

## CHAPTER 4

# Numerical Methods for Solving First-Order Ordinary Differential Equations

- 4.1. Ordinary differential equations in hydrology
- 4.2. Euler and Taylor-series methods
- 4.3. Modified Euler method or Euler predictor-corrector method
- 4.4. Runge-Kutta methods
- 4.5. Runge-Kutta-Fehlberg method
- 4.6. *MATLAB* methods

### Example

- 4.7. Multistep methods
- 4.8. Sources of error, convergence, and stability
- 4.9. Systems of equations
- 4.10. Problems
- 4.11. References

## **4. Numerical Methods for Solving First-Order Ordinary Differential Equations**

### **4.1. Ordinary differential equations in hydrology**

Ordinary differential equations (ODEs) arise naturally in the hydrological sciences in cases where we know something about rates of change of variables and about their relationship to other quantities. Several simple examples illustrate.

1. The Green-Ampt equation is used to describe the progress of a wetting front into an initially dry soil (e.g., see Hornberger et al., 1998). The depth of the wetting front  $L_f$  is the variable to be determined as a function of time,  $t$ . The differential equation involves several parameters – the saturated hydraulic conductivity,  $K_s$ , the difference between the saturated moisture content and the initial moisture content,  $\Delta q$ , and the capillary pressure head at the wetting-front,  $y_f$ . The equation describing progress of the infiltration front is:

$$\frac{dL_f}{dt} = \frac{K_s}{\Delta q} \left( \frac{-y_f + L_f}{L_f} \right)$$

2. The hydration of carbon dioxide in water,  $CO_2(aq) + H_2O(l) \Rightarrow H_2CO_3(aq)$ , occurs at a rate proportional to the concentration of  $CO_2$  (e.g., see Lasaga and Kirkpatrick, 1981):

$$\frac{dm_{CO_2}(aq)}{dt} = -km_{CO_2}(aq). \text{ The rate constant, } k, \text{ is } 2 \times 10^{-3} \text{ s}^{-1} \text{ at } 0^\circ\text{C}.$$

In many instances, a *system* of equations arises. We may be interested in the cycling of nutrients through different soil "pools" or in the progress of a geochemical reaction involving several interacting species, for example. In such cases, we expect to write a "rate-of-change" equation for each pool or chemical species. Because of interdependencies among the variables (e.g., uptake of nutrients by trees affects the concentration in soil as well as concentration in the tree), the equations are linked. This is what is meant by a system of equations.

Below we examine several methods for solving first-order ordinary differential equations (including systems of equations).

### **4.2. Euler and Taylor-series methods**

A first-order differential equation has the form

$$\frac{dy}{dx} = f(x, y).$$

Often what we want are values of  $y$  for any value of  $x$  given that  $y=y_0$  at  $x=0$ . That is, we want a solution to the initial-value problem.

If we approximate the derivative  $dy/dx$  as a finite difference,  $(y_{i+1} - y_i)/(x_{i+1} - x_i)$  and consider evaluation of the function at  $[x_i, y_i]$ , we obtain, after rearranging the equation

$$y_{i+1} = y_i + \Delta x f(x_i, y_i) \quad (4.1)$$

where  $\Delta x = (x_{i+1} - x_i)$ . Equation (4-1) is known as the Euler method for solving a first-order ODE numerically. Given an initial value for  $y$  ( $y = y_0$  at  $x = 0$ ) and a step size  $\Delta x$ , the equation is successively applied to find  $y_1, y_2, y_3, \dots, y_n$  at the corresponding values of  $x_1, x_2, x_3, \dots, x_n$ .

Intuitively, we can guess that the accuracy of the results from the Euler method (and other numerical methods) depends on the size of the step,  $\Delta x$ . We can obtain insight about the error by deriving the Euler method using a Taylor series expansion of  $y$ ,

$$y(x + \Delta x) = y(x) + y'(x)\Delta x + y''(x)\frac{(\Delta x)^2}{2!} + y'''(x)\frac{(\Delta x)^3}{3!} + \dots$$

or, using the notation in equation (4.1), we can write the series expansion about the point  $x = x_i$  as,

$$y_{i+1} = y_i + y'_i \Delta x + y''_i \frac{(\Delta x)^2}{2} + \dots \quad (4.2)$$

If the series is truncated after the second term, the Euler method is recovered. [Note that  $y'_i = f(x_i, y_i)$ .] The truncation, or local, error is  $O(\Delta x^2)$ , i.e., is given by  $y''_i (\Delta x)^2 / 2$ . The global error, the accumulated error over the whole range of the solution, is  $O(\Delta x)$ . This can be appreciated by noting that the Euler method for approximation of the derivative is obtained by rearranging equation (4.2):

$$\left. \frac{dy}{dx} \right|_{x=x_i} = y'_i = \frac{y_{i+1} - y_i}{\Delta x} + y''_i \frac{\Delta x}{2} + \dots$$

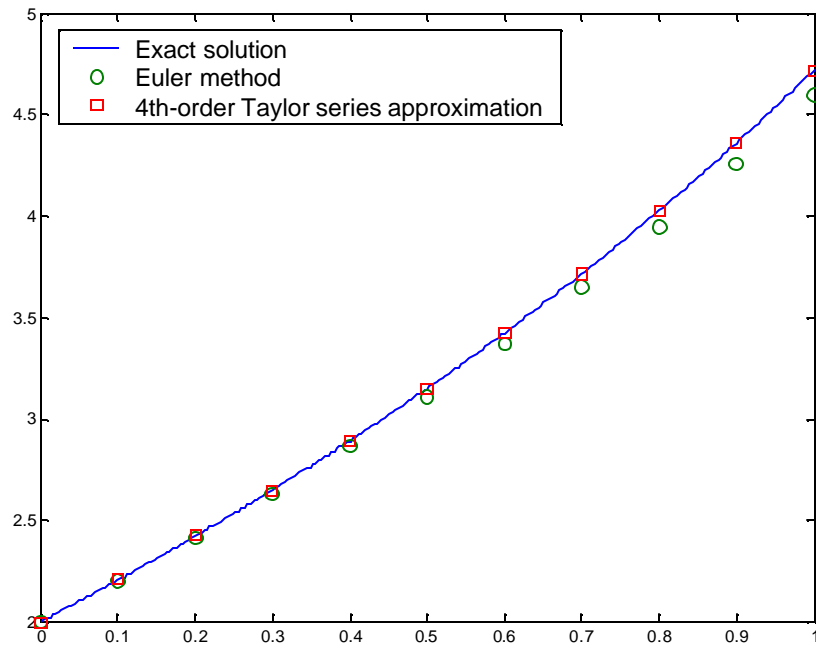
Because we divide through by  $\Delta x$ , the error in the Euler method is  $O(\Delta x)$ . The step size for the Euler method generally must be very small to get good accuracy.

As an example, consider the equation

$$\frac{dy}{dx} = y - x \quad (4.3)$$

with initial condition  $y_0 = 2$ . The exact solution<sup>1</sup> for this problem is  $y = e^x + x + 1$ . A *MATLAB* code for the Euler solution for this problem is given below. For  $\Delta x = 0.1$  the difference between the known exact solution and the Euler solution can be seen to be growing after only about ten steps (Figure 4.1).

