Numerical Solution of Initial-Value Problems

An nth-order differential equation is accompanied by n auxiliary conditions, which are needed to determine the n constants of integration that arise in the solution process. When these conditions are provided at the same value of the independent variable, we speak of an initial-value problem (IVP). In other situations, the supplementary conditions are specified at different values of the independent variable. And since these values are usually stated at the extremities of the system, these types of problems are referred to as boundary-value problems (BVPs).

In this chapter, we will discuss various methods to numerically solve IVPs. Stability and stiffness of differential equations will also be covered. Treatment of BVPs is presented in Chapter 8. Numerical methods for a single, first-order IVP will be studied first; Figure 7.1. Some of these methods will then be extended and used to solve higher-order and systems of differential equations.

A single, first-order IVP is represented as

$$y' = f(x,y), \quad y(x_0) = y_0, \quad a = x_0 \le x \le x_n = b \tag{7.1}$$

where y_0 is the specified initial condition, the independent variable x assumes values in $[a, b]$, and it is assumed that a unique solution $y(x)$ exists in the interval $[a, b]$. The interval is divided into n segments of equal width h so that

$$x_1 = x_0 + h, \quad x_2 = x_0 + 2h, \ldots, \quad x_n = x_0 + nh$$

The solution at the point x_0 is available from the initial condition. The objective is to find estimates of the solution at the subsequent points $x_1, x_2, \ldots, x_n$.

7.1 One-Step Methods

One-step methods find the solution estimate y_{i+1} at the location x_{i+1} by extrapolating from the solution estimate y_i at the previous location x_i. Exactly, how this new estimate is extrapolated from the previous estimate depends on the numerical method used. Figure 7.2 describes a very simple one-step method, where the slope φ is used to extrapolate from y_i to the new estimate y_{i+1}

$$y_{i+1} = y_i + h\varphi \quad (i = 0, 1, 2, \ldots, n-1) \tag{7.2}$$

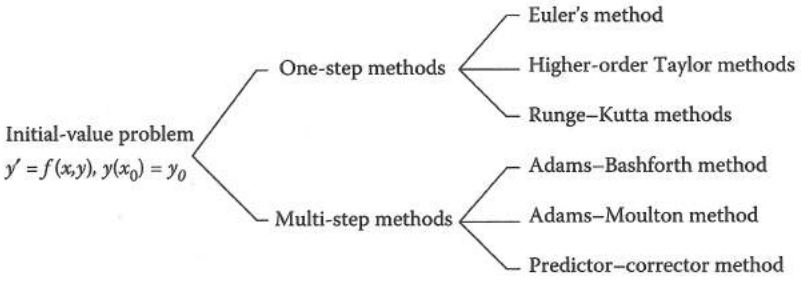

FIGURE 7.1
Classification of methods to solve an initial-value problem.

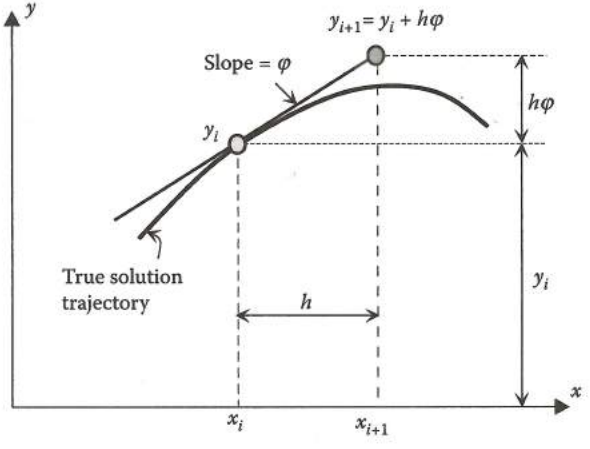

FIGURE 7.2
A simple one-step method.

Starting with the prescribed initial condition y_0, Equation 7.2 is applied in every subinterval $[x_i, x_{i+1}]$ to find solution estimates at $x_1, x_2, \ldots, x_n$. The general form in Equation 7.2 describes all one-step methods, with each method using a specific approach to estimate the slope φ. The simplest of all one-step methods is Euler's method, explained below.

7.2 Euler's Method

The expansion of $y(x_1)$ in a Taylor series about x_0 yields

$$y(x_1) = y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{1}{2!}h^2y''(x_0) + \cdots$$

Retaining the linear terms only, the above is rewritten as

$$y(x_1) = y(x_0) + hy'(x_0) + \frac{1}{2!}h^2y''(\xi_0)$$

for some ξ_0 between x_0 and x_1. In general, expanding $y(x_{i+1})$ about x_i yields

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{1}{2!}h^2y''(\xi_i)$$

for some ξ_i between x_i and x_{i+1}. Note that $y'(x_i) = f(x_i, y_i)$ by Equation 7.1. Introducing notations $y_i = y(x_i)$ and $y_{i+1} = y(x_{i+1})$, the estimated solution y_{i+1} can be found via

$$y_{i+1} = y_i + hf(x_i, y_i), \quad i = 0, 1, 2, \ldots, n-1 \tag{7.3}$$

known as Euler's method. Comparing with the description of the general one-step method, Equation 7.2, we see that the slope φ at x_i is simply estimated by $f(x_i, y_i)$, which is the first derivative at x_i, namely, $y'(x_i)$. Equation 7.3 is called the difference equation for Euler's method.

The user-defined function `EulerODE` uses Euler's method to estimate the solution of an IVP.

```
function y=EulerODE(f,x,y0)
%
% EulerODE uses Euler's method to solve a first-order
% ODE given in the form y'=f(x,y) subject to initial
% condition y0.
%
%  y=EulerODE(f,x,y0) where
%
%     f is an inline function representing f(x,y),
%     x is a vector representing the mesh points,
%     y0 is a scalar representing the initial value of y,
%
%     y is the vector of solution estimates at the mesh
%     points.

y=0*x;      % Pre-allocate
y(1)=y0; h=x(2)-x(1);
for n=1:length(x)-1,
   y(n+1)=y(n)+h*f(x(n),y(n));
end
```

Example 7.1: Euler's Method

Consider the IVP

$$y' + y = 2x, \quad y(0) = 1, \quad 0 \le x \le 1$$

The exact solution is derived as $y_{exact}(x) = 2x + 3e^{-x} - 2$. We will solve the IVP numerically using Euler's method with step size $h = 0.1$. Comparing with Equation 7.1, we find $f(x,y) = -y + 2x$. Starting with $y_0 = 1$, we use Equation 7.3 to find the estimate at the next point, $x = 0.1$, as

$$y_1 = y_0 + hf(x_0, y_0) = 1 + 0.1f(0,1) = 1 + 0.1(-1) = 0.9$$

The exact solution at $x = 0.1$ is calculated as

$$y_{exact}(0.1) = 2(0.1) + 3e^{-0.1} - 2 = 0.914512$$

Therefore, the relative error is 1.59%. Similar computations may be performed at the subsequent points 0.2, 0.3, ..., 1. The following MATLAB® script file uses the user-defined function `EulerODE` to find the numerical solution of the IVP and returns the results, including the exact values, in tabulated form. Figure 7.3 shows that Euler estimates capture the trend of the actual solution.

```
disp('   x          yEuler        yExact')
h = 0.1; x = 0:h:1; y0 = 1;
f = inline('-y+2*x','x','y');
yEuler = EulerODE(f,x,y0);
```

