CHAPTER 12

Ordinary Differential Equations: Fundamentals

The numerical differentiation formulas presented in the previous chapter are used extensively in the numerical solution of ordinary and partial differential equations; techniques for solving first-order ordinary differential equations (ODE) are the subject of this chapter. We assume that the differential equation is written in the form $y' = f(x, y)$ with the value of the function $y(x)$ given at x_0; i.e., $y(x_0) = y_0$. The basic idea is to divide the interval of interest into discrete steps (of fixed width h) and find approximations to the function y at those values of x. In other words, we find solutions at $x_1, x_2, x_3, ..., x_n$.

The first methods we consider are based on the Taylor polynomial representation of the unknown function $y(x)$. The simplest method, Euler's, retains only the first-derivative term in the Taylor expansion. In order to use higher-order Taylor polynomials, it is necessary to find higher derivatives of the function $f(x, y)$ that defines the slope of the unknown function $y(x)$. The Runge-Kutta method achieves a more accurate solution than Euler's method does, without computing higher derivatives of f. Each step of a Runge-Kutta method involves evaluating $f(x, y)$ at several different values of x and y and combining the results to form the approximation to y at the next x.

The third group of techniques that we study are the "multistep methods," which include both explicit and implicit forms. The term "multistep" refers to the fact that these methods make use of the computed value of the solution at several previous points. Implicit methods have superior stability characteristics but are more difficult to solve than the explicit methods. An implicit and an explicit method are often combined to form a predictor-corrector formula.

In the next chapter, we investigate techniques for solving higher-order ordinary differential equation initial-value problems (ODE-IVP) and systems of first-order ODE-IVP. In Chapter 14, we consider techniques for solving ODE boundary-value problems (ODE-BVP). Finally, Chapter 15 presents an introduction to methods for numerically solving partial differential equations.

Application 12-A Motion of a Falling Body

The motion of a falling body is described by Newton's second law, $F = ma$. The acceleration a is the rate of change of the velocity of the body with respect to time. The forces acting on the body may include, in addition to gravity, air resistance that is proportional to a power of the velocity. Empirical studies suggest that air resistance can be modeled as

$$F = kv^p$$

where $1 \leq p \leq 2$ and the value of the proportionality constant k depends on the size and shape of the body, as well as the density and viscosity of the air. Typically, $p = 1$ for relatively slow velocities and $p = 2$ for high velocities. For intermediate velocities (with $0 < p < 1$), numerical methods may be especially appropriate.

Resistance Proportional to Velocity

For $p = 1$, the differential equation takes the form

$$m\frac{dv}{dt} = -kv - mg$$

with the positive y-direction upward and $y = 0$ at ground level. Thus, when the body is falling ($v < 0$), the resistance force is positive (upward); when the body is rising ($v > 0$), the resistance acts in the downward direction.

The ratio $r = k/m$ is known as the drag coefficient; in terms of r the differential equation becomes $dv/dt = -rv - g$.

It is easy to find that the terminal speed is

$$|v_t| = \frac{g}{r} = \frac{mg}{k}$$

For a parachutist, a typical drag coefficient is $r = k/m \approx 1.6$, so the terminal speed is on the order of 20 ft/s. (See Edwards and Penney, 2004.)

Resistance Proportional to Velocity Squared

For resistance proportional to the square of the velocity, we must distinguish between upward and downward motion, to be sure that the resistance force is acting opposite to the motion of the object. If we leave our coordinate axis as before (with up positive), we have (in terms of the drag coefficient, $r = k/m$)