<sup>1</sup> This solution can be obtained using the *MATLAB* symbolic toolbox: `y=dsolve('Dy=y-x','y(0)=2','x')`.



**Figure 4.1.** Example solution with the Euler and Taylor series methods for  $\Delta x=0.1$ .

```
% euler_ex.m
delta_x=input('delta_x= ')
x0=0;
xf=1;
x=(x0:delta_x:xf)'; %domain of solution
n=length(x);
y=zeros(size(x));

y(1)=2; %initial condition
for i=1:n-1
    y(i+1)=y(i)+delta_x*yprime(x(i),y(i));
end
xx=x0:delta_x/25:xf;
exact=exp(xx)+xx+1; %exact solution
plot(xx,exact,x,y,'o')
title('Euler solution to dy/dx=x-y')
```

```
function yp=yprime(x,y)
yp=y-x
```

By retaining more terms in the Taylor series, we can derive methods that have higher-order accuracy (i.e., methods with global errors of  $O(\Delta x^2)$ ,  $O(\Delta x^3)$  or as high as we wish) and for which larger values of  $\Delta x$  can be used and still retain reasonable accuracy. For example, consider a second-order Taylor-series method derived by retaining terms up to and including  $y''$  in the expansion.

$$y_{i+1} = y_i + y'_i \Delta x + y''_i \frac{(\Delta x)^2}{2!} + y'''_i \frac{(\Delta x)^3}{3!} + \dots \quad (4.4)$$

Consider the example we used in the previous section,  $y' = y - x$ . The derivatives can be evaluated directly from the function:

$$y' = y - x; \quad y'' = y' - 1; \quad y''' = y'' = y' - 1.$$


Substituting for these derivatives in equation (4.4) gives

$$y_{i+1} = y_i + (y_i - x_i) \Delta x + (y_i - x_i - 1) \frac{(\Delta x)^2}{2!} + O((\Delta x)^3) \quad (4.5)$$

The initial condition is  $y(x=0) = 2$ . Let  $\Delta x = 0.1$ . Starting with  $i = 0$  in equation (4.5), we get

$$y_1 = 2 + (2)0.1 + (2 - 1)0.005 = 2.2050$$

Using this value of  $y_1$  at  $x_1 = 0.1$ , we obtain  $y_2 = 2.4210$ . This stepping can be repeated to obtain an estimate of  $y$  at successive values of  $x_i$ . The *MATLAB* m-file `taylor_ex.m` listed below solves the example equation for several orders of Taylor approximations. The results from the program show how the accuracy of the solution increases as additional terms in the Taylor series approximation are retained.



```
%taylor_ex.m
%Calculates 4 solutions to y'=y-x with initial condition y(x=0)=2
%based on 2, 3, 4 and 5 terms of the Taylor series.
delta_x=input('delta_x = ')
x0=0;
xf=1.;
x=(x0:delta_x:xf)';
n=length(x);
y=zeros(size(x));
y1(1)=2; y2(1)=2; y3(1)=2; y4(1)=2; %initial conditions
for i=1:n-1,
    yp1=yprime(x(i),y1(i));
    y1(i+1)=y1(i)+delta_x.*yp1; %two terms - simple Euler method
    yp2=yprime(x(i),y2(i));
    y2(i+1)=y2(i)+delta_x.*yp2+(delta_x.^2)./2.*(yp2-1); %three terms
    yp3=yprime(x(i),y3(i));
    y3(i+1)=y3(i)+delta_x.*yp3+(delta_x.^2)./2.*(yp3-1)...
        +(delta_x.^3)./6.*(yp3-1); %four terms
    yp4=yprime(x(i),y4(i));
    y4(i+1)=y4(i)+delta_x.*yp4+(delta_x.^2)./2.*(yp4-1)...
        +(delta_x.^3)./6.*(yp4-1)+(delta_x.^4)./24.*(yp4-1); %five terms
end;
exact=exp(x)+x+1;[x exact y1' y2' y3' y4' (y4'-exact).*1e4]
```

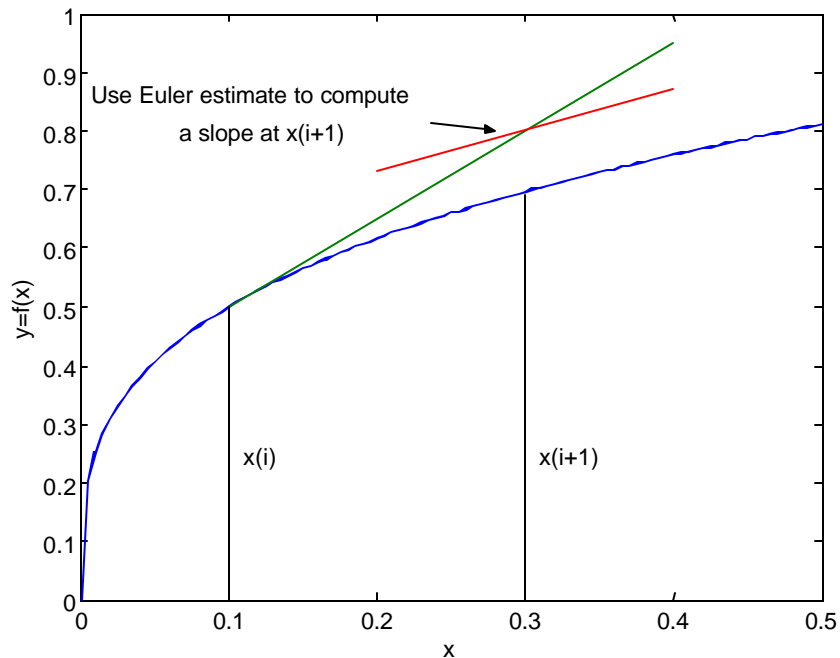
The 5-term ( $O(\Delta x)^4$ ) solution is shown in Figure 4.1. The disadvantages of the higher-order Taylor-series methods for obtaining higher accuracy are that they are cumbersome and involve many computations (i.e., they are slow).

#### 4.3. Modified Euler method or Euler predictor-corrector method

In the Euler method, we estimate each value of  $y$  based on a linear extrapolation from the last computed value. The more nearly linear the function or the smaller the step size, the better this estimate is likely to be. We could expect that our accuracy would be improved, however, if we could make a better estimate of the average slope of the function over each interval as we step through the solution. If we knew in advance the value of  $y'$  at  $x$  and  $x + \Delta x$ , we could use the mean of these slopes to improve our estimate:

$$y_{i+1} = y_i + \Delta x \frac{y'_i + y'_{i+1}}{2} \quad (4.6)$$

We don't know  $y_{i+1}$  or  $y'_{i+1}$  in advance, but once we compute  $y_{i+1}$  using the Euler method (Figure 4.2), we can use the computed value to estimate  $y'_{i+1}$ . This value can then be used to get an improved, or "corrected" value of  $y_{i+1}$ . The difference between the two estimates of  $y_{i+1}$  provides an estimate of the accuracy of the approximation.