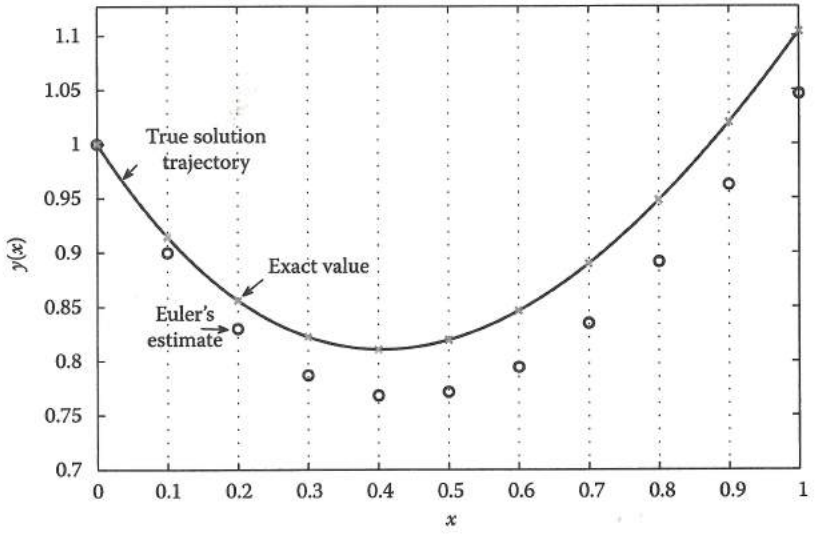

FIGURE 7.3
Comparison of Euler's and exact solutions in Example 7.1.

```
yExact = inline('2*x+3*exp(-x)-2');

for k = 1:length(x),
    x_coord = x(k);
    yE = yEuler(k);
    yEx = yExact(x(k));

   fprintf('%6.2f   %11.6f   %11.6f\n',x_coord,yE,yEx)

end
```

x	yEuler	yExact
0.00	1.000000	1.000000
0.10	0.900000	0.914512
0.20	0.830000	0.856192
0.30	0.787000	0.822455
0.40	0.768300	0.810960
0.50	0.771470	0.819592
0.60	0.794323	0.846435
0.70	0.834891	0.889756
0.80	0.891402	0.947987
0.90	0.962261	1.019709
1.00	1.046035	1.103638

The largest relative error is roughly 6.17% and occurs at $x = 0.7$. Using a smaller step size h will reduce the errors. Executing the above script file with $h = 0.05$, for example, shows a maximum relative error of 3.01% at $x = 0.65$.

7.2.1 Error Analysis for Euler's Method

Two sources of error are involved in the numerical solution of ordinary differential equations: round-off and truncation. Round-off errors are caused by the number of significant digits retained and used for calculations by the computer. Truncation errors are caused by the way a numerical method approximates the solution, and comprise two parts. The first part is a local truncation error resulting from the application of the numerical method in each step. The second part is a propagated truncation error caused by the approximations made in the previous steps. Adding the local and propagated truncation errors yields the global truncation error. It can be shown that the local truncation error is $O(h^2)$, while the global truncation error is $O(h)$.

7.2.2 Calculation of Local and Global Truncation Errors

The global truncation error at the point x_{i+1} is simply the difference between the actual solution y^a_{i+1} and the computed solution y^c_{i+1}. This contains the local truncation error, as well as the effects of all the errors accumulated in the steps prior to the current location x_{i+1}:

$$\text{Global truncation error at } x_{i+1} = \underbrace{\boxed{y^a_{i+1}}}_{\substack{\text{Actual}\\ \text{solution}\\ \text{at } x_{i+1}}} - \underbrace{\boxed{y^c_i + hf(x_i, y^c_i)}}_{\substack{\text{Euler's estimate at } x_{i+1}\\ \text{using computed solution at } x_i}} \tag{7.4}$$

The local truncation error at x_{i+1} is the difference between the actual solution y^a_{i+1} and the solution that would have been computed had the actual solution been used by Euler's method going from x_i to x_{i+1},

$$\text{Local truncation error at } x_{i+1} = \underbrace{\boxed{y^a_{i+1}}}_{\substack{\text{Actual solution} \\ \text{at } x_{i+1}}} - \underbrace{\boxed{y^a_i + hf(x_i, y^a_i)}}_{\substack{\text{Euler's estimate at } x_{i+1} \\ \text{using actual solution at } x_i}} \tag{7.5}$$

Example 7.2: Local and Global Truncation Errors

In Example 7.1, calculate the local and global truncation errors at each point and tabulate the results.

SOLUTION

Starting with the initial condition $y_0 = 1$, the Euler's computed value at $x_1 = 0.1$ is $y^c_1 = 0.9$ while the actual value is $y^a_1 = 0.914512$. At this stage, the global and local truncation errors are the same because Euler's method used the initial condition, which is exact, to find the estimate. At $x_2 = 0.2$, the computed value is $y^c_2 = 0.83$, which was calculated by Euler's method using the estimated value $y^c_1 = 0.9$ from the previous step. If instead of y^c_1 we use the actual value $y^a_1 = 0.914512$, the computed value at x_2 is

$$\tilde{y}_2 = y^a_1 + hf(x_1, y^a_1) = 0.914512 + 0.1f(0.1, 0.914512) = 0.843061$$

Therefore

$$\text{Local truncation error at } x_2 = y^a_2 - \tilde{y}_2$$

$$= 0.856192 - 0.843061 = 0.013131$$

The global truncation error at x_2 is simply calculated as

$$\text{Global truncation error at } x_2 = y^a_2 - y^c_2$$

$$= 0.856192 - 0.830000 = 0.026192$$

It is common to express these errors in the form of percent relative errors; hence, at each point we evaluate

$$\frac{\text{(Local or global) Truncation error}}{\text{Actual value}} \times 100$$

With this, the (local) percent relative error at x_2 is

$$\frac{y^a_2 - \tilde{y}_2}{y^a_2} \times 100 = \frac{0.013131}{0.856192} \times 100 = 1.53$$

The (global) percent relative error at x_2 is

$$\frac{y_2^a - y_2^c}{y_2^a} \times 100 = \frac{0.026192}{0.856192} \times 100 = 3.06$$

The following MATLAB script file uses this approach to find the percent relative errors at all x_i, and completes the table presented earlier in Example 7.1.

```
disp('   x        yEuler        yExact     e_local    e_global')
h=0.1; x=0:h:1; y0=1; f=inline('-y+2*x','x','y');
yEuler=EulerODE(f,x,y0); yExact=inline('2*x+3*exp(-x)-2');

ytilda=0*x; ytilda(1)=y0;
for n=1:length(x)-1,
    ytilda(n+1)=yExact(x(n))+h*f(x(n),yExact(x(n)));
end

for k=1:length(x),
    x_coord=x(k);
    yE=yEuler(k);
    yEx=yExact(x(k));
    e_local=(yEx-ytilda(k))/yEx*100;
    e_global=(yEx-yE)/yEx*100;

   fprintf('%6.2f %11.6f %11.6f %6.2f
   %6.2f\n',x_coord,yE,yEx,e_local,e_global)
end
```

x	yEuler	yExact	e_local	e_global	
0.00	1.000000	1.000000	0.00	0.00	
0.10	0.900000	0.914512	1.59	1.59	
0.20	0.830000	0.856192	1.53	3.06	Calculated by hand earlier
0.30	0.787000	0.822455	1.44	4.31	
0.40	0.768300	0.810960	1.33	5.26	
0.50	0.771470	0.819592	1.19	5.87	
0.60	0.794323	0.846435	1.04	6.16	
0.70	0.834891	0.889756	0.90	6.17	
0.80	0.891402	0.947987	0.76	5.97	
0.90	0.962261	1.019709	0.64	5.63	
1.00	1.046035	1.103638	0.53	5.22	

7.2.3 Higher-Order Taylor Methods

Euler's method was developed by retaining only the linear terms in a Taylor series. Retaining more terms in the series is the premise of higher-order Taylor methods. Expanding $y(x_{i+1})$ in a Taylor series about x_i yields

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{1}{2!}h^2y''(x_i) + \cdots + \frac{1}{k!}h^k y^{(k)}(x_i) + \frac{1}{(k+1)!}h^{k+1}y^{(k+1)}(\xi_i)$$

where ξ_i is between x_i and x_{i+1}. The kth-order Taylor method is defined as

$$y_{i+1} = y_i + hp_k(x_i, y_i), \quad i = 0, 1, 2, \ldots, n-1 \tag{7.6}$$

where

$$p_k(x_i, y_i) = f(x_i, y_i) + \frac{1}{2!}hf'(x_i, y_i) + \cdots + \frac{1}{k!}h^{k-1}f^{(k-1)}(x_i, y_i)$$

It is then clear that Euler's method is a first-order Taylor method. Recall that Euler's method has a local truncation error $O(h^2)$ and a global truncation error $O(h)$. The kth-order Taylor method has a local truncation error $O(h^{k+1})$ and a global truncation error $O(h^k)$. Therefore, the higher the order of the Taylor method, the more accurately it estimates the solution of the IVP. However, this reduction in error demands the calculation of the derivatives of $f(x,y)$, which is an obvious drawback.

