To rewrite the Bessel equation as a system of first order equations, let $y_1=y$ and $y_2=y'$. Then, the equation (5.3) can be written

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= -y_2 / x - y_1
\end{aligned}
$$

with initial conditions, $y_1(0)$ = 1 and $y_2(0)$ = 0. [Note that equation (5.3) for y'' is undefined at x=0; we will begin the calculation a small distance from 0, e.g., x = 0.001.] We can write a *MATLAB* function file `yprime0.m` to solve the problem. The analytical solution (a series solution) is given by the *MATLAB* function `besselj(nu,x)` with `nu=0`.

```
function yp=yprime0(x,y)
yp=[y(2);-y(2)/x-y(1)];
```

The *MATLAB* commands

```
[x,y]=ode45('yprime0',[0.001 1],[1 0]');
exact=besselj(0,x);
[x y exact]
plot(x,exact,'-k',x,y(:,1),'-b',x,y(:,2),'-r')
```

result in (partial list)

```
ans =
    0.0010    1.0000         0    1.0000
    0.2776    0.9808   -0.1375    0.9808
    0.6330    0.9023   -0.3009    0.9023
    1.2225    0.6598   -0.5039    0.6599
    1.9189    0.2705   -0.5807    0.2708
    2.6368   -0.1140   -0.4602   -0.1139
    3.2905   -0.3420   -0.2244   -0.3422
    3.9011   -0.4016    0.0277   -0.4018
    4.4645   -0.3283    0.2213   -0.3286
    5.0260   -0.1689    0.3302   -0.1691
    5.6307    0.0372    0.3313    0.0372
    6.2212    0.2065    0.2276    0.2066
    6.8112    0.2936    0.0617    0.2938
    7.3941    0.2790   -0.1081    0.2792
    7.9630    0.1800   -0.2290    0.1802
    8.5550    0.0268   -0.2730    0.0269
    9.1537   -0.1265   -0.2244   -0.1265
    9.6663   -0.2176   -0.1242   -0.2177
   10.0000   -0.2457   -0.0433   -0.2459
      (x)       (y)       (y′)      (exact)
```

The solution is illustrated in Figure 5.1. Note that the exact solution and our calculated solution lie on top of each other.

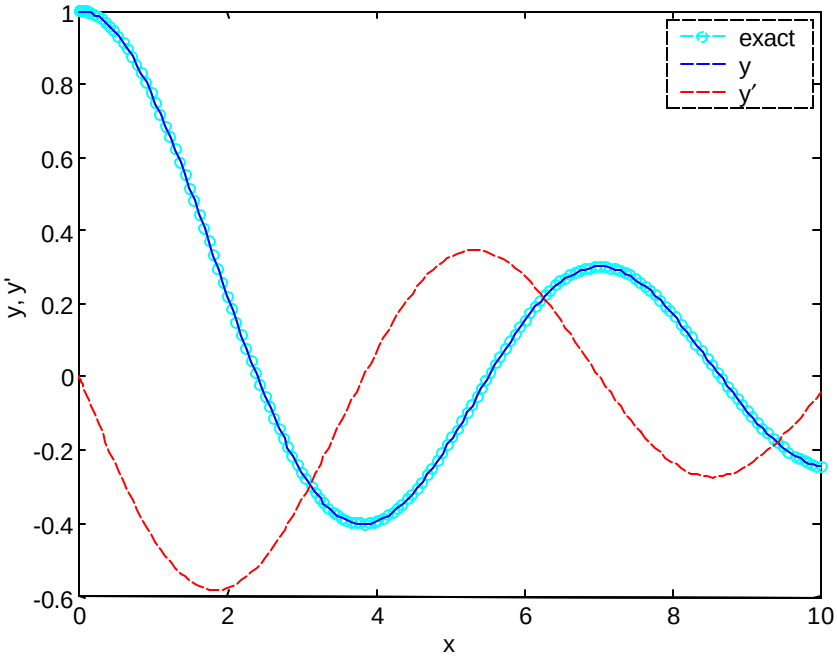


Figure 5.1. Solution to the Bessel equation of order 0 with specified initial values.