**Figure 4.2.** Schematic for modified Euler method.

The modified method then consists of two steps. First, the Euler method (equation 4.1) is used to *predict* (compute an estimate of)  $y_{i+1}$ . Next this predicted value is *corrected* (improved) by using equation (4.6). The local error for this method is  $O(\Delta x)^3$ , while the global error is  $O(\Delta x)^2$  (e.g., see Gerald and Wheatley, 1999). This is one order better than the simple Euler method.

#### 4.4. Runge-Kutta methods

The idea of using information about the function  $y(x)$  at points other than the initial point of an interval can be generalized to produce more efficient and more accurate schemes for solving ordinary differential equations. Probably the most widely used of these are the methods of Runge and Kutta.

The second-order Runge-Kutta method is a good illustration of the approach. The general method is represented by:

$$y_{i+1} = y_i + ak_1 + bk_2 \quad (4.7)$$

where

$$\begin{aligned} k_1 &= f(x_i, y_i)\Delta x \\ k_2 &= f(x_i + \mathbf{a}\Delta x, y_i + \mathbf{b}k_1)\Delta x \end{aligned}$$

and  $a, b, \mathbf{a}$  and  $\mathbf{b}$  are constants to be selected. If we set  $a=1$  and  $b=0$ , we get the simple Euler method. If  $a=b=1/2$  and  $\mathbf{a}=\mathbf{b}=1$ , we recover the modified Euler method.

The second-order Runge-Kutta formula is obtained by setting the values of the four parameters  $a, b, \mathbf{a}$  and  $\mathbf{b}$  to make the expression for  $y_{i+1}$  agree with the Taylor series through the second-order in  $\Delta x$ . The Runge-Kutta methods are derived by rewriting  $k_2$  in terms of the function  $f$  at  $[x_i, y_i]$ . This can be done using a Taylor series expansion of  $f(x_i, y_i)$ . The general form of a Taylor series for a function of two variables is

$$\begin{aligned} f(x + \Delta x, y + \Delta y) &= f(x, y) + f_x(x, y)\Delta x + f_y(x, y)\Delta y \\ &+ \frac{1}{2} \left[ f_{xx}(\Delta x)^2 + 2f_{xy}(\Delta x\Delta y) + f_{yy}(\Delta y)^2 \right] + \dots \end{aligned}$$

where the  $x$  and  $y$  subscripts stand for partial derivatives. This allows us to approximate  $k_2$  as

$$k_2 \cong \Delta x \left[ f(x_i, y_i) + (f_x \mathbf{a} \Delta x + f_y \mathbf{b} \Delta x)_i \right]$$

with truncation error  $O((\Delta x)^2)$ . Substituting this expression for  $k_2$  into equation (4.7) gives

$$\begin{aligned} y_{i+1} &= y_i + af\Delta x + b\Delta x(f + \Delta x\mathbf{a}f_x + \Delta x\mathbf{b}f_yf) \\ &= y_i + \Delta x f(a+b) + (\Delta x)^2 (b\mathbf{a}f_x + b\mathbf{b}f_yf) \end{aligned} \quad (4.8)$$

where  $f$  and its derivatives are evaluated at  $x_i, y_i$ . We want to match the terms of equation (4.8) to the first 3 terms of the Taylor series for  $y_{i+1}$ ,

$$y_{i+1} = y_i + y'_i \Delta x + \frac{(\Delta x)^2}{2} y''_i + \dots = y_i + f\Delta x + \frac{(\Delta x)^2}{2} (f_x + f_y f) + \dots \quad (4.9)$$

The substitution for  $y''$  after the second equal sign comes directly from  $dy/dx = f(x, y)$  by taking the derivative of the right-hand side. For the terms of equations (4.8) and (4.9) to match, we must take  $a+b=1$ ,  $\mathbf{a}b=1/2$ , and  $\mathbf{b}b=1/2$ . These relationships do not uniquely specify the four parameters, but once we choose one, the other three are set. If we pick  $a=1/2$ , then  $b=1/2$ ,  $\mathbf{b}=1$

1 and  $\mathbf{a} = \mathbf{I}$ ; as we noted above, this combination of values gives the modified Euler method.

The fourth-order Runge-Kutta method is a commonly used algorithm. The approach is the same as that for the second-order method, but now we define

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= \Delta x [f(x_i, y_i)] \\ k_2 &= \Delta x [f(x_i + \Delta x / 2, y_i + k_1 / 2)] \\ k_3 &= \Delta x [f(x_i + \Delta x / 2, y_i + k_2 / 2)] \\ k_4 &= \Delta x [f(x_i + \Delta x, y_i + k_3)] \end{aligned}$$

where the coefficients have been set to their commonly used values. Matching the equation for  $y_{i+1}$  to the first 5 terms of the Taylor series yields 11 equations for 13 unknown coefficients, of which two may be chosen arbitrarily. The local error term for the fourth-order Runge-Kutta method is  $O((\Delta x)^5)$  because we matched terms up to  $O((\Delta x)^4)$  in setting the coefficients. The global error is  $O((\Delta x)^4)$ . The method is more efficient than the Euler method because the step size can be much larger even though there are more function evaluations per step.

#### 4.5. Runge-Kutta-Fehlberg method

A goal of most numerical solutions to differential equations is to attain the desired accuracy with the fewest computations, which, for any given method means running it with the largest step size that satisfies the accuracy conditions. How can we determine what that step size is, particularly inasmuch as we generally don't know the solution in advance? One strategy would be to halve the step size and compare the answers with two step sizes. Because smaller step sizes generally lead to more accurate results, a significant difference between the two answers would suggest that the step size is too big. This process can be continued until an acceptably small difference between the two estimates is obtained. Although this approach is straightforward, it can be computationally intensive.

An approach that requires fewer function evaluations is to compare the results using two Runge-Kutta methods of different order, e.g. a second and third-order method or a fourth and fifth-order method. This is called the Runge-Kutta-Fehlberg method. The resulting values of  $y_{i+1}$  are compared, and  $\Delta x$  is adjusted until the difference between values computed using the two methods is as small as desired. By making use of the same  $k$ 's in both methods, the number of function evaluations can be minimized - only six are required for the fourth- and fifth-order Runge-Kutta formulas. Forms for the  $k$ 's and values of the coefficients can be found in texts on numerical analysis (e.g., see Gerald and Wheatley, 1999).

#### 4.6. MATLAB methods

*MATLAB* has several functions to solve ordinary differential equations, including `ode23` and `ode45`. They are both implementations of the Runge-Kutta-Fehlberg method, using second-third-order and fourth-fifth-order formulas, respectively. The syntax of both commands is similar. From



the *MATLAB* documentation<sup>2</sup>:

ODE45 Solve non-stiff differential equations, medium order method.  
 [T,Y] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the system of differential equations  $y'=f(t,y)$  from time T0 to TFINAL with initial conditions Y0. Function ODEFUN(T,Y) must return a column vector corresponding to  $f(t,y)$ . Each row in the solution array Y corresponds to a time returned in column vector T. To obtain solutions at specific times T0, T1, ..., TFINAL (all increasing or all decreasing), use TSPAN = [T0 T1 ... TFINAL].