Example 7.3: Second-Order Taylor Method

Solve the IVP in Examples 7.1 and 7.2 using the second-order Taylor method with the same step size $h = 0.1$ as before, and compare the numerical results with those produced by Euler's method.

SOLUTION

The problem is

$$y' + y = 2x, \quad y(0) = 1, \quad 0 \le x \le 1$$

so that $f(x,y) = -y + 2x$. Implicit differentiation with respect to x yields

$$f'(x, y) = -y' + 2 \overset{y'=f(x,y)}{=} -(-y + 2x) + 2 = y - 2x + 2$$

By Equation 7.6, the second-order Taylor method is defined as

$$y_{i+1} = y_i + hp_2(x_i, y_i), \quad i = 0, 1, 2, \ldots, n-1$$

where

$$p_2(x_i, y_i) = f(x_i, y_i) + \frac{1}{2!}hf'(x_i, y_i)$$

Therefore

$$y_{i+1} = y_i + h\left[f(x_i, y_i) + \frac{1}{2!}hf'(x_i, y_i)\right]$$

Starting with $y_0 = 1$, the solution estimate at the next location $x_1 = 0.1$ is calculated as

$$y_1 = y_0 + h\left[f(x_0, y_0) + \frac{1}{2}hf'(x_0, y_0)\right] = 0.9150$$

Noting the actual value at $x_1 = 0.1$ is 0.914512, this estimate has a (global) percent relative error of −0.05%, which is a significant improvement over the 1.59% offered by Euler's method at the same location. This upgrading of accuracy was expected because the second-order Taylor method has a (global) truncation error $O(h^2)$ compared with $O(h)$ for Euler's method. As mentioned before, this came at the expense of the evaluation of the first derivative $f'(x, y)$. The following MATLAB script tabulates the solution estimates generated by the second-order Taylor method:

```
disp('   x       yEuler     yTaylor2      e_Euler    e_Taylor2')
h=0.1; x=0:h:1; y0=1;
f=inline('-y+2*x','x','y'); fp=inline('y-2*x+2','x','y');
yEuler=EulerODE(f,x,y0); yExact=inline('2*x+3*exp(-x)-2');

yTaylor2=0*x; yTaylor2(1)=y0;
for n=1:length(x)-1,
   yTaylor2(n+1)=yTaylor2(n)+h*(f(x(n),yTaylor2(n))+(1/2)*h
   *fp(x(n),yTaylor2(n)));
end

for k=1:length(x),
    x_coord=x(k);
    yE=yEuler(k);
    yEx=yExact(x(k));
    yT=yTaylor2(k);
    e_Euler=(yEx-yE)/yEx*100;
    e_Taylor2=(yEx-yT)/yEx*100;
  fprintf('%6.2f  %11.6f  %11.6f  %6.2f
  %6.2f\n',x_coord,yE,yT,e_Euler,e_Taylor2)
end
```

x	yEuler	yTaylor2	e_Euler	e_Taylor2	
0.00	1.000000	1.000000	0.00	0.00	
0.10	0.900000	0.915000	1.59	−0.05	Calculated by hand earlier
0.20	0.830000	0.857075	3.06	−0.10	
0.30	0.787000	0.823653	4.31	−0.15	
0.40	0.768300	0.812406	5.26	−0.18	
0.50	0.771470	0.821227	5.87	−0.20	
0.60	0.794323	0.848211	6.16	−0.21	
0.70	0.834891	0.891631	6.17	−0.21	
0.80	0.891402	0.949926	5.97	−0.20	
0.90	0.962261	1.021683	5.63	−0.19	
1.00	1.046035	1.105623	5.22	−0.18	

7.3 Runge–Kutta Methods

In the last section, we learned that a kth-order Taylor method has a global truncation error $O(h^k)$ but requires the calculation of derivatives of $f(x,y)$. Runge–Kutta methods generate solution estimates with the accuracy of Taylor methods without having to calculate these derivatives. Recall from Equation 7.2 that all one-step methods to solve the IVP

$$y' = f(x,y), \quad y(x_0) = y_0, \quad a = x_0 \le x \le x_n = b$$

are expressed as

$$y_{i+1} = y_i + h\varphi(x_i, y_i)$$

where $\varphi(x_i, y_i)$ is an increment function and is essentially a suitable slope over the interval $[x_i, x_{i+1}]$ that is used for extrapolating y_{i+1} from y_i. The order of the Runge–Kutta method is the number of points that are used in $[x_i, x_{i+1}]$ to determine this suitable slope. For example, second-order Runge–Kutta methods use two points in each subinterval to find the representative slope, and so on.

7.3.1 Second-Order Runge–Kutta Methods

For the second-order Runge–Kutta (RK2) methods, the increment function is expressed as $\varphi(x_i, y_i) = a_1k_1 + a_2k_2$ so that

$$y_{i+1} = y_i + h(a_1k_1 + a_2k_2) \tag{7.7}$$

with

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + b_1h, y_i + c_{11}k_1h) \end{aligned} \tag{7.8}$$

where a_1, a_2, b_1, and c_{11} are constants, each set determined separately for each specific RK2 method. These constants are evaluated by setting Equation 7.7 equal to the first three terms in a Taylor series, neglecting terms with h^3 and higher:

$$y_{i+1} = y_i + hy'\big|_{x_i} + \frac{1}{2}h^2 y''\big|_{x_i} + O(h^3) \tag{7.9}$$

The term $y'|_{x_i}$ is simply $f(x_i, y_i)$, while

$$y''|_{x_i} = f'(x_i, y_i) \overset{\text{Chain rule}}{=} \left.\frac{\partial f}{\partial x}\right|_{(x_i, y_i)} + \left.\frac{\partial f}{\partial y}\right|_{(x_i, y_i)} \left.\frac{dy}{dx}\right|_{x_i}$$

$$\overset{y'=f(x,y)}{=} \left.\frac{\partial f}{\partial x}\right|_{(x_i, y_i)} + \left.\frac{\partial f}{\partial y}\right|_{(x_i, y_i)} f(x_i, y_i)$$

Substituting for $y'|_{x_i}$ and $y''|_{x_i}$ in Equation 7.9, we have

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{1}{2}h^2 \left.\frac{\partial f}{\partial x}\right|_{(x_i, y_i)} + \frac{1}{2}h^2 \left.\frac{\partial f}{\partial y}\right|_{(x_i, y_i)} f(x_i, y_i) + O(h^3) \tag{7.10}$$

Next, we will calculate y_{i+1} using a different approach as follows. In Equation 7.7, the term $k_2 = f(x_i + b_1 h, y_i + c_{11} k_1 h)$ is a function of two variables, and can be expanded about (x_i, y_i) as

$$k_2 = f(x_i + b_1 h, y_i + c_{11} k_1 h) = f(x_i, y_i) + b_1 h \left.\frac{\partial f}{\partial x}\right|_{(x_i, y_i)}$$

$$+ c_{11} k_1 h \left.\frac{\partial f}{\partial y}\right|_{(x_i, y_i)} + O(h^2) \tag{7.11}$$

Substituting Equation 7.11 and $k_1 = f(x_i, y_i)$ in Equation 7.7, we find

$$y_{i+1} = y_i + h\left\{a_1 f(x_i, y_i) + a_2\left[f(x_i, y_i) + b_1 h \left.\frac{\partial f}{\partial x}\right|_{(x_i, y_i)} + c_{11} k_1 h \left.\frac{\partial f}{\partial y}\right|_{(x_i, y_i)} + O(h^2)\right]\right\}$$

$$\overset{k_1 = f(x_i, y_i)}{=} y_i + (a_1 + a_2) hf(x_i, y_i) + a_2 b_1 h^2 \left.\frac{\partial f}{\partial x}\right|_{(x_i, y_i)} + a_2 c_{11} h^2 \left.\frac{\partial f}{\partial y}\right|_{(x_i, y_i)} f(x_i, y_i) + O(h^3) \tag{7.12}$$

The right-hand sides of Equations 7.10 and 7.12 represent the same quantity, y_{i+1}; hence, they must be equal. That yields

$$a_1 + a_2 = 1, \quad a_2 b_1 = \frac{1}{2}, \quad a_2 c_{11} = \frac{1}{2} \tag{7.13}$$

Since there are four unknowns and only three equations, a unique set of solutions does not exist. But if a value is assigned to one of the constants, the other three can be calculated. This is why there are several versions of RK2 methods, three of which are presented below. RK2 methods have local truncation error $O(h^3)$ and global truncation error $O(h^2)$, as did the second-order Taylor methods.