5.4. Solving boundary value problems using the shooting method

Boundary value ODEs are problems in which the known conditions are specified at the two end points of the domain. One approach to solving such an equation is to express it as a system of simultaneous first order equations as described above, with the known initial conditions specified and the others estimated. (This approach works best with only one or two unknown initial conditions.) The equation is solved, and the solution at the far boundary compared to the specified condition(s) on that boundary. Assuming the first guess is not correct, it can be successively adjusted until the value at the far boundary matches the boundary condition. This can be done by trial and adjustment, but there are several more efficient approaches for finding the proper condition.

The initial conditions necessary to match the far boundary condition can be found quite easily for linear equations, particularly second and third-order equations. A differential equation is linear if the dependent variable and all of its derivatives appear only as linear terms. Linear equations have the nice property that linear combinations of any two solutions are also solutions. This means that if we make two estimates of the unknown initial condition, g_1 and g_2, which result in the two values of y at the far boundary, v_1 and v_2, then the value of the "guessed" initial condition that will give the desired boundary condition at the far boundary can be calculated as

$$g_{correct} = g_1 + \frac{g_2 - g_1}{v_2 - v_1}\left[BC_{desired} - v_1\right] \tag{5.4}$$

Consider the example equation used above, but with ν=1 and specified boundary conditions rather than initial conditions, y(0) = 0; y(10) = 0.0435 (corresponding to a Bessel equation of order 1).

If $y'(0) = g_1 = 0.5$, then $y(2) = v_1 = 0.0217$. If we repeat the calculation with $y'(0) = g_2 = 1.5$, then the calculation gives $y(2) = v_2 = 0.0650$. Based on these values we find from equation (5.4) that $g_{correct} = 1.0032$.

Using this value in `ode45`:

```
[x,z]=ode45('yprime1',[0 10],[1 1.0032]);
exact=besselj(1,x);
[x y exact]
plot(x1,y1(:,1),'-b',x2,y2(:,2),'-r',x,z,'-g',x,exact,'-k')
```

The results, illustrated in Figure 5.2, show that the method works for second-order, linear differential equations like the Bessel equation.

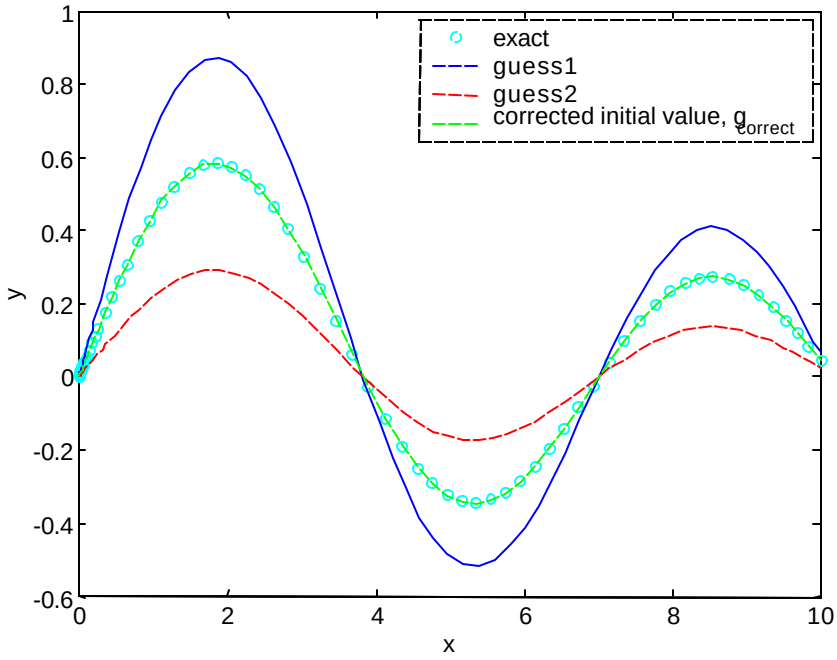


Figure 5.2. Solution to Bessel equation boundary value problem using the shooting method.