$$\frac{dv}{dt} = -rv^2 - g \quad \text{(for a body moving upward)}$$

or

$$\frac{dv}{dt} = rv^2 - g \quad \text{(for a body moving downward)}$$

The velocity for downward motion in this case is

$$v(t) = \sqrt{\frac{g}{r}}\,\frac{1 + Ce^{dt}}{1 - Ce^{dt}}$$

where $d = 2\sqrt{gr}$. The terminal velocity is

$$v_f = -\sqrt{\frac{g}{r}}$$

12.1 EULER'S METHOD

In this section, we consider Euler's method, which is the most basic procedure for finding the approximate solution of a first-order ODE, $y' = f(x, y)$, with initial condition $y(x_0) = y_0$ on an interval $[x_0, x_n]$ (or $[a, b]$). We find the step size $h = (x_0 - x_n)/n$, and find the approximate solution values $y_1, y_2, \ldots, y_n$ at the points $x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh$ by computing

$$y_{i+1} = y_i + hf(x_i, y_i)$$

for $i = 0, 1, 2, \ldots, n - 1$.

There are several ways of looking at this formula. The geometric interpretation gives an intuitive motivation for the method. Algebraically the method can be viewed as coming from a numerical approximation to the derivative or as using numerical integration, or as being derived from the Taylor series expansion for the unknown function y. Each of these views serves as an introduction to more powerful methods presented in the following sections.

12.1.1 Geometric Introduction

Geometrically, Euler's method corresponds to using the line tangent to the solution curve $y(x)$ to find the value of y_1, the approximation to the value of y at x_1. The equation of the tangent line, with slope $y' = f(x_0, y_0)$ and passing through the point (x_0, y_0), is $y - y_0 = (x - x_0)f(x_0, y_0)$. At $(x, y) = (x_1, y_1)$ this gives

$$y_1 = y_0 + (x_1 - x_0)f(x_0, y_0)$$

In general, going from (x_i, y_i) to (x_{i+1}, y_{i+1}), (and using the fact that $h = x_{i+1} - x_i$), Euler's method gives

$$y_{i+1} = y_i + hf(x_i, y_i) \quad \text{for} \quad i = 0, 1, \ldots, n.$$

If we take $f(x, y) = x + y$, $x_0 = 0$, $y_0 = 2$, and $h = 1/2$, the first step of Euler's method proceeds along the straight line shown in Figure 12.1.

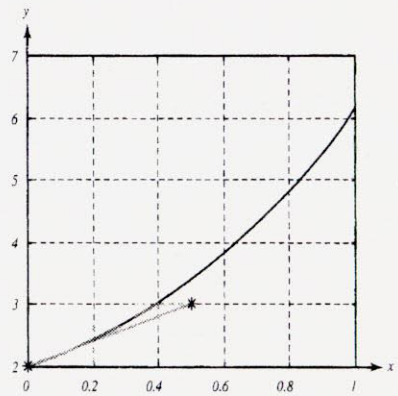

FIGURE 12.1: Exact solution and first step of Euler's method for $y' = x + y$.

12.1.2 Approximating the Derivative

Algebraically, Euler's method can be found by replacing the derivative y' by the forward difference approximation; at the first step

$$y' = f(x, y)$$

becomes

$$\frac{y_1 - y_0}{x_1 - x_0} = f(x_0, y_0)$$

so that

$$y_1 = y_0 + (x_1 - x_0) f(x_0, y_0)$$

or, in general (to go from (x_i, y_i) to (x_{i+1}, y_{i+1})), and using $h = (x_n - x_0)/n = x_{i+1} - x_i$,

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

EXAMPLE 12.1 Solving a Simple ODE with Euler's Method

Consider the differential equation $y' = f(x, y)$ on $a = x_0 \le x \le b = x_n$. Let

$$y' = x + y, \quad y(0) = 2, \quad \text{for } 0 \le x \le 1$$

First, we find the approximate solution for $h = 0.5$ $(n = 2)$, a very large step size. The approximation at $x_1 = 0.5$ is

$$y_1 = y_0 + h(x_0 + y_0) = 2.0 + 0.5(0.0 + 2.0) = 3.0$$

Next, we find the approximate solution y_2 at $x_2 = 0.0 + 2h = 1.0$:

$$y_2 = y_1 + h(x_1 + y_1) = 3.0 + 0.5(0.5 + 3.0) = 4.75$$

To find a better approximate solution, we use $n = 20$ intervals, so that $h = 0.05$. Figure 12.2 shows the computed solution, and the exact solution, $y = 3e^x - x - 1$.