7.3.1.1 Improved Euler's Method

Assuming $a_2 = 1$, the other three constants in Equation 7.13 are determined as $a_1 = 0$, $b_1 = \frac{1}{2}$, and $c_{11} = \frac{1}{2}$. Inserting into Equations 7.7 and 7.8, the improved Euler's method is described by

$$y_{i+1} = y_i + hk_2 \tag{7.14}$$

where

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \end{aligned} \tag{7.15}$$

7.3.1.2 Heun's Method

Assuming $a_2 = \frac{1}{2}$, the other three constants in Equation 7.13 are determined as $a_1 = \frac{1}{2}$, $b_1 = 1$, and $c_{11} = 1$. Inserting into Equations 7.7 and 7.8, Heun's method is described by

$$y_{i+1} = y_i + \frac{1}{2}h(k_1 + k_2) \tag{7.16}$$

where

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h, y_i + k_1 h) \end{aligned} \tag{7.17}$$

7.3.1.3 Ralston's Method

Assuming $a_2 = \frac{2}{3}$, the other three constants in Equation 7.13 are determined as $a_1 = \frac{1}{3}$, $b_1 = \frac{3}{4}$, and $c_{11} = \frac{3}{4}$. Inserting into Equations 7.7 and 7.8, Ralston's method is described by

$$y_{i+1} = y_i + \frac{1}{3}h(k_1 + 2k_2) \tag{7.18}$$

where

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right) \tag{7.19}$$

Note that each of these RK2 methods produces estimates with the accuracy of a second-order Taylor method without calculating the derivative of $f(x,y)$. Instead, each method requires two function evaluations per step.

7.3.1.4 Graphical Representation of Heun's Method

Equations 7.16 and 7.17 can be combined as

$$y_{i+1} = y_i + h\frac{f(x_i, y_i) + f(x_i + h, y_i + k_1 h)}{2} \tag{7.20}$$

Since $k_1 = f(x_i, y_i)$, we have $y_i + k_1 h = y_i + hf(x_i, y_i)$. But this is the estimate y_{i+1} given by Euler's method at x_{i+1}, which we denote by y_{i+1}^{Euler} to avoid confusion with y_{i+1} in Equation 7.20. With this, and the fact that $x_i + h = x_{i+1}$, Equation 7.20 is rewritten as

$$y_{i+1} = y_i + h\frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{\text{Euler}})}{2} \tag{7.21}$$

The fraction multiplying h is the average of two quantities: the first one is the slope at the left end x_i of the interval; the second one is the estimated slope at the right end x_{i+1} of the interval. This is illustrated in Figure 7.4.

In Figure 7.4a, the slope at the left end of the interval is shown as $f(x_i, y_i)$. Figure 7.4b shows the estimated slope at the right end of the interval to be $f(x_{i+1}, y_{i+1}^{\text{Euler}})$. The line whose slope is the average of these two slopes, Figure 7.4c, yields an estimate that is superior to y_{i+1}^{Euler}. In Heun's method, y_{i+1} is extrapolated from y_i using this line.

The user-defined function `HeunODE` uses Heun's method to estimate the solution of an IVP.

```
function y=HeunODE(f,x,y0)
%
% HeunODE uses Heun's method to solve a first-order ODE
% given in the form y'=f(x,y) subject to initial
% condition y0.
%
%   y=HeunODE(f,x,y0) where
%
%     f is an inline function representing f(x,y),
%     x is a vector representing the mesh points,
%     y0 is a scalar representing the initial value of y,
%
%     y is the vector of solution estimates at the mesh
%     points.
y=0*x;   % Pre-allocate
y(1)=y0; h=x(2)-x(1);
for n=1:length(x)-1,
    k1=f(x(n),y(n));
    k2=f(x(n)+h,y(n)+h*k1);
    y(n+1)=y(n)+h*(k1+k2)/2;
end
```

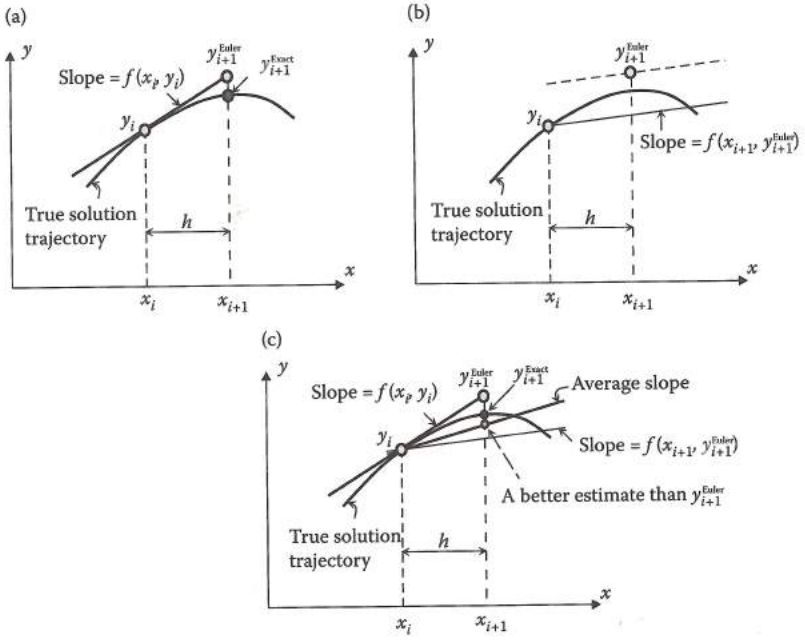

FIGURE 7.4
Graphical representation of Heun's method.

Example 7.4: RK2 Methods

Consider the IVP

$$y' - x^2y = 2x^2, \quad y(0) = 1, \quad 0 \le x \le 1, \quad h = 0.1$$

Compute the estimated value of $y_1 = y(0.1)$ using each of the three RK2 methods discussed earlier.

SOLUTION

Noting that $f(x, y) = x^2(2 + y)$, the value of $y_1 = y(0.1)$ estimated by each RK2 method is calculated as follows:

Improved Euler's Method

$$k_1 = f(x_0, y_0) = f(0,1) = 0$$
$$k_2 = f\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_1h\right) = f(0.05,1) = 0.0075 \Rightarrow y_1 = y_0 + hk_2$$
$$= 1 + 0.1(0.0075) = 1.0008$$

Heun's Method

$$k_1 = f(x_0, y_0) = f(0,1) = 0$$
$$k_2 = f(x_0 + h, y_0 + k_1h) = f(0.1,1) = 0.0300 \Rightarrow y_1 = y_0 + \frac{1}{2}h(k_1 + k_2)$$
$$= 1 + 0.05(0.0300) = 1.0015$$

Ralston's Method

$$k_1 = f(x_0, y_0) = f(0,1) = 0$$
$$k_2 = f\left(x_0 + \frac{3}{4}h, y_0 + \frac{3}{4}k_1h\right) = f(0.075,1) = 0.0169 \Rightarrow y_1 = y_0 + \frac{1}{3}h(k_1 + 2k_2)$$
$$= 1 + \frac{1}{3}(0.1)(0.0338) = 1.0011$$

Continuing this process, the estimates given by the three methods at the remaining points will be obtained and tabulated as in Table 7.1. The exact solution is $y = 3e^{x^3/3} - 2$. The global percent relative errors for all three methods are also listed, where it is readily observed that all RK2 methods perform better than Euler, and that Ralston's method is producing the most accurate estimates.