If the equation is a third-order, linear, boundary value ODE, then there will be two conditions on one boundary and one on the other. Only one condition must be "shot for" if we start the calculation at the boundary on which two of the three conditions are specified, although this may mean working in negative time or space steps. For fourth-order, linear, boundary value ODE's, two conditions may be given at each boundary. In that case, we could still find the solution by taking a linear combination, but we would need to use four solutions, and the algebra gets more complicated.

If the differential equation is not linear, then this method of combining solutions generally won't work unless the two estimates are quite close to the correct value. We can, however, think of the problem as one of finding the root of the equation given by the difference of the true boundary condition and the estimated values. Therefore, we could use something like the secant method

[Chapter 2.3] to find the initial condition that gives the desired boundary conditions at the far boundary.

5.5. Solving boundary value problems using finite differences

Another approach to solving a boundary value ODE is to express the equation in terms of finite differences. This leads to a set of algebraic equations that can be solved, for example, by Gaussian elimination [Chapter 2.11].

Previously we discussed several ways to estimate derivatives using finite differences [Chapter 3.2]. We found that forward and backward difference estimates of a first derivative have an error of $O(\Delta x)$, while central difference estimates have an error of $O(\Delta x^2)$; we also found estimates for the second derivative with errors $O(\Delta x^2)$. Using the $O(\Delta x^2)$ formulations, we can estimate the first and second derivatives of $y(x)$ as

$$\begin{aligned} y' &= \frac{y_{i+1} - y_{i-1}}{2\Delta x} + O((\Delta x)^2) \\ y'' &= \frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2} + O((\Delta x)^2) \end{aligned} \tag{5.5}$$

To illustrate the use of finite differences for solving ordinary differential equations, we will again use the Bessel equation of order 0, with $y(0) = 1.0000$ and $y(7) = 0.3001$. Substituting the finite difference approximations for y' and y'' into the differential equation (5.3), we obtain

$$x_i \frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2} + \frac{y_{i+1} - y_{i-1}}{2(\Delta x)} + x_i y_i = 0,$$

or

$$\left(x_i - \frac{\Delta x}{2}\right) y_{i-1} + \left(-2x_i + x_i(\Delta x)^2\right) y_i + \left(x_i + \frac{\Delta x}{2}\right) y_{i+1} = 0 \tag{5.6}$$

Equation (5.6) applies for each grid point x_i except the two end points, at which the values of y are known. If we divide the solution interval into n subintervals (n+l grid points), this procedure results in n-1 equations for n-1 unknowns (the values of y at each internal grid point). We can express this system of equations as a matrix equation, $\mathbf{A}y = b$, where A contains the coefficients of the y terms and b is a vector made up of the right-hand sides of the equations.

If we take $\Delta x = 0.5$ ($n = 14$), we obtain 13 equations:

$$\begin{aligned} &0.25y_0 - 0.875y_1 + 0.75y_2 = 0 \\ &0.75y_1 - 1.750y_2 + 1.25y_3 = 0 \\ &1.25y_2 - 2.625y_3 + 1.75y_4 = 0 \\ &\vdots \\ &6.25y_{12} - 11.375y_{13} + 6.75y_{14} = 0 \end{aligned}$$

The first equation above contains y_0 which is known to be 1.0000 from the specified boundary

condition; likewise the last of the equations contains y_{14} which is known to be 0.3001. The set of equations can be written in matrix-vector form as:

$$\begin{vmatrix} -0.875 & 0.750 & 0 & 0 & & & & \\ 0.75 & -1.750 & 1.25 & 0 & & & & \\ 0 & 1.25 & -2.625 & 1.75 & & & & \\ & & & \bullet & & & & \\ & & & & \bullet & & & \\ & & & & & \bullet & & \\ & & & & & 5.25 & -9.625 & 5.75 & 0 \\ & & & & & 0 & 5.75 & -10.50 & 6.25 \\ & & & & & 0 & 0 & 6.25 & -11.375 \end{vmatrix} \begin{vmatrix} y_1 \\ y_2 \\ y_3 \\ \bullet \\ \bullet \\ \bullet \\ y_{11} \\ y_{12} \\ y_{13} \end{vmatrix} = \begin{vmatrix} -0.250 \\ 0.000 \\ 0.000 \\ \bullet \\ \bullet \\ \bullet \\ 0.000 \\ 0.000 \\ -2.0257 \end{vmatrix}$$