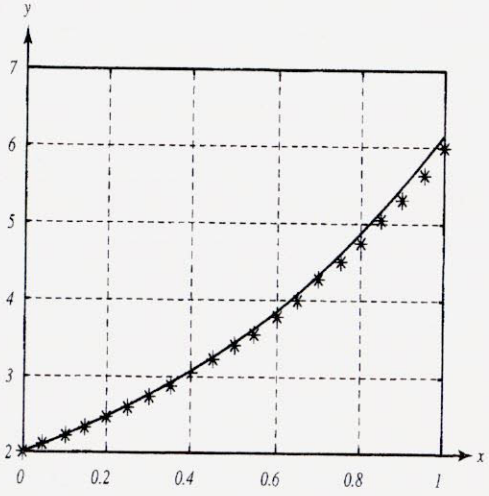

FIGURE 12.2: Exact solution and Euler solution for $n = 20$.

12.1.3 Approximating the Integral

From another perspective, Euler's method may be seen as the result of writing the differential equation as an integral equation:

$$\int_{y_0}^{y_1} dy = \int_{x_0}^{x_1} f(x, y)dx$$

Using the left rectangle approximation to the integral, we have

$$\int_{y_0}^{y_1} dy = \int_{x_0}^{x_1} f(x_0, y_0)\, dx$$

or

$$y_1 = y_0 + hf(x_0, y_0)$$

As above, the generalization from the first step to the i^{th} step is trivial.

A MATLAB function for Euler's method on the interval $[a, b]$ is given below, followed by an example (which includes the MATLAB function for the ODE). The interval $[a, b]$ is defined as the vector `tspan` in the the MATLAB functions for ODE given in this chapter so that the syntax for calling these functions corresponds to the syntax for calling the built-in MATLAB functions for ODE.

Euler's Method

```
function [x , y]  = Euler( f, tspan, y0, n )
% solve y' = f(x,y) with initial condition y(a) = y0
% f is an inline function
a = tspan(1);          b = tspan(2);
h = (b-a) / n;         x = (a+h : h : b);
y(1) = y0 + h* feval(f, a, y0);
for i = 2 : n
    y(i) = y(i-1) + h* feval(f, x(i-1), y(i-1));
end
x = [ a   x ];         y = [ y0  y ];
```

To illustrate the use of the simple function above, consider the ODE from Example 12.1. We define $f(x, y) = x + y$ as an inline function and use the function `Euler` to obtain a table of values for $y(x)$ at $x = 0, 0.2, \ldots, 1.0$. The input is

```
f = inline('x + y');
[x, y] = Euler( f, [0, 1], 2, 5 )
```

The results from the function are

```
x =    0        0.2000    0.4000    0.6000    0.8000    1.0000
y =    2.0000   2.4000    2.9200    3.5840    4.4208    5.4650
```

The next example shows the use of Euler's method for a more challenging problem.

EXAMPLE 12.2 A More Challenging Example

Consider the differential equation

$$y' = f(x, y) = \begin{cases} y(-2x + 1/x) & \text{if} \quad x \neq 0 \\ 1 & \text{if} \quad x = 0 \end{cases}$$

on the interval $0 \leq x \leq 2$ with initial value $y(0) = 0.0$. With $n = 10$ intervals, the step size $h = (b - a)/n = 0.2$. The approximate and exact solutions are graphed in Figure 12.3. The approximate values, z, and the values of the exact solution $y = x \exp(-x^2)$, are listed below.

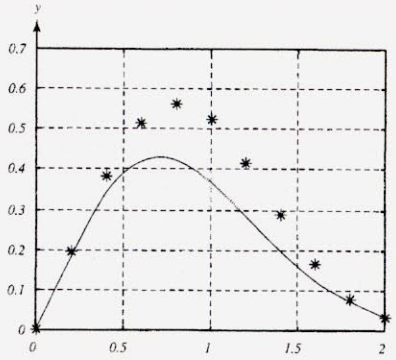

FIGURE 12.3: Euler solution with $n = 10$, and exact solution.