7.3.2 Third-Order Runge–Kutta Methods

For the third-order Runge–Kutta (RK3) methods, the increment function is expressed as $\varphi(x_i, y_i) = a_1k_1 + a_2k_2 + a_3k_3$, so that

$$y_{i+1} = y_i + h(a_1k_1 + a_2k_2 + a_3k_3) \tag{7.22}$$

TABLE 7.1

Summary of Calculations in Example 7.4

x	y_{Euler}	y_{Heun}	y_{Imp_Euler}	$y_{Ralston}$	e_{Euler}	e_{Heun}	e_{Imp_Euler}	$e_{Ralston}$
0.0	1.0000	1.0000	1.0000	1.0000	0.00	0.00	0.00	0.00
0.1	1.0000	1.0015	1.0008	1.0011	0.10	−0.05	0.02	−0.01
0.2	1.0030	1.0090	1.0075	1.0083	0.50	−0.10	0.05	−0.02
0.3	1.0150	1.0286	1.0263	1.0275	1.18	−0.15	0.08	−0.03
0.4	1.0421	1.0667	1.0636	1.0651	2.12	−0.19	0.10	−0.04
0.5	1.0908	1.1302	1.1261	1.1281	3.27	−0.23	0.14	−0.04
0.6	1.1681	1.2271	1.2219	1.2245	4.56	−0.25	0.17	−0.04
0.7	1.2821	1.3671	1.3604	1.3637	5.96	−0.27	0.22	−0.03
0.8	1.4430	1.5626	1.5541	1.5583	7.40	−0.28	0.27	−0.00
0.9	1.6633	1.8301	1.8191	1.8246	8.87	−0.27	0.34	0.04
1.0	1.9600	2.1922	2.1777	2.1849	10.37	−0.25	0.42	0.09

with

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + b_1 h, y_i + c_{11} k_1 h)$$
$$k_3 = f(x_i + b_2 h, y_i + c_{21} k_1 h + c_{22} k_2 h)$$

where a_1, a_2, a_3, b_1, b_2, c_{11}, c_{21}, and c_{22} are constants, each set determined separately for each specific RK3 method. These constants are found by setting Equation 7.22 equal to the first four terms in a Taylor series, neglecting terms with h^4 and higher. Proceeding as with RK2 methods, we will end up with six equations and eight unknowns. By assigning values to two of the constants, the other six can be determined. Because of this, there are several RK3 methods, two of which are presented here. RK3 methods have local truncation error $O(h^4)$ and global truncation error $O(h^3)$, as did the third-order Taylor methods.

7.3.2.1 Classical RK3 Method

The classical RK3 method is described by

$$y_{i+1} = y_i + \frac{1}{6} h(k_1 + 4k_2 + k_3) \tag{7.23}$$

where

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$
$$k_3 = f(x_i + h, y_i - k_1 h + 2k_2 h)$$

7.3.2.2 Heun's RK3 Method

Heun's RK3 method is described by

$$y_{i+1} = y_i + \frac{1}{4}h(k_1 + 3k_3) \tag{7.24}$$

where

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{1}{3}h, y_i + \frac{1}{3}k_1 h\right)$$
$$k_3 = f\left(x_i + \frac{2}{3}h, y_i + \frac{2}{3}k_2 h\right)$$

Each of these RK3 methods produces estimates with the accuracy of a third-order Taylor method without calculating the derivatives of $f(x,y)$. Instead, each method requires three function evaluations per step.

Example 7.5: RK3 Methods

Consider the IVP in Example 7.4:

$$y' - x^2 y = 2x^2, \quad y(0) = 1, \quad 0 \le x \le 1, \quad h = 0.1$$

Compute the estimated value of $y_1 = y(0.1)$ using the two RK3 methods presented above.

SOLUTION

Noting that $f(x, y) = x^2 (2 + y)$, the calculations are carried out as follows.

Classical RK3 Method

$$k_1 = f(x_0, y_0) = f(0,1) = 0$$
$$k_2 = f(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_1 h) = f(0.05,1) = 0.0075$$
$$k_3 = f(x_0 + h, y_0 - k_1 h + 2k_2 h) = f(0.1,1.0015) = 0.0300$$

$$y_1 = y_0 + \frac{1}{6}h(k_1 + 4k_2 + k_3)$$

$$= 1 + \frac{1}{6}(0.1)(4 \times 0.0075 + 0.0300) = 1.0010$$

TABLE 7.2

Summary of Calculations in Example 7.5

x	y_{Euler}	y_{RK3}	$y_{\text{Heun\_RK3}}$	e_{Euler}	e_{RK3}	$e_{\text{Heun\_RK3}}$
0.0	1.0000	1.0000	1.0000	0.00	0.0000	0.0000
0.1	1.0000	1.0010	1.0010	0.10	–0.0000	0.0000
0.2	1.0030	1.0080	1.0080	0.50	–0.0001	0.0001
0.3	1.0150	1.0271	1.0271	1.18	–0.0004	0.0003
0.4	1.0421	1.0647	1.0647	2.12	–0.0010	0.0007
0.5	1.0908	1.1277	1.1276	3.27	–0.0018	0.0014
0.6	1.1681	1.2240	1.2239	4.56	–0.0030	0.0024
0.7	1.2821	1.3634	1.3633	5.96	–0.0044	0.0038
0.8	1.4430	1.5584	1.5582	7.40	–0.0059	0.0059
0.9	1.6633	1.8253	1.8250	8.87	–0.0074	0.0086
1.0	1.9600	2.1870	2.1866	10.37	–0.0087	0.0124

Heun's RK3 Method

$$k_1 = f(x_0, y_0) = f(0,1) = 0$$

$$k_2 = f\left(x_0 + \frac{1}{3}h, y_0 + \frac{1}{3}k_1 h\right) = f(0.0333,1) = 0.0033$$

$$k_3 = f\left(x_0 + \frac{2}{3}h, y_0 + \frac{2}{3}k_2 h\right) = f(0.0667,1.0002) = 0.0133$$

$$y_1 = y_0 + \frac{1}{4}h(k_1 + 3k_3) = 1 + 0.05(0.0300) = 1.0010$$

A summary of all calculations is given in Table 7.2 where it is easily seen that the global percent relative errors for the two RK3 methods are considerably lower than all previous methods covered up to this point.

7.3.3 Fourth-Order Runge–Kutta Methods

For the fourth-order Runge–Kutta (RK4) methods, the increment function is expressed as

$$\varphi(x_i, y_i) = a_1 k_1 + a_2 k_2 + a_3 k_3 + a_4 k_4$$

so that

$$y_{i+1} = y_i + h(a_1 k_1 + a_2 k_2 + a_3 k_3 + a_4 k_4) \tag{7.25}$$

with

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + b_1 h, y_i + c_{11} k_1 h)$$
$$k_3 = f(x_i + b_2 h, y_i + c_{21} k_1 h + c_{22} k_2 h)$$
$$k_4 = f(x_i + b_3 h, y_i + c_{31} k_1 h + c_{32} k_2 h + c_{33} k_3 h)$$

where a_i, b_i, and c_{ij} are constants, each set determined separately for each specific RK4 method. These constants are found by setting Equation 7.25 equal to the first five terms in a Taylor series, neglecting terms with h^5 and higher. Proceeding as before leads to 10 equations and 13 unknowns. By assigning values to three of the constants, the other 10 can be determined. This is why there are many RK4 methods, but only the classical RK4 is presented here. RK4s have local truncation error $O(h^5)$ and global truncation error $O(h^4)$.

7.3.3.1 Classical RK4 Method

The classical RK4 method is described by

$$y_{i+1} = y_i + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{7.26}$$

where

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \\ k_4 &= f(x_i + h, y_i + k_3 h) \end{aligned} \tag{7.27}$$

RK4 methods produce estimates with the accuracy of a fourth-order Taylor method without calculating the derivatives of $f(x,y)$. Instead, four function evaluations per step are performed. The classical RK4 method is the most commonly used technique for numerical solution of first-order IVPs, as it offers the most acceptable balance of accuracy and computational effort.