Letting A be the coefficient matrix, *y* be the vector of unknowns, and *b* be the right-hand-side vector, we can use the *MATLAB* command `y=A\b` to solve the set of equations. The result is (to 4 decimal places)

x	y	exact	error
0	1.0000	1.0000	0
0.5000	0.9345	0.9385	0.0040
1.0000	0.7569	0.7652	0.0083
1.5000	0.4989	0.5118	0.0129
2.0000	0.2078	0.2239	0.0161
2.5000	-0.0648	-0.0484	0.0164
3.0000	-0.2732	-0.2601	0.0131
3.5000	-0.3864	-0.3801	0.0063
4.0000	-0.3944	-0.3971	-0.0028
4.5000	-0.3086	-0.3205	-0.0119
5.0000	-0.1588	-0.1776	-0.0188
5.5000	0.0146	-0.0068	-0.0214
6.0000	0.1694	0.1506	-0.0187
6.5000	0.2711	0.2601	-0.0110
7.0000	0.3001	0.3001	-0.0000

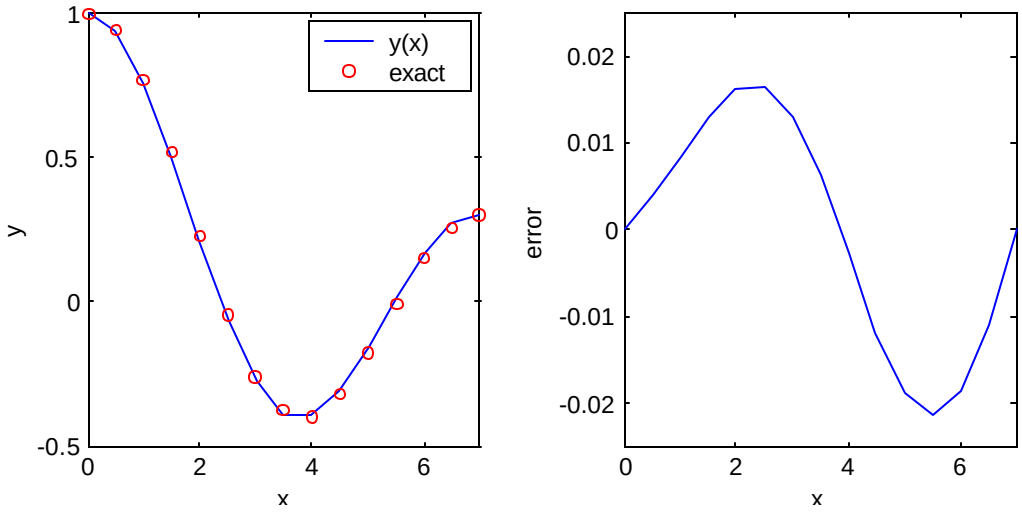


Figure 5.3. Finite difference solution and error for Δx=0.5.