[T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default integration properties replaced by values in OPTIONS, an argument created with the ODESET function. See ODESET for details. Commonly used options are scalar relative error tolerance 'RelTol' (1e-3 by default) and vector of absolute error tolerances 'AbsTol' (all components 1e-6 by default).

[T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS,P1,P2,...) passes the additional parameters P1,P2,... to the ODE file as ODEFUN(T,Y,P1,P2,...) and to all functions specified in OPTIONS. Use OPTIONS = [] as a place holder if no options are set.

The ode commands in *MATLAB* require a function m-file defining the differential equation, e.g. f.m. The default accuracy for ode23 is  $1 \times 10^{-3}$  (1E-3); and for ode45 is  $1 \times 10^{-6}$  (1E-6). The default accuracy can be changed using "odeset" to adjust the options (see *MATLAB* help for details).

### Example

The equation

$$m_s \frac{dw}{dt} = (m_s - \mathbf{r}V)g - \frac{1}{2}\mathbf{r}C_D A w^2$$

describes the acceleration to terminal velocity of a spherical object of mass  $m_s$  and volume  $V$  released in a non-moving fluid of density  $\mathbf{r}$ , where  $w$  is the vertical velocity of the sphere,  $g$  is gravitational acceleration,  $C_D$  is the drag coefficient, and  $A$  is the cross-sectional area of the sphere. For a small sphere ( $\mathbf{R}_D = wD\mathbf{r}/\mathbf{m} < 0.5$ ), where  $D$  is sphere diameter and  $\mathbf{m}$  is fluid viscosity),  $C_D = 24/\mathbf{R}_D$  and the terminal velocity,  $w_s$ , is given by Stokes law:

$$w_s = \frac{(\mathbf{r}_s - \mathbf{r}) g D^2}{18\mathbf{m}}$$

where  $\mathbf{r}_s$  is the density of the sphere. We can check our solution by comparing it to  $w_s$  after  $w$  reaches a constant value (i.e. its terminal velocity, also referred to as settling velocity or fall velocity). A *MATLAB* code to solve this equation is given below for an example in which  $D = 3 \times 10^{-5}$  m (30  $\mu\text{m}$ ),  $\mathbf{r}_s = 2650 \text{ kg m}^{-3}$ ,  $\mathbf{r} = 1000 \text{ kg m}^{-3}$ , and  $\mathbf{m} = 0.001 \text{ Pa}\cdot\text{s}$ .

---

<sup>2</sup> Reprinted with permission from MathWorks, Inc.

`%ode_ex.m`

```
[t,w]=ode23('wprime',[0 0.001],0);
[t2,w2]=ode45('wprime',[0 0.001],0);
opt=odeset('RelTol',1e-6); %reset tol value
[t3,w3]=ode23('wprime',[0 0.001],0,opt);

%calculate Stokes settling velocity
rho=1000; rhos=2650; mu=1.3e-3; g=9.81; D=.3e-4;
ws=(rhos-rho)*g*D.^2/(18*mu);
%check if particle Reynolds number < 0.5 as required for Stokes equation
Rd=ws*D*rho/mu;
if Rd>0.5, fprintf('Stokes eqn invalid'), end;
%check that particle Reynolds number < 800 as required
% for the Schiller et al. (1933) drag coefficient formula.
if Rd>800, fprintf('Drag coefficient invalid'), end;

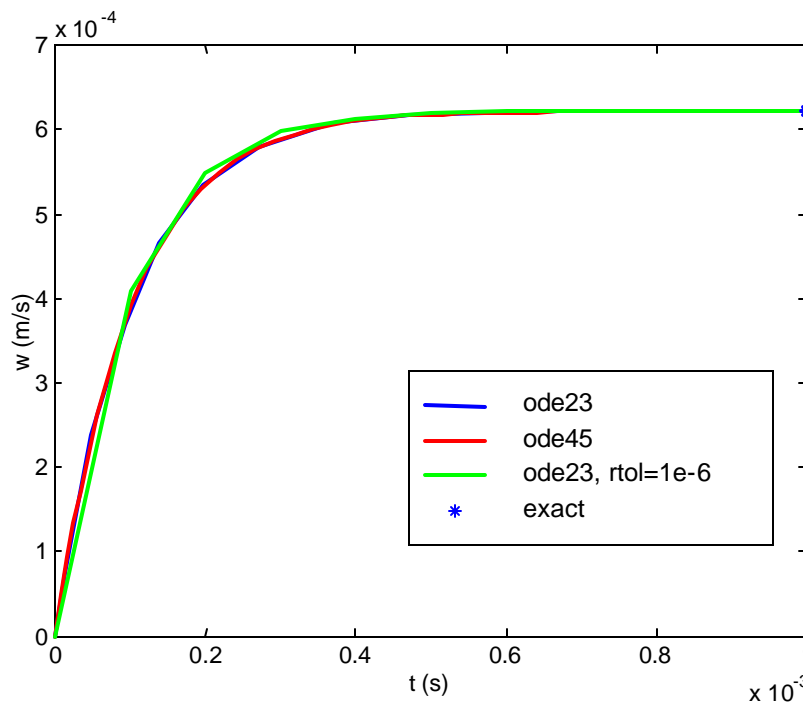
plot(t,w,'-b',t2,w2,'-r',t3,w3,'-g',max(t),ws,'*')
xlabel('t (s)'); ylabel('w (m/s)')
legend('ode23','ode45','ode23, rtol=1e-6','exact')
```



```
function wp=wprime(t,w)

rho=1000; rhos=2650; mu=1.3e-3; g=9.81;
D=.3e-4; V=pi/6*D.^3; A=pi/4*D.^2; ms=rhos*V;
Rd=rho*D*w/mu; Cd=0;
if w~=0, Cd=24/Rd*(1+0.150*Rd^0.687); end; %let Cd=0 if w=0
a=g*(ms-rho*V)/ms; b=0.5*rho*Cd*A ./ms;
wp=a-b*w.^2;
```

The solutions are shown in Figure 4.3. For this example,  $\mathbf{R}_D < 0.5$ , so comparing the calculated terminal velocity to Stokes equation is appropriate. For larger  $\mathbf{R}_D$ , the drag coefficient for spheres takes a different form, one that cannot be represented by a simple algebraic expression. For values of  $\mathbf{R}_D$  up to 800, the approximate formula of Schiller et al. (1933; see Graf, 1971) can be used,  $C_D = (24/\mathbf{R}_D)(1 + 0.15\mathbf{R}_D^{0.687})$ , as in the m-file above. Note that the acceleration time is very short for small particles like silt and sand falling through water and generally can be (and is) neglected in calculations of particle settling.