The user-defined function `RK4` uses the classical RK4 method to estimate the solution of an IVP.

```
function y=RK4(f,x,y0)
%
% RK4 uses the classical RK4 method to solve a first-
% order ODE given in the form y'=f(x,y) subject to
% initial condition y0.
%
%   y=RK4(f,x,y0) where
%
%   f is an inline function representing f(x,y),
```

```
%     x is a vector representing the mesh points,
%     y0 is a scalar representing the initial value of y,
%
%     y is the vector of solution estimates at the mesh
%     points.

y=0*x;      % Pre-allocate
y(1)=y0; h=x(2)-x(1);
for n=1:length(x)-1,
    k1=f(x(n),y(n));
    k2=f(x(n)+h/2,y(n)+h*k1/2);
    k3=f(x(n)+h/2,y(n)+h*k2/2);
    k4=f(x(n)+h,y(n)+h*k3);
  y(n+1)=y(n)+h*(k1+2*k2+2*k3+k4)/6;
end
```

Example 7.6: RK4 methods

Consider the IVP in Examples 7.4 and 7.5:

$$y' - x^2 y = 2x^2, \quad y(0) = 1, \quad 0 \le x \le 1, \quad h = 0.1$$

Compute the estimated value of $y_1 = y(0.1)$ using the classical RK4 method.

SOLUTION

Noting that $f(x, y) = x^2 (2 + y)$, the calculations are carried out as follows.

Classical RK4 Method

$$k_1 = f(x_0, y_0) = f(0,1) = 0$$

$$k_2 = f\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_1 h\right) = f(0.05,1) = 0.0075$$

$$k_3 = f\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_2 h\right) = f(0.05,1.0004) = 0.0075$$

$$k_4 = f(x_0 + h, y_0 + k_3 h) = f(0.1,1.0008) = 0.0300$$

$$y_1 = y_0 + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) = 1.0010$$

A summary of all calculations is provided in Table 7.3 where it is easily seen that the global percent relative error for the classical RK4 method is significantly lower than all previous methods covered up to this point. As expected, starting with Euler's method, which is indeed a first-order Runge–Kutta method, the accuracy improves with the order of the RK method.

TABLE 7.3

Summary of Calculations in Example 7.6

		RK4	RK3	RK2	RK1
x	y_{RK4}	e_{RK4}	e_{RK3}	e_{Heun}	e_{Euler}
0.0	1.000000	0.000000	0.0000	0.00	0.00
0.1	1.001000	0.000001	−0.0000	−0.05	0.10
0.2	1.008011	0.000002	−0.0001	−0.10	0.50
0.3	1.027122	0.000003	−0.0004	−0.15	1.18
0.4	1.064688	0.000004	−0.0010	−0.19	2.12
0.5	1.127641	0.000005	−0.0018	−0.23	3.27
0.6	1.223966	0.000006	−0.0030	−0.25	4.56
0.7	1.363377	0.000007	−0.0044	−0.27	5.96
0.8	1.558286	0.000010	−0.0059	−0.28	7.40
0.9	1.825206	0.000016	−0.0074	−0.27	8.87
1.0	2.186837	0.000028	−0.0087	−0.25	10.37

7.3.3.2 Higher-Order Runge–Kutta Methods

The classical RK4 is the most commonly used numerical method for solving first-order IVPs. If higher levels of accuracy are desired, the recommended technique is Butcher's fifth-order Runge–Kutta method (RK5), which is defined as

$$y_{i+1} = y_i + \frac{1}{90}h(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6) \tag{7.28}$$

where

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{8}k_1 h + \frac{1}{8}k_2 h\right)$$

$$k_4 = f\left(x_i + \frac{1}{2}h, y_i - \frac{1}{2}k_2 h + k_3 h\right)$$

$$k_5 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 h + \frac{9}{16}k_4 h\right)$$

$$k_6 = f\left(x_i + h, y_i - \frac{3}{7}k_1 h + \frac{2}{7}k_2 h + \frac{12}{7}k_3 h - \frac{12}{7}k_4 h + \frac{8}{7}k_5 h\right)$$

Therefore, Butcher's RK5 method requires six function evaluations per step.

7.3.4 Runge–Kutta–Fehlberg Method

One way to estimate the local truncation error for Runge–Kutta methods is to use two RK methods of different order and subtract the results. For cases involving variable step size, the error estimate can be used to decide when the step size needs to be adjusted. Naturally, a drawback of this approach is the number of function evaluations required per step. For example, we consider a common approach that uses a fourth-order and a fifth-order RK. This requires a total of 10 (four for RK4 and six for RK5) function evaluations per step. To get around the computational burden, the Runge–Kutta–Fehlberg (RKF) method utilizes an RK5 method that uses the function evaluations provided by its accompanying RK4 method.* This will reduce the number of function evaluations per step from 10 to 6.

$$y_{i+1} = y_i + h\left(\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right) \tag{7.29}$$

together with a fifth-order method

$$y_{i+1} = y_i + h\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right) \tag{7.30}$$

where

$$\begin{aligned}
k_1 &= 2f(x_i, y_i)\\
k_2 &= f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1h\right)\\
k_3 &= f\left(x_i + \frac{3}{8}h, y_i + \frac{3}{32}k_1h + \frac{9}{32}k_2h\right)\\
k_4 &= f\left(x_i + \frac{12}{13}h, y_i + \frac{1932}{2197}k_1h - \frac{7200}{2197}k_2h + \frac{7296}{2197}k_3h\right)\\
k_5 &= f\left(x_i + h, y_i + \frac{439}{216}k_1h - 8k_2h + \frac{3680}{513}k_3h - \frac{845}{4104}k_4h\right)\\
k_6 &= f\left(x_i + \frac{1}{2}h, y_i - \frac{8}{27}k_1h + 2k_2h - \frac{3544}{2565}k_3h + \frac{1859}{4104}k_4h - \frac{11}{40}k_5h\right)
\end{aligned}$$

* Refer to K.E. Atkinson, *An Introduction to Numerical Analysis*. 2nd edition, John Wiley, New York, 1989.

Subtracting Equation 7.29 from Equation 7.30 yields the estimate of the local truncation error:

$$\text{Error} = h\left(\frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6\right) \tag{7.31}$$

In each step, Equation 7.29 gives the fourth-order accurate estimate, Equation 7.30 gives the fifth-order accurate estimate, and Equation 7.31 returns the estimated local truncation error.

7.4 Multistep Methods

In single-step methods, the solution estimate y_{i+1} at the point x_{i+1} is obtained by using information at a single previous point x_i. Multistep methods are based on the idea that a more accurate estimate for y_{i+1} at x_{i+1} can be attained by utilizing information on two or more previous points rather than x_i only. Consider $y' = f(x, y)$ subject to initial condition $y(x_0) = y_0$. To use a multistep method to find an estimate for y_1, information on at least two previous points are needed. However, the only available information is y_0. This means that such methods cannot self-start and the estimates at the first few points—depending on the order of the method—must be found using either a single-step method such as the classical RK4 or another multistep method that uses fewer previous points.

Multistep methods can be explicit or implicit. Explicit methods employ an explicit formula to calculate the estimate. For example, if an explicit method uses two previous points, the estimate y_{i+1} at x_{i+1} is in the form

$$y_{i+1} = F(x_{i+1}, x_i, y_i, x_{i-1}, y_{i-1})$$

This way, only known values appear on the right-hand side. In implicit methods, the unknown estimate y_{i+1} is involved on both sides of the equation

$$y_{i+1} = \tilde{F}(x_{i+1}, y_{i+1}, x_i, y_i, x_{i-1}, y_{i-1})$$

and must be determined iteratively using the methods described in Chapter 3.

7.4.1 Adams–Bashforth Method

Adams–Bashforth method is an explicit multistep method to estimate the solution y_{i+1} of an IVP at x_{i+1} by using the solution estimates at two or more previous points. Several formulas can be derived depending on the number of previous points used. The order of each formula is the number of previous

points it uses. For example, a second-order formula finds y_{i+1} by utilizing the estimates y_i and y_{i-1} at the two prior points x_i and x_{i-1}.