We match the endpoints, but the accuracy is not good because of the coarse grid size. We can make Δx smaller to improve the accuracy, but in that case we probably don't want to type the whole of the matrix A in by hand because the size of the matrix becomes much larger. We can write an m-file to assign the values of A and solve the problem for arbitrary step size, as given below. As we would expect from the $O((\Delta x)^2)$ finite difference estimates we used to generate the finite difference equation, if we want accuracy to 3 decimal places, we need a step size of roughly $\sqrt{0.001} \cong 0.03$.

```
%besselfd.m
%finite difference solution to Bessel equation

dx=input('step size: ')
n=round((7/dx)-1);
dx=7/(n+1);  %adjust dx to give integer number of nodes
x=dx:dx:7-dx;
y1=0; y7=0.3001; %boundary conditions
A=zeros(n,n);
%form matrix A using MATLAB diag command; see 'help diag'
A=diag(-2*x+x*dx^2)+diag(x(2:n)-dx/2,-1)+diag(x(1:n-1)+dx/2,1);
b=zeros(n,1);  %right hand side
b(1)=-(x(1)-dx/2)*y1;   %left boundary condition
b(n)=-(x(n)+dx/2)*y7; %right boundary condition
y=A\b;      %solution
exact=besselj(0,x);
error=exact'-y;
plot(x,error)
xlabel('x'); ylabel('error');
```

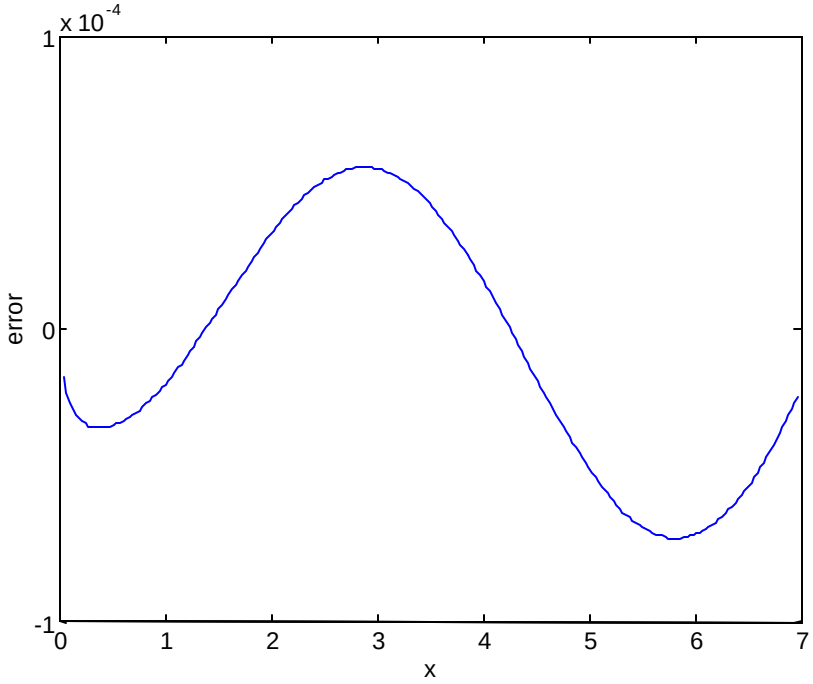


Figure 5.4. Error in finite difference approximation when Δx is reduced to 0.03.

An interesting note in regard to step size is that if we compute values of y for step sizes Δx and Δx /2, we can obtain an improved estimate using Richardson extrapolation. If we begin with $O((\Delta x)^2)$ estimates of the derivatives in our finite difference equation, we can improve the accuracy to $O((\Delta x)^4)$ by calculating a new estimate that is a weighted sum of the results for step Δx and step Δx /2, specifically the "better" value (i.e., done with Δx /2) plus 1/3rd of the difference between the "better" value and the "less good" (i.e., done with Δx). The resulting calculation has error $O((\Delta x)^4)$! Schematically, the calculation is:

$$y_{O((\Delta x)^4)} = y[\Delta x/2] + \frac{1}{3}\{y[\Delta x/2] - y[\Delta x]\}$$

where $y[\Delta x]$ and $y[\Delta x/2]$ refer to the values obtained for step sizes Δx and $\Delta x/2$, respectively.

5.6. Derivative boundary conditions

Derivative boundary conditions are relatively easily handled when the equation is solved as a system of simultaneous first-order equations. In fact, derivative initial conditions must be specified for second and higher order equations. When this method is being used to shoot for a derivative boundary condition at the far end of the domain, we can use the same procedures described above, but applied to the term in the vector y corresponding to that derivative, e.g., $y(2)$ for a condition specified in terms of y'.