**Figure 4.3.** Solution to the example problem using *MATLAB* ode functions.

#### 4.7. Multistep methods

All of the methods described above are examples of single-step methods, that is, they use information about the solution at just one location,  $x_i$ , to advance the solution to  $x_{i+1}$ . Single step methods have a number of advantages, including being self-starting and having the flexibility to change step size from one step to the next. Another approach is to use information about the solution at several previous locations, thus more fully utilizing what we know about the function. This is the idea behind the multistep methods. Most methods use equally spaced points to facilitate the calculations. This limits flexibility in terms of changing step size, but the methods can be very efficient.

Most multistep methods use past (already computed) values to construct a polynomial that approximates the derivative of the function and uses that to extrapolate to the next point. The number of past points used determines the order of the polynomial we can fit, and therefore the truncation error. Using two previous points would allow a quadratic approximation, and would result in a  $O((\Delta x)^3)$  global error. Because in general we don't initially know values for  $y$  at the first three points, this method is not self-starting. It is necessary to use other methods, e.g. a Runge-Kutta method, to get values at, say, the first three points, after which a three-point multistep method can be used. See a text on numerical analysis, e.g. Gerald and Wheatley (1999), for details on multistep methods.

#### 4.8. Sources of error, convergence, and stability

It is important to keep in mind that there are three possible sources of error in numerical

calculations, such as solutions to ordinary differential equations. The one we have discussed the most is truncation error, the error associated with the number of terms in a series, e.g. Taylor series. Other sources of error are round-off errors, which will always be present even if our method is exact, and original data errors, associated with not knowing the initial conditions or boundary conditions exactly [see Box 3.1]. Errors at each step propagate through the solution, so the global error is generally about an order larger than the local error.

A method is convergent if the approximate solution tends toward the true solution as  $\Delta x \rightarrow 0$ . All of the widely used methods for solving ordinary differential equations (in particular the ones we have discussed) are convergent. Stability refers to the growth of errors as the solution proceeds. A stable method is one in which the global error does not grow in an unbounded manner. For many ODE's, the step size required to obtain an accurate solution is much smaller than the step size required for stability. As a result, stability is often not a major concern. When a solution is unstable, it is usually obvious. A smaller step size will generally rectify the problem, although there are cases that are unconditionally unstable for some methods. Changing methods is the best approach in such cases.

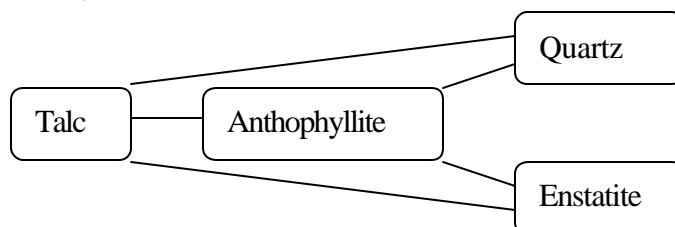
#### 4.9. Systems of equations

The differential equations we have considered so far are simple first-order ordinary differential equations. There are other problems described by several coupled, first-order differential equations. The predator-prey equations (e.g., see Kot 2001) used in population ecology are an example. This problem is defined by the two equations

$$\begin{aligned}y_1' &= (1 - \mathbf{a} y_2) y_1 \\y_2' &= (-1 + \mathbf{b} y_1) y_2\end{aligned}$$

The size of the prey population is given by  $y_1$ , and the predator population by  $y_2$ . The prey population,  $y_1$ , decreases for large values of  $y_2$ , which in turn decreases the predator population, which increases the number of prey, and so forth. The *MATLAB* ode routines can solve a set of equations such as this by defining  $y$  and  $y'$  as column vectors; the initial conditions are also given by a column vector. (The m-file defining the equations must return the  $y'$  values as a vector.) The solution is a matrix in which the first column is  $y_1$  and the second column is  $y_2$ .

As an example of solving a system of equations, consider the chemical reaction defining the dehydration of talc,



The rates of the reactions are well described by first-order kinetics (rate proportional to amount) so the differential equations are:

$$\frac{dy_1}{dt} = a_{11}y_1$$

$$\frac{dy_2}{dt} = a_{21}y_1 + a_{22}y_2$$

$$\frac{dy_3}{dt} = a_{31}y_1 + a_{32}y_2$$

$$\frac{dy_4}{dt} = a_{41}y_1 + a_{42}y_2$$

where  $y_1$  refers to talc,  $y_2$  refers to anthophyllite,  $y_3$  refers to enstatite, and  $y_4$  refers to quartz. Expressing the constituents in terms of volume and for a temperature of 830°C and a pressure of 1000 bars, the coefficients are (see Greenwood, 1963, but be advised that some tables are mislabeled):  $a_{11} = -18.0 \times 10^{-4} \text{ min}^{-1}$ ,  $a_{21} = 11.0 \times 10^{-4} \text{ min}^{-1}$ ,  $a_{22} = -5.9 \times 10^{-4} \text{ min}^{-1}$ ,  $a_{31} = 3.9 \times 10^{-4} \text{ min}^{-1}$ ,  $a_{32} = 4.8 \times 10^{-4} \text{ min}^{-1}$ ,  $a_{41} = 2.1 \times 10^{-4} \text{ min}^{-1}$ ,  $a_{42} = 0.5 \times 10^{-4} \text{ min}^{-1}$ . The set of equations can be rewritten in matrix form as

$$\frac{d\mathbf{y}}{dt} = \begin{pmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & 0 & 0 \\ a_{41} & a_{42} & 0 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

We can write a function file to define the equations. Call this `rates.m`.



```
function ydot=rates(t,y)
% function for the dehydration of talc
a=zeros(4,4);
a(1,1)=-18.0e-4; % all a's are min^-1
a(2,1)=11.0e-4; a(2,2)=-5.9e-4; a(3,1)=3.9e-4;
a(3,2)=4.8e-4; a(4,1)=2.1e-4; a(4,2)=0.5e-4;
ydot=a*y;
```

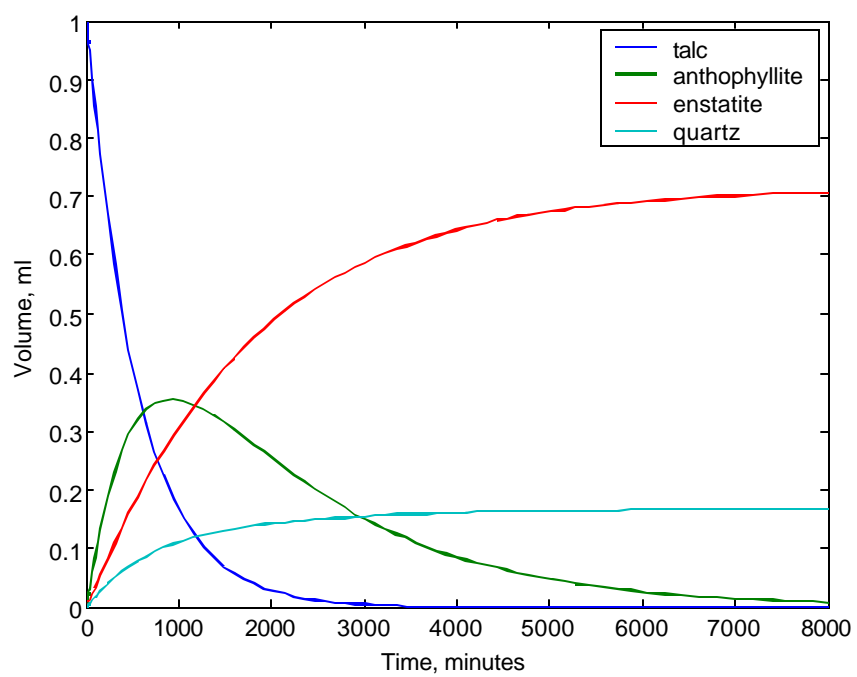
If we start at time zero with a unit volume of talc and consider the evolution over 8000 minutes, we can obtain the solution with the *MATLAB* command

```
[t,y]=ode45('rates',[0 8000],[1 0 0 0]');
```