Grid points:

$$x = [0.000 \quad 0.200 \quad 0.400 \quad 0.600 \quad 0.800 \quad 1.000 \quad 1.200 \quad 1.400 \quad 1.600 \quad 1.800 \quad 2.000]$$

Approximate solution:

$$z = [0.000 \quad 0.200 \quad 0.384 \quad 0.515 \quad 0.563 \quad 0.523 \quad 0.419 \quad 0.287 \quad 0.168 \quad 0.081 \quad 0.0317]$$

Exact solution:

$$y = [0.100 \quad 0.192 \quad 0.341 \quad 0.419 \quad 0.422 \quad 0.368 \quad 0.284 \quad 0.197 \quad 0.124 \quad 0.705 \quad 0.0366]$$

Function for Example 12.2

```
function dy = f12_2(x, y)
n = length(x);
for i = 1 : n
    if (x(i)==0),      dy(i) = 1;
    else,              dy(i) = y(i)*(- 2*x(i) + 1/x(i));
    end
end
```

12.1.4 Using Taylor Series

Our final view of Euler's method is that it is based on the first term in a Taylor series approximation to the unknown function $y(x)$. Since the Taylor series with remainder is

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2}y''(\eta)$$

we find Euler's method by taking $y(x+h) = y_{i+1}, y(x) = y_i$, and $y'(x) = f(x_i, y_i)$ and neglecting the remainder.

This suggests that one way to obtain a better solution technique is to use more terms in the Taylor series for y, in order to obtain higher-order truncation error. For example, a second-order Taylor method uses

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2}y''(x) + O(h^3)$$

However, we do not have a formula for $y''(x)$. In the next section, we consider an approach that allows us to achieve results that agree with retaining higher-order terms in the Taylor series expansion without the necessity of computing higher derivatives of y.

Truncation Error

The error in neglecting the remainder is called the *local truncation (or discretization) error*. Thus the local truncation error for Euler's method is $O(h^2)$. Since the actual form of y is, in general, unknown, the dependence of the error on the step size is the most direct way to compare methods.

We denote the *total truncation (or discretization) error* in going from x_0 to $x_n = x_0 + nh$ as e_n. To find a bound on e_n, assume that f satisfies a Lipschitz condition, i.e., there is a constant L such that

$$|f(x, y_2) - f(x, y_1)| < L|y_2 - y_1|$$

and that the second derivative of y is bounded (so that $|y''(\eta)| \leq N$). Then the error in going from x_0 to x_n is

$$|\epsilon_n| \leq \frac{h}{2}\frac{N}{L}(e^{L(x_n - x_0)} - 1)$$

Thus, the total truncation error for Euler's method is $O(h)$ and it is a *first-order method*. This means that if the step size is reduced by a factor of $1/2$, we expect the error to be reduced by approximately the same factor.

In the discussions that follow, the order of a method is taken to be the order of the total (global) truncation error.

12.2 RUNGE-KUTTA METHODS

The previous section suggested that using more terms in the Taylor series representation for the unknown function y gives more accurate results, but the necessary computation of derivatives of $f(x, y)$ is often too difficult to make the higher-order Taylor methods attractive.

In this section, we consider a family of methods that give results that match the higher-order Taylor series expansions, but do not require differentiating $f(x, y)$. We begin with a second-order method with a simple geometric interpretation, and then show how this and other second-order methods can be found algebraically. The formulas for several higher-order methods are also given. The basic idea is to find a sequence of approximations to the change $y_{i+1} - y_i$, and then use a combination of these to compute the new y.

12.2.1 Second-Order Runge-Kutta Methods

The simplest Runge-Kutta methods use one evaluation of the function $f(x, y)$ in addition to the evaluation at the starting point for each step. The point at which the function is evaluated is chosen so that the results agree with a second-order Taylor expansion for y.