The problem is less straightforward when a finite difference approximation is used to solve the equation. To specify a derivative boundary condition in this case, it is necessary to extend the domain beyond the specified interval and then to use finite difference forms of the boundary

conditions to eliminate the fictitious points. This is probably most easily understood in the context of an example.

Consider the second-order equation we have been working with in these examples, but now with a derivative boundary condition specified at $x = 0$,

$$xy'' + y' + xy = 0 \qquad y(0) = 1, \quad y'(7) = 0.0046$$

We need a difference equation for each point at which y is unknown, which now includes $y(7)$. In this case all of the equations are the same as those given above except for the equations involving y_n. For example, with $\Delta x = 0.5$ we have as our equation at $x = 7$ (i.e., x_{14})

$$6.75y_{13} - 12.25y_{14} + 7.25y_{15} = 0$$

and for x_{13}

$$6.25y_{12} - 11.375y_{13} + 6.75y_{14} = 0$$

We had to add the equation at $x = 7$ because y at this location (y_{14}) is now an unknown. Doing this, however, introduces another unknown, y_{15}, into the difference equation; y_{15} indicates a value of y at $x = 7+\Delta x$, a point outside the domain of the problem. To eliminate y_{15} from the difference equation, we employ a finite difference approximation to the boundary condition, $y'(7) = 0.0046$.

$$y'(7) = \frac{y_{15} - y_{13}}{2\Delta x} + O((\Delta x)^2) = 0.0046$$

Thus, to $O((\Delta x)^2)$, we can write $y_{15} = y_{13} + 2\Delta x*0.0046$, and the difference equation for y_{14} becomes

$$14y_{13} - 12.25y_{14} = -0.0046(2\Delta x)(7.25)$$

Now we once again have as many equations as unknowns and can solve the system of equations as before. More details on derivative boundary conditions in finite difference equations are provided in the next chapter.