The results can be plotted (Figure 4.4) with the commands

```
plot(t,y)
xlabel('Time, minutes')
ylabel('Volume, ml')
legend('talc','anthophyllite','enstatite','quartz')
```

In cases where there are several dependent variables (e.g., with a system of equations rather than just a single equation), it sometimes is useful to plot the dependent variables against each other and not simply view each variable as a function of time [Box 4.1].



**Figure 4.4.** Results from *MATLAB* integration of talc equations.

#### 4.10. Problems

1. As mentioned in Section 4.1, the Green-Ampt equation for the movement of a wetting front into a dry soil is

$$\frac{dL_f}{dt} = \frac{K_s}{\Delta q} \left( \frac{-y_f + L_f}{L_f} \right)$$

Solve this equation using the *MATLAB* functions `ode23`, `ode45`, a standard fourth-order Runge-Kutta method, and the Euler method. Take  $K_s = 3 \times 10^{-6} \text{ m s}^{-1}$ ,  $\Delta q = 0.2$ , and  $y_f = -0.2 \text{ m}$ .

Compare the accuracy of the methods. The analytical solution is:

$$t = \frac{L_f \Delta q}{K_s} + \frac{y_f \Delta q}{K_s} \log_e \left( 1 + \frac{L_f}{-y_f} \right).$$

2. When open channel flow encounters a change in bed slope, there is an adjustment in flow depth (see, e.g., Henderson 1970). Assuming water depth and bed elevation vary gradually, the change of depth with downstream distance is given by

$$\frac{dh}{dx} = \frac{S_b - S_f}{1 - F^2}$$

where  $S_f$  is energy slope,  $S_b$  is channel bed slope,  $h$  is flow depth,  $F$  is Froude number,  $U/\sqrt{gh}$ ,  $U$  is channel mean velocity  $= Q/A$ ,  $A$  is channel cross-sectional area and  $g$  is gravitational acceleration ( $9.81 \text{ m s}^{-2}$ ).  $S_f$  is given by Manning's equation (as a function of  $h$  in a rectangular channel) for a given  $Q$ ,  $n$ , and channel width,  $w$  [see Box 2.2]. Let  $Q = 20 \text{ m}^3 \text{ s}^{-1}$ ,  $S_b = 0.0008$ ,  $n = 0.015$ , and  $w = 15 \text{ m}$ . Use `ode45` to solve for  $h(x)$  from  $x = 0 \text{ m}$  to  $x = 150 \text{ m}$  for an initial flow depth of  $h = 0.8 \text{ m}$  at  $x = 0$ . Plot the result.

3. The interaction between hosts and parasites can be described by a pair of coupled first-order ordinary differential equations:

$$\begin{aligned} \frac{dP}{dt} &= P - a \frac{P^2}{H} \\ \frac{dH}{dt} &= H - bPH \end{aligned}$$

where  $P$  represents the population of parasites and  $H$  represents the population of hosts. Let  $a = 2.5$  and  $b = 0.1$ . At  $t = 0$ ,  $P = 6$  and  $H = 15$ . Use the *MATLAB* ode functions to solve for  $P$  and  $H$  from  $t = 0$  to  $t = 20$ . Plot  $P$  and  $H$  against  $t$  and plot  $P$  against  $H$ .

4. When an organic waste is discharged into a stream, a biochemical oxygen demand (BOD) is exerted; that is, bacterial decomposition of the organic waste uses oxygen and thus depletes dissolved oxygen in the water (e.g., Hemond and Fechner, 1994). Dissolved oxygen (DO) is replenished through atmospheric reaeration (by diffusion through the air-water interface). If a

point discharge into a river with steady uniform flow is envisioned, a typical pattern of DO, called a "sag" is observed. Below the waste outfall, DO drops in response to the BOD. As the decomposition of the waste proceeds, the BOD is decreased and atmospheric reaeration replenishes the DO. DO decreases downstream to a "critical value" (a minimum) and then gradually returns to ambient conditions. The process can be modelled using a form of the Streeter-Phelps equations (Streeter and Phelps, 1925):

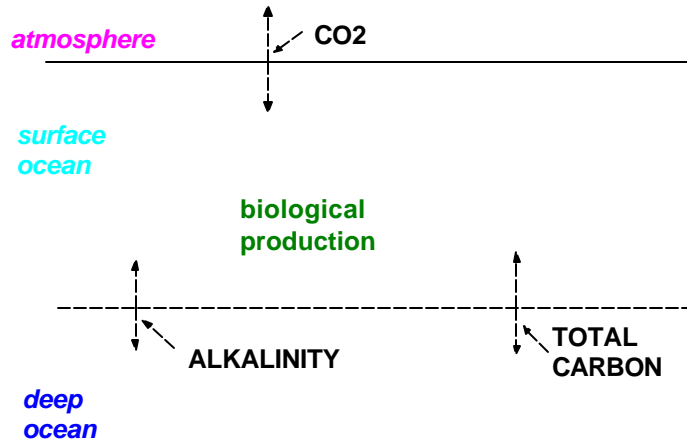
$$\begin{aligned}\frac{dL_C}{dx} &= -(K_C / U)L_C \\ \frac{dL_N}{dx} &= -(K_N / U)L_N \\ \frac{dc}{dx} &= -(K_C / U)L_C - (K_N / U)L_N + (K_a / U)(c_s - c)\end{aligned}$$

where  $L_C$ =carbonaceous BOD,  $L_N$ =nitrogenous BOD,  $c$ =DO concentration,  $c_s$ =saturation DO concentration,  $U$  is mean velocity, and the  $K$ 's are rate constants.

Solve the Streeter-Phelps equations and plot the results for  $K_C=1.2 \text{ day}^{-1}$ ,  $K_N=0.8 \text{ day}^{-1}$ ,  $K_a=1.9 \text{ day}^{-1}$ ,  $L_C(0)=5 \text{ mg L}^{-1}$ ,  $L_N(0)=8 \text{ mg L}^{-1}$ , river velocity= $10 \text{ km day}^{-1}$ ; all rate constants are for  $T=20^\circ\text{C}$ . Temperature adjustments for the rate constants are:  $K_C=K_C(20)\times(1.024)^{(T-20)}$ ,  $K_N=K_N(20)\times(1.024)^{(T-20)}$ , and  $K_a=K_a(20)\times(1.047)^{(T-20)}$ . Saturation DO concentration is a function of  $T$  as well:  $c_s=14.652-0.41022T+0.007991T^2-0.00077774T^3$ . Consider results for  $T=5^\circ\text{C}$  and  $T=20^\circ\text{C}$ .