Midpoint Method

One of the simplest of the Runge-Kutta methods is known as the midpoint method. It is based on approximating the value of y at the midpoint of the interval — i.e., at $x_i + h/2$ — by taking one-half of the change in y that is given by Euler's method and adding that on to the current value y_i. The first step is illustrated in Figure 12.4. The point (x_e, y_e) would be the next estimate from Euler's method; the slope at (x_m, y_m) is used to find (x_2, y_2).

Although the geometric interpretation helps the method seem more intuitively plausible, the derivation of a Runge-Kutta method depends on finding ways of approximating the slope of y, using the function f evaluated at various points in the interval, so that the accuracy agrees with that obtained from Taylor series approximations.

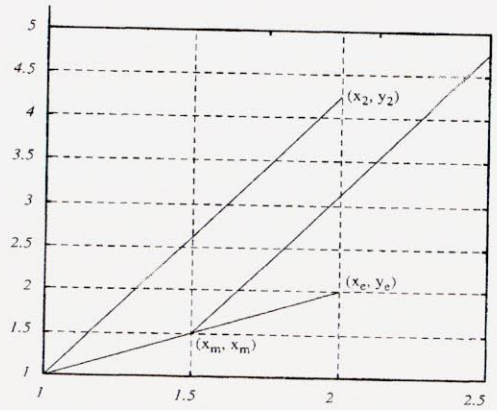

FIGURE 12.4: Geometric interpretation of midpoint method.

The Runge-Kutta midpoint method is given by the computations:

$$k_1 = hf(x_i, y_i) \qquad \text{(change given by Euler's method)}$$
$$k_2 = hf(x_i + 0.5h, y_i + 0.5k_1) \qquad \text{(change using slope at midpoint)}$$
$$y_{i+1} = y_i + k_2$$

Midpoint Method

```
function [x , y]  = RK2_Midpt(f, tspan, y0, n)
a = tspan(1);   b = tspan(2);      h = (b-a)/n;      x = (a+h : h : b);
k1 = h*feval(f, a, y0);            k2 = h*feval(f, a + h/2, y0 +k1/2);
y(1) = y0 + k2;
for i = 1 : n-1
    k1 = h*feval(f, x(i), y(i));  k2 = h*feval(f, x(i)+h/2, y(i)+k1/2);
    y(i+1) = y(i) + k2;
end
x = [ a    x ];      y = [ y0   y ];
```

EXAMPLE 12.3 ODE for Dawson's Integral

Consider the ODE $y' = 1 - 2xy, y(0) = 0$.

The results of solving the ODE using the midpoint method, with $a = 0$, $b = 1$, $n = 10$, plotted in Figure 12.5, are as follows:

```
x =
0.00   0.10   0.20   0.30   0.40   0.50   0.60   0.70   0.80   0.90   1.00
y =
0.00   0.099  0.19   0.28   0.36   0.42   0.47   0.51   0.53   0.54   0.54
```

The exact solution, $y = \exp(-x^2) \int_0^x \exp(t^2)dt$, is known as Dawson's integral.

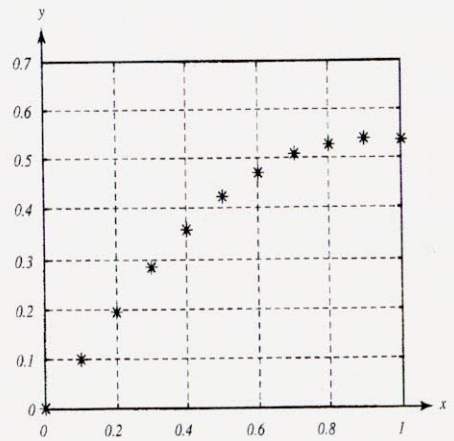

FIGURE 12.5: $y' = 1 - 2xy$, midpoint method, $n = 10$.

EXAMPLE 12.4 Solving a Simple ODE with the Midpoint Method

Consider the differential equation $y' = f(x, y)$ on $a \leq x \leq b$.