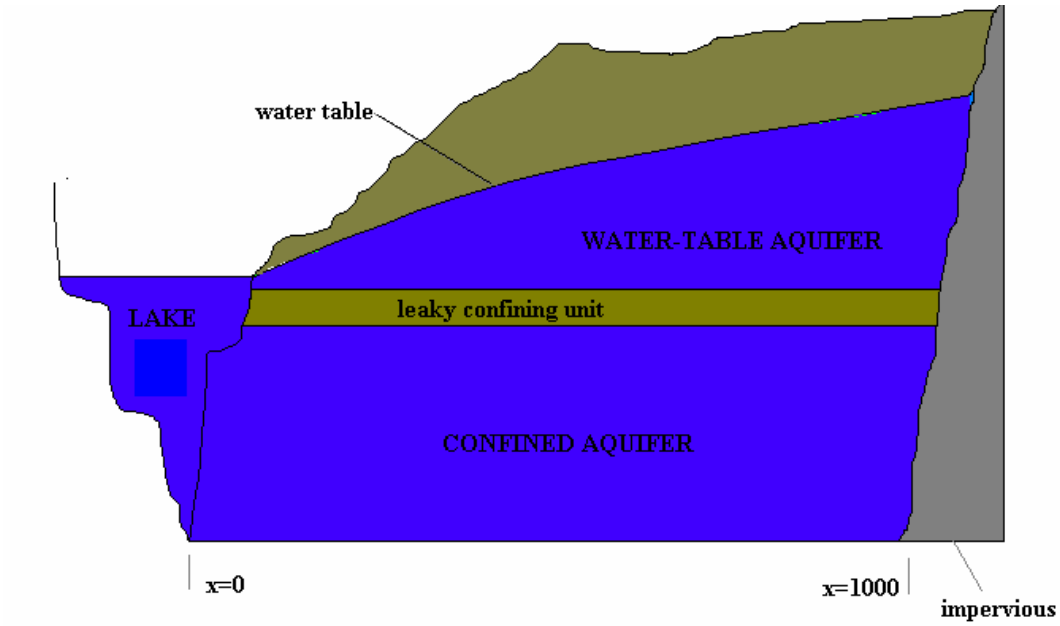
5.7. Problems

1. Solve the equation

$$ff''+2f'''=0$$

for the boundary conditions $f(z=0)=0;\ f'(z=0)=0;\ f'(z\to\infty)=1$. This is the Blasius equation for the velocity distribution in a laminar boundary layer on a flat plate. The velocity profile is given by f' (i.e., df/dz) as a function of z. To solve the equation numerically, the infinite value of z must be replaced by some large value z_{max}; z_{max}= 10 should do for this problem. Rewrite the 3rd-order ordinary differential equation as a system of first-order equations and use the shooting method to solve the problem using `ode45`. You can do this by trial and adjustment. For a challenge, try coding a more efficient method for finding the initial condition $f''(0)$ that gives back the desired boundary condition at z_{max} (i.e., $f'(z_{max})=1$.

2. At one end of a confined aquifer that extends from x = 0 to x = l000m, head is maintained at 100m by a lake (see figure below). The other end of the aquifer (x = l000) is intersected by an impervious unit, so that dh/dx = 0. The aquifer, with transmissivity 2.5x10^{-5} m^2 s^{-1}, is overlain by a 10m-thick, leaky confining bed with hydraulic conductivity 10^{-10} m s^{-1}. The overlying water-table aquifer has a water-table elevation given by $h_{wt}=100+0.06x-0.00003x^2$.



The equation describing the head distribution in the aquifer is

$$\frac{d^2h}{dx^2}=-\frac{C_{confining}}{T}\left(h-h_{wt}\right)$$

where T is aquifer transmissivity and $C_{confining}$ = hydraulic conductivity/thickness of the confining layer. Use the finite difference method to obtain the head distribution $h(x)$ in the aquifer. Examine the solution for different values of Δx.

3. The equations describing the trajectory of a projectile (e.g., an artillery shell) through the atmosphere are based on Newton's second law of motion. The problem is complicated by several aspects of the physical system: (1) drag force exerted on the projectile depends on the square of the velocity; (2) drag is dependent on air density which varies with altitude; (3) drag must be computed from experimental data rather than a theoretical relationship. These complexities dictate that a numerical rather than an analytical solution is necessary.

   Solve the dynamic equations for a projectile in flight.

$$m\frac{d^2x}{dt^2} = -F_D \cos\theta$$

$$m\frac{d^2y}{dt^2} = -F_D \sin\theta - mg$$

   where $F_D = \frac{1}{2} C_D \rho v^2 A$, $v = \sqrt{v_x^2 + v_y^2}$, and θ is the angle between the x axis and the velocity vector. [Note that $\cos\theta$ and $\sin\theta$ can be expressed as v_x/v and v_y/v respectively.] The diameter of the projectile is 0.30 m and its mass is 7 kg. The initial velocity of the projectile is 670 m s^{-1} at an angle of 45°. Table 1 gives values of air density and the speed of sound as a function of altitude. Table 2 gives C_D as a function of Mach number, the ratio of projectile speed to the speed of sound at any given altitude. [You can use the *MATLAB* function `interp1` to obtain values of ρ and C_D for any altitude.]

Table 1.

Altitude (m)	*Air Density (kg m^{-3})*	*Speed of Sound (m s^{-1})*
0.00	1.225	340.16
2000.00	1.008	332.46
4000.00	0.821	324.46
6000.00	0.660	316.18
8000.00	0.526	307.85
10000.00	0.414	299.51
12000.00	0.311	295.24
14000.00	0.228	295.05
16000.00	0.167	295.05
18000.00	0.121	295.05
20000.00	0.088	295.16
22000.00	0.064	296.36
26000.00	0.034	299.06
28000.00	0.025	300.38
24000.00	0.047	297.56
30000.00	0.018	301.77

Table 2.

Mach Number	*Drag Coefficient*
0.0	0.50
0.4	0.52
0.8	0.66
1.2	0.93
1.6	1.03
2.0	1.01
2.4	0.99
2.8	0.97
3.2	0.95
3.6	0.93
4.0	0.92