5. An issue of current concern is the fate of carbon dioxide being added to the atmosphere by the burning of fossil fuel. Over the long term, oceanographers view the problem as one of partitioning carbon among the atmosphere, the surface ocean, and the deep ocean. Given this simplified view, we can construct a simple model for the atmosphere-ocean coupling with regard to carbon balance (Walker, 1991). The model must keep track of  $\text{CO}_2$  in the atmosphere, total carbon in the surface and deep ocean, and the speciation of inorganic carbon in the surface ocean because the exchange with the atmosphere is a diffusive process depending on the partial pressure of  $\text{CO}_2$  in the surface ocean.





Schematic of the ocean-atmosphere system for carbon

The equations are (see Walker 1991):

$$\begin{aligned}
 \frac{d(CO_2)_{ATM}}{dt} &= \frac{[(CO_2)_{SO} - (CO_2)_{ATM}]}{T} + \frac{input}{4.95 \cdot 10^{16}} \\
 \frac{dC_{SO}}{dt} &= \left\{ -\frac{[(CO_2)_{SO} - (CO_2)_{ATM}] 4.95 \cdot 10^{16}}{T} - 1.25P + (C_{DO} - C_{SO})w \right\} / V_{SO} \\
 \frac{dALK_{SO}}{dt} &= \left[ (ALK_{DO} - ALK_{SO})w - 0.35P \right] / V_{SO} \\
 \frac{dC_{DO}}{dt} &= \left[ 1.25P - (C_{DO} - C_{SO})w \right] / V_{DO} \\
 \frac{dALK_{DO}}{dt} &= \left[ - (ALK_{DO} - ALK_{SO})w + 0.35P \right] / V_{DO} \\
 (CO_2)_{SO} &= 0.054 \frac{(2C_{SO} - ALK_{SO})^2}{(ALK_{SO} - C_{SO})}
 \end{aligned}$$

where the subscripts refer to the atmosphere, the surface ocean, and the deep ocean.  $CO_2$  is mass of carbon dioxide (expressed in relative units, i.e., unity for present day),  $C$  is total carbon concentration,  $ALK$  is alkalinity. The constant  $4.95 \times 10^{16}$  in the equations is a conversion from relative  $CO_2$  to actual moles of  $CO_2$  in the atmosphere.  $T$  is a time constant,  $w$  is a flux (exchange) from surface to deep ocean,  $P$  is biological production of organic (and associated inorganic) carbon that "rains" into the deep ocean, and  $V$  represents the volume of the ocean compartments. Finally, *input* represents the rate of addition of carbon to the atmosphere by fossil-fuel burning. Appropriate values for the computation are (with initial conditions at 1850):  $T = 8.64$  y;  $V_{SO} = 0.12 \times 10^{18} \text{ m}^3$ ;  $V_{DO} = 1.23 \times 10^{18} \text{ m}^3$ ;  $P = 0.000175 \times 10^{18} \text{ mole y}^{-1}$ ;  $w =$

$0.001 \times 10^{18} \text{ m}^3$ ; and, at time "zero",  $(\text{CO}_2)_{\text{ATM}} = 1$  (relative units,  $4.95 \times 10^{16}$  mole actual units),  $C_{\text{SO}} = 2.01 \text{ mole m}^{-3}$ ,  $C_{\text{DO}} = 2.23 \text{ mole m}^{-3}$ ,  $\text{ALK}_{\text{SO}} = 2.2 \text{ mole m}^{-3}$ ,  $\text{ALK}_{\text{DO}} = 2.26 \text{ mole m}^{-3}$ . Assuming the following time course for *input* (in  $10^{14} \text{ mole y}^{-1}$ ), with approximately linear changes between specific times and with all inputs zero after 2500, write a code to compute the time course of atmospheric and oceanic carbon over (a) 1850-2600 and (b) 1850-6000.

Year	1850	1950	1980	2000	2050	2080	2100	2120	2150	2225	2300	2500
Input	0	1	4	5	8	10	10.5	10	8	3.5	2	0

#### 4.11. References

- Fetter, C.W. Jr., *Applied Hydrogeology*, 598 pp., Prentice-Hall, Upper Saddle River, NJ, 2001.
- Gerald C.F. and P.O. Wheatley, *Applied Numerical Analysis*, 319 pp., Addison Wesley, Reading, MA, 1999.
- Graf, W.H., *Hydraulics of Sediment Transport*, 513 pp., McGraw-Hill, New York, 1971.
- Greenwood, H.J., The synthesis and stability of anthophyllite. *J. Petrology*, 4: 317-351, 1963.
- Hemond, H.F. and E.J. Fechner, *Chemical fate and transport in the environment*, 433pp., Academic Press, San Diego, 1994.
- Henderson, F.M., *Open Channel Flow*, 522 pp., Macmillan, New York, 1970.
- Hornberger, G.M., Raffensperger, J.P., Wiberg, P.L., and K. Eshleman, *Elements of Physical Hydrology*, 302 pp., Johns Hopkins Press, Baltimore, 1998.
- Kot, M., *Elements of Mathematical Ecology*, 453 pp., Cambridge University Press, New York, 2001.
- Lasaga, A.C. and R.J. Kirkpatrick (eds.), *Kinetics of Geochemical Processes*, 398 pp., Mineralogical Society of America, Washington, DC, 1981.
- Nicolis, G. and I. Prigogine, *Self-organization in Non-equilibrium Systems: From Dissipative Structures to Order through Fluctuations*, 512 pp., Wiley, New York, 1977.
- Streeter, H.W. and E.B. Phelps, A study of the pollution and natural purification of the Ohio River, III, Factors concerned in the phenomena of oxidation and reaeration. U.S. Public Health Service, Pub. Health Bull. No. 146, 75 pp., 1925.
- Walker, J.C.G., *Numerical adventures with geochemical cycles*, 192 pp., Oxford University Press, New York, 1991.