To derive the Adams–Bashforth formulas, we integrate $y' = f(x, y)$ over an arbitrary interval $[x_i, x_{i+1}]$

$$\int_{x_i}^{x_{i+1}} y'\,dx = \int_{x_i}^{x_{i+1}} f(x,y)\,dx$$

Because $\int_{x_i}^{x_{i+1}} y'\,dx = y(x_{i+1}) - y(x_i)$, the above can be rewritten as

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} f(x,y)\,dx$$

or

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} f(x,y)\,dx \tag{7.32}$$

But since $y(x)$ is unknown, $f(x,y)$ cannot be integrated. To remedy this, $f(x,y)$ is approximated by a polynomial that interpolates the data at (x_i, y_i) and a few previous points. The number of the previous points that end up being used depends on the order of the formula to be derived. For example, for a second-order Adams–Bashforth formula, we use the polynomial that interpolates the data at (x_i, y_i) and one previous point, (x_{i-1}, y_{i-1}), and so on.

7.4.1.1 Second-Order Adams–Bashforth Formula

The polynomial that interpolates the data at (x_i, y_i) and (x_{i-1}, y_{i-1}) is linear and in the form

$$p_1(x) = f(x_i, y_i) + \frac{f(x_i, y_i) - f(x_{i-1}, y_{i-1})}{x_i - x_{i-1}}(x - x_i)$$

Letting $f_i = f(x_i, y_i)$ and $f_{i-1} = f(x_{i-1}, y_{i-1})$ for brevity, using $p_1(x)$ in Equation 7.32

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} p_1(x)\,dx$$

and assuming equally spaced data with spacing h, we arrive at

$$y_{i+1} = y_i + \frac{1}{2}h(3f_i - f_{i-1}) \tag{7.33}$$

As mentioned earlier, this formula cannot self-start because finding y_1 requires y_0 and y_{-1}, the latter not known. First, a single-step method such as RK4 is used to find y_1 from the initial condition y_0. The first application of Equation 7.33 is when $i = 1$ so that y_2 can be obtained using the information on y_0 and y_1.

7.4.1.2 Third-Order Adams–Bashforth Formula

Approximating the integrand $f(x,y)$ in Equation 7.32 by the second-degree polynomial $p_2(x)$ (Section 5.5) that interpolates the data at (x_i, y_i), (x_{i-1}, y_{i-1}), and (x_{i-2}, y_{i-2}), and carrying out the integration yields

$$y_{i+1} = y_i + \frac{1}{12}h(23f_i - 16f_{i-1} + 5f_{i-2}) \tag{7.34}$$

Since only y_0 is known, we first apply a method such as RK4 to find y_1 and y_2. The first application of Equation 7.34 is when $i = 2$ to obtain y_3 by using the information on y_0, y_1, and y_2.

7.4.1.3 Fourth-Order Adams–Bashforth Formula

Approximating the integrand $f(x,y)$ in Equation 7.32 by the third-degree polynomial $p_3(x)$ (Section 5.5) that interpolates the data at (x_i, y_i), (x_{i-1}, y_{i-1}), (x_{i-2}, y_{i-2}), and (x_{i-3}, y_{i-3}), and carrying out the integration yields

$$y_{i+1} = y_i + \frac{1}{24}h(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}) \tag{7.35}$$

Since only y_0 is known, we first apply a method such as RK4 to find y_1, y_2, and y_3. The first application of Equation 7.35 is when $i = 3$ to obtain y_4 by using the information on y_0, y_1, y_2, and y_3.

Adams–Bashforth formulas are primarily used in conjunction with the Adams–Moulton formulas, which are also multistep but implicit, to be presented next. A weakness of higher-order Adams–Bashforth formulas is that stability requirements place limitations on the step size that is necessary for desired accuracy. The user-defined function `AdamsBashforth4` uses the fourth-order Adams–Bashforth formula to estimate the solution of an IVP.

```
function y=AdamsBashforth4(f,x,y0)
%
% AdamsBashforth4 uses the fourth-order Adams-Bashforth
% formula to solve a first-order ODE in the form y'=f(x,y)
% subject to initial condition y0.
%
%    y=AdamsBashforth4(f,x,y0) where
%
%      f is an inline function representing f(x,y),
%      x is a vector representing the mesh points,
%      y0 is a scalar representing the initial value of y,
%
%      y is the vector of solution estimates at the mesh
%      points.

y(1:4)=RK4(f,x(1:4),y0);
for n=4:length(x)-1,
    h=x(n+1)-x(n);
    y(n+1)=y(n)+h*(55*f(x(n),y(n))-59*f(x(n-1),y(n-1))
    +37*f(x(n-2),y(n-2))-9*f(x(n-3),y(n-3)))/24;
end
```

7.4.2 Adams–Moulton Method

Adams–Moulton method is an implicit multistep method to estimate the solution y_{i+1} of an IVP at x_{i+1} by using the solution estimates at two or more previous points, as well as (x_{i+1}, y_{i+1}), where the solution is to be determined. Several formulas can be derived depending on the number of points used. The order of each formula is the total number of points it uses. For example, a second-order formula finds y_{i+1} by utilizing the estimates y_i and y_{i+1} at the points x_i and x_{i+1}. This makes the formula implicit because the unknown y_{i+1} will appear on both sides of the ensuing equation.

Derivation of Adams–Moulton formulas is similar to Adams–Bashforth where the integrand in Equation 7.32 is approximated by a polynomial that interpolates the data at prior points, as well as the point where the solution is being determined.

7.4.2.1 Second-Order Adams–Moulton Formula

The polynomial that interpolates the data at (x_i, y_i) and (x_{i+1}, y_{i+1}) is linear and in the form

$$p_1(x) = f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}(x - x_i)$$

where $f_i = f(x_i, y_i)$ and $f_{i+1} = f(x_{i+1}, y_{i+1})$. Replacing $f(x,y)$ in Equation 7.32 with $p_1(x)$ and carrying out the integration yields

$$y_{i+1} = y_i + \frac{1}{2}h(f_i + f_{i+1}) \tag{7.36}$$

This formula is implicit because $f_{i+1} = f(x_{i+1}, y_{i+1})$ contains y_{i+1}, which is the solution being sought. In this type of a situation, y_{i+1} must be found iteratively using the techniques listed in Chapter 3. This formula has a global truncation error $O(h^2)$.

7.4.2.2 Third-Order Adams–Moulton Formula

Approximating the integrand $f(x,y)$ in Equation 7.32 by the second-degree polynomial that interpolates the data at (x_{i+1}, y_{i+1}), (x_i, y_i), and (x_{i-1}, y_{i-1}), and carrying out the integration yields

$$y_{i+1} = y_i + \frac{1}{12}h(5f_{i+1} + 8f_i - f_{i-1}) \tag{7.37}$$

Since only y_0 is initially known, a method such as RK4 is first applied to find y_1. The first application of Equation 7.37 is when $i = 1$ to obtain y_2 implicitly. This formula has a global truncation error $O(h^3)$.

7.4.2.3 Fourth-Order Adams–Moulton Formula

Approximating the integrand $f(x,y)$ in Equation 7.32 by the third-degree polynomial that interpolates the data at (x_{i+1}, y_{i+1}), (x_i, y_i), (x_{i-1}, y_{i-1}), and (x_{i-2}, y_{i-2}), and carrying out the integration we find

$$y_{i+1} = y_i + \frac{1}{24}h(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}) \tag{7.38}$$

Since only y_0 is initially known, a method such as RK4 is first applied to find y_1 and y_2. The first application of Equation 7.38 is when $i = 2$ to obtain y_3 implicitly. This formula has a global truncation error $O(h^4)$.

7.4.3 Predictor–Corrector Methods

Predictor–corrector methods are a class of techniques that employ a combination of an explicit formula and an implicit formula to solve an IVP. First, the explicit formula is used to predict the value of y_{i+1}. This predicted value is denoted by $\tilde{y}_{i+1}$. The predicted $\tilde{y}_{i+1}$ is then used on the right-hand side of

an implicit formula to obtain a new, more accurate value for y_{i+1} on the left-hand side.