### Box 4.1. The brusselator

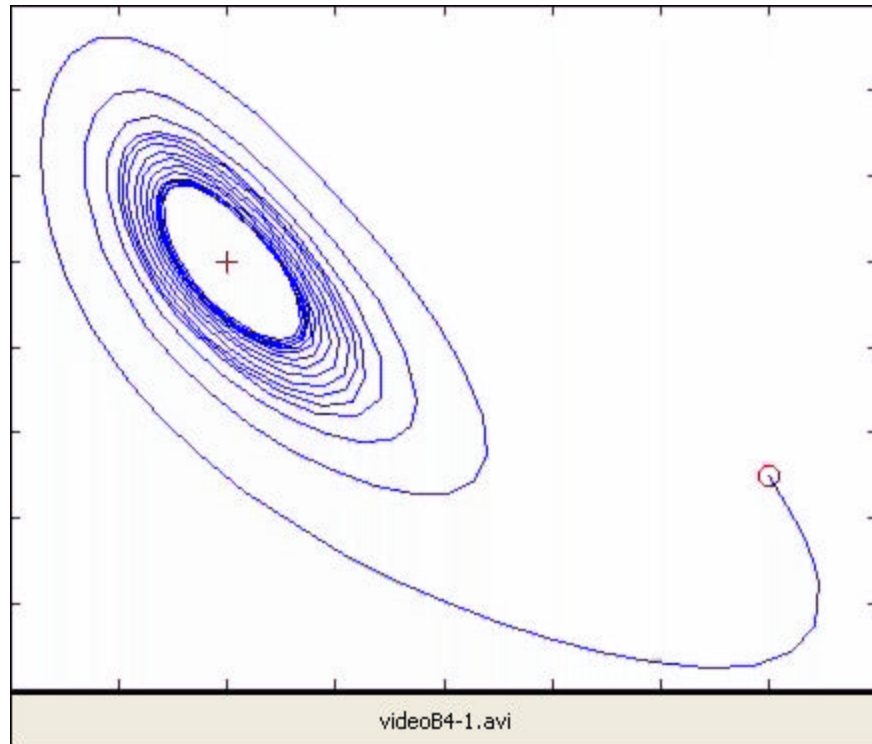
The following equations, defining what has been called the "brusselator" (Nicolis and Prigogine, 1977), serve as a model for certain self-catalyzing biochemical reactions.

$$\frac{dx}{dt} = A - Bx + x^2y - x$$
$$\frac{dy}{dt} = Bx - x^2y$$

The code for solving the equations and for plotting the "phase plane", i.e.  $x$  versus  $y$ , is shown below as are the results for  $A=1$  and  $B=2$  (Video B4.1). The "o" marks the initial condition and the "+" marks the equilibrium point where the time derivatives are zero. For these values of the parameters  $A$  and  $B$ , the solution is converging toward the equilibrium point. For larger values of  $B$  the solution to the equations is a limit cycle in which the values of  $x$  and  $y$  cycle around the unstable equilibrium point.

```
% brus_mov.m
% make a brusselator movie
% z0 is the initial condition
% par contains parameters A and B
%
z0=[2;1.5];par=[1 2];
[t, z]=ode45(@brusselator,tspan,z0,[],par);
A=par(1);B=par(2);
x=z(:,1);
y=z(:,2);
plot(x,y,'w');
hold on
plot(A,B/A,'+r','MarkerSize',8);
plot(z0(1),z0(2),'or','MarkerSize',8);
k=1;
M(k)=getframe;
for i=1:10:length(x)-10
    k=k+1;
    plot(x(i:i+10),y(i:i+10),'b','LineWidth',1.2)
    M(k)=getframe;
end
hold off

function ydot=brusselator(t,z,params)
%
% brusselator equations, parameters A and B
% ydot=brusselator(x,y,A,B)
%
A=params(1);B=params(2);
x=z(1,:);y=z(2,:);
dxdt=A+x.^2*y-B*x-x;
dydt=B*x-x.^2*y;
ydot=[dxdt; dydt];
```



**Video 4.1.** Brusselator results.

**Box 2.2. Manning equation**

The Manning equation is an equation commonly used to calculate the mean velocity  $U$  in a channel:

$$U = \frac{1}{n} R_H^{2/3} S^{1/2}$$

where  $n$  is the Manning roughness coefficient,  $R_H$  is hydraulic radius, and  $S$  is channel slope. Hydraulic radius is the ratio of the cross-sectional area,  $A$ , of flow in a channel to the length of the wetted perimeter,  $P$ . For a rectangular channel,  $R_H = wh/(w + 2h)$ ; if the channel is wide ( $w \gg h$ ),  $R_H \cong h$ . Values of  $n$  range from 0.025 for relatively smooth, straight streams to 0.075 for coarse bedded, overgrown channels. For steady, uniform flow, the channel bed slope  $S$  is equal to the water surface (friction) slope  $S_f$ . For non uniform flow, Manning's equation can still be used if  $S$  is replaced by  $S_f$ . The Manning equation can be combined with the discharge relationship  $Q=UA$  to give an expression for discharge

$$Q = \frac{1}{n} R_H^{2/3} S^{1/2} wh .$$

### Box 3.1. Errors in numerical methods

There are two principal sources of error in numerical computation. One is due to the fact that an *approximation* is being made (e.g., a derivative is being approximated by a finite difference). These errors are called *truncation errors*. The second is due to the fact that computers have limited precision, i.e., they can store only a finite number of decimal places. These errors are called *roundoff errors*.

Truncation errors arise when a function is approximated using a finite number of terms in an infinite series. For example, truncated Taylor series are the basis of finite difference approximations to derivatives (Chapter 3.2). The error in a finite difference approximation to a derivative is a direct result of the number of terms retained in the Taylor series (i.e., where the series is truncated). Truncation error is also present in other numerical approximations. In numerical integration, for example, when each increment of area under a curve is calculated using a polynomial approximation to the true function (Chapters 3.6–3.7), truncation errors arise that are related to the order of the approximating polynomial. For example an  $n^{\text{th}}$ -order polynomial approximation to a function results in an error in the integral over an increment  $\Delta x$  of  $O(\Delta x)^{n+2}$  (*local error*). When the integrals over each increment are summed to approximate the integral over some domain  $a \leq x \leq b$ , the local errors sum to give a *global error* of  $O(\Delta x)^{n+1}$ . Truncation errors decrease as step size ( $\Delta x$ ) is decreased – the finite difference approximation to a derivative is better (has lower truncation error) when  $\Delta x$  is "small" relative to when  $\Delta x$  is "large."

Roundoff errors stem from the fact that computers have a maximum number of digits that can be used to express a number. This means that the machine value given to fractional numbers without finite digit representations, for example,  $1/3 = 0.33333333\dots$ , will be rounded or chopped at the precision of the computer. It also means that there is a limit to how large or small a number a computer can represent in floating point form. For many computations, the small changes in values resulting from roundoff are insignificant. However, roundoff errors can become important. For example, subtraction of two nearly identical numbers (as occurs when computing finite differences with very small values of  $\Delta x$ ) can lead to relatively large roundoff error depending on the number of significant digits retained in the calculation. Interestingly, this means that approximations to derivatives will be improved by reducing  $\Delta x$  to some level because truncation errors are reduced, but that further decreases in  $\Delta x$  will make the estimate *worse* because roundoff error becomes large and dominates for very small values of  $\Delta x$ . Roundoff error can also complicate some logical operations that depend on establishing equality between two values if one or both are the result of computations that involved chopping or rounding.

Finally, in the hydrological sciences, measured data are often used in a calculation. For example, one might want to find the derivative of water velocity with respect to height above a streambed using numerical differentiation of data measured using a flowmeter. Such data are subject to *measurement errors*, which are then inserted into any numerical computation in which they are used.