The simplest predictor–corrector method is Heun's method, presented in Section 7.3. Heun's method first uses Euler's method—an explicit formula—as the predictor to obtain y_{i+1}^{Euler}. This predicted value is then used in Equation 7.21, which is the corrector

$$y_{i+1} = y_i + h\frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{\text{Euler}})}{2}$$

to find a more accurate value for y_{i+1}. A modified version of this approach is derived next so that a desired accuracy may be achieved through repeated applications of the corrector formula.

7.4.3.1 Heun's Predictor–Corrector Method

The objective is to find an estimate for y_{i+1}. The method is implemented as follows:

1. Find a first estimate for y_{i+1}, denoted by $y_{i+1}^{(1)}$, using Euler's method, which is an explicit formula

   Predictor $$y_{i+1}^{(1)} = y_i + hf(x_i, y_i) \tag{7.39}$$

2. Improve the predicted estimate by solving Equation 7.21 iteratively

   Corrector $$y_{i+1}^{(k+1)} = y_i + h\frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{(k)})}{2}, \quad k = 1, 2, 3, \ldots \tag{7.40}$$

   Therefore, $y_{i+1}^{(1)}$ is used in Equation 7.40 to obtain $y_{i+1}^{(2)}$, and so on.

3. The iterations in Step 2 are terminated when the following criterion is satisfied:

   Tolerance $$\left|\frac{y_{i+1}^{(k+1)} - y_{i+1}^{(k)}}{y_{i+1}^{(k+1)}}\right| < \varepsilon \tag{7.41}$$

   where ε is a prescribed tolerance.

4. If the tolerance criterion is met, increment i by 1 and set y_i equal to this last $y_{i+1}^{(k+1)}$ and go to Step 1.

7.4.3.2 Adams–Bashforth–Moulton Predictor–Corrector Method

Several predictor–corrector formulas can be created by combining one of the (explicit) Adams–Bashforth formulas of a particular order as the predictor with the (implicit) Adams–Moulton formula of the same order as the corrector. The fourth-order formulas of these two methods, for example, can be combined to create the fourth-order Adams–Bashforth–Moulton (ABM4) predictor–corrector:

Predictor $$y_{i+1}^{(1)} = y_i + \frac{1}{24}h(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}), \quad i = 3, 4, \ldots, n \tag{7.42}$$

Corrector $$y_{i+1}^{(k+1)} = y_i + \frac{1}{24}h(9f_{i+1}^{(k)} + 19f_i - 5f_{i-1} + f_{i-2}), \quad k = 1, 2, 3, \ldots \tag{7.43}$$

where $f_{i+1}^{(k)} = f(x_{i+1}, y_{i+1}^{(k)})$. This method cannot self-start and is implemented as follows: starting with the initial condition y_0, apply a method such as RK4 to find estimates for y_1, y_2, and y_3 and calculate their respective $f(x, y)$ values. At this stage, the predictor (Equation 7.42) is applied to find $y_4^{(1)}$, which is then used to calculate $f_4^{(1)}$. The corrector (Equation 7.43) is next applied to obtain $y_4^{(2)}$. The estimate can be substituted back into Equation 7.43 for iterative correction. The process is repeated for the remaining values of the index i.

The user-defined function `ABM4PredCorr` uses the fourth-order ABM predictor–corrector method to estimate the solution of an IVP. Note that the function does not perform the iterative correction mentioned above.

```
function y=ABM4PredCorr(f,x,y0)
%
% ABM4PredCorr uses the fourth-order Adams-Bashforth-
% Moulton predictor-corrector formula to solve y'=f(x,y)
% subject to initial condition y0.
%
%   y=ABM4PredCorr(f,x,y0) where
%
%     f is an inline function representing f(x,y),
%     x is a vector representing the mesh points,
%     y0 is a scalar representing the initial value of y,
%
%     y is the vector of solution estimates at the mesh
%     points.
%
py=zeros(4,1);    % Pre-allocate
```

```
y(1:4)=RK4(f,x(1:4),y0); % Find the first 4 elements by RK4
h=x(2)-x(1);

% Start ABM4
for n=4:length(x)-1,
    py(n+1)=y(n)+(h/24)*(55*f(x(n),y(n))-59*f(x(n-1),
    y(n-1))+37*f(x(n-2),y(n-2))-9*f(x(n-3),y(n-3)));
    y(n+1)=y(n)+(h/24)*(9*f(x(n+1),py(n+1))+19*f(x(n),
    y(n))-5*f(x(n-1),y(n-1))+f(x(n-2),y(n-2)));
end
```

Example 7.7: ABM4 Predictor–Corrector Method

Consider the IVP in Examples 7.4 through 7.6:

$$y' - x^2 y = 2x^2, \quad y(0) = 1, \quad 0 \le x \le 1, \quad h = 0.1$$

Compute the estimated value of $y_4 = y(0.4)$ using the ABM4 predictor–corrector method.

SOLUTION

$f(x, y) = x^2(2 + y)$. The first element y_0 is given by the initial condition. The next three are obtained by RK4 as

$$y_1 = 1.001000, \quad y_2 = 1.008011, \quad y_3 = 1.027122$$

The respective $f(x, y)$ values are calculated next:

$$f_0 = f(x_0, y_0) = f(0,1) = 0$$

$$f_1 = f(x_1, y_1) = f(0.1,1.001000) = 0.030010$$

$$f_2 = f(x_2, y_2) = f(0.2,1.008011) = 0.120320$$

$$f_3 = f(x_3, y_3) = f(0.3,1.027122) = 0.272441$$

Prediction

Equation 7.42 yields

$$y_4^{(1)} = y_3 + \frac{1}{24}h(55f_3 - 59f_2 + 37f_1 - 9f_0) = 1.064604$$

Calculate $f_4^{(1)} = f(x_4, y_4^{(1)}) = f(0.4,1.064604) = 0.490337$.

Correction

Equation 7.43 yields

$$y_4^{(2)} = y_3 + \frac{1}{24}h(9f_4^{(1)} + 19f_3 - 5f_2 + f_1)$$

$$= 1.064696 \quad \text{Rel error} = 0.0008\%$$

This corrected value may be improved by substituting $y_4^{(2)}$ and the corresponding $f_4^{(2)} = f(x_4, y_4^{(2)})$ into Equation 7.43

$$y_4^{(3)} = y_3 + \frac{1}{24}h(9f_4^{(2)} + 19f_3 - 5f_2 + f_1)$$

and inspecting the accuracy. In the present analysis, we perform only one correction so that $y_4^{(2)}$ is regarded as the value that will be used for y_4. This estimate is then used in Equation 7.42 with the index i incremented from 3 to 4. Continuing this process, we generate the numerical results in Table 7.4.

Another well-known predictor–corrector method is the fourth-order Milne's method:

$$\text{Predictor} \quad y_{i+1}^{(1)} = y_{i-3} + \frac{4}{3}h(2f_i - f_{i-1} + 2f_{i-2}), \quad i = 3, 4, \ldots, n$$

$$\text{Corrector} \quad y_{i+1}^{(k+1)} = y_{i-1} + \frac{1}{3}h(f_{i+1}^{(k)} + 4f_i + f_{i-1}), \quad k = 1, 2, 3, \ldots$$

where $f_{i+1}^{(k)} = f(x_{i+1}, y_{i+1}^{(k)})$. As with the fourth-order ABM, this method cannot self-start and needs a method such as RK4 for estimating y_1, y_2, and y_3 first.

TABLE 7.4

Summary of Calculations in Example 7.7

x	y_{RK4}	$\tilde{y}_i$ Predicted	y_i Corrected
0.0	1.000000		
0.1	1.001000		
0.2	1.008011		
0.3	1.027122	Start ABM4	
0.4		1.064604	1.064696
0.5		1.127517	1.127662
0.6		1.223795	1.224004
0.7		1.363143	1.363439
0.8		1.557958	1.558381
0.9		1.824733	1.825350
1.0		2.186134	2.187052