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

First, we find the approximate solution for $h = 0.5$ $(n = 2)$, a very large step size. The approximation at $x_1 = 0.5$ is

$$k_1 = hf(x_0, y_0) = 0.5(x_0 + y_0) = 1.0$$
$$k_2 = hf(x_0 + 0.5h, y_0 + 0.5k_1) = 0.5(x_0 + 0.25 + y_0 + 0.5) = 1.375$$
$$y_1 = y_0 + k_2 = 3.375$$

Next, we find the approximate solution y_2 at point $x_2 = 0.0 + 2h = 1.0$:

$$k_1 = hf(x_1, y_1) = 0.5(x_1 + y_1) = 0.5(0.5 + 3.375) = 1.9375$$
$$k_2 = hf(x_1 + 0.5h, y_1 + 0.5k_1) = 0.5(0.5 + 0.25 + 3.375 + 0.97) = 2.547$$
$$y_2 = y_1 + k_2 = 3.375 + 2.5469 = 5.922$$

The computed points and the exact solution, $y = 3e^x - x - 1$, are shown in Figure 12.6. With $n = 10$, the computed values appear to fall directly on the graph of the exact solution.

For comparison, the midpoint method with $n = 10$ requires approximately the same number of function evaluations as does Euler's method with $n = 20$ (Figure 12.2) and the fourth-order Runge-Kutta method presented in the next section, with $n = 5$.

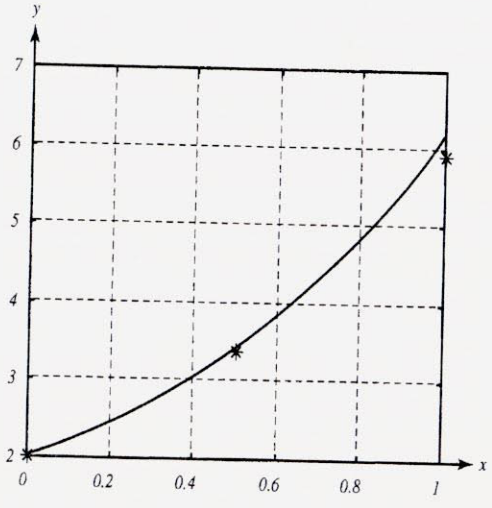

FIGURE 12.6: $y' = x + y$ using midpoint method, $n = 2$, and exact solution.

Second-Order Runge-Kutta Methods

We can define a second-order Runge-Kutta method by giving the equations for k_1 and k_2 along with the the weighted combination k_1 and k_2 used to compute the next y. Or we can summarize such a method by listing its parameters in an array. The rows of the array give the coefficients for the computation of each k (except for k_1, which is always computed in the same way). A parameter array is even more useful for higher-order methods, which we consider shortly.

The general form for a second-order Runge-Kutta method is

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + c_2 h, y_i + a_{21} k_1) \\ y_{i+1} &= y_i + w_1 k_1 + w_2 k_2 \end{aligned}$$

The midpoint method, discussed above, is given by:

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \\ y_{i+1} &= y_i + k_2 \end{aligned}$$

The trapezoid method is given by the equations

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + h, y_i + k_1) \\ y_{i+1} &= y_i + \frac{1}{2}k_1 + \frac{1}{2}k_2 \end{aligned}$$

The optimal method

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{2}{3}h, y_i + \frac{2}{3}k_1) \\ y_{i+1} &= y_i + \frac{1}{4}k_1 + \frac{3}{4}k_2 \end{aligned}$$

corresponds to minimizing the bound on the lowest-order derivative term in the error (Ralston and Rabinowitz, 1978, p. 216).

The parameter array for a general second-order Runge-Kutta method is

0		
c_2	a_{21}	
	w_1	w_2

Parameters for the midpoint method:

0		
1/2	1/2	
	0	1

Parameters for the trapezoid method:

0		
1	1	
	1/2	1/2

Parameters for the optimal method:

0		
2/3	2/3	
	1/4	3/4

Discussion

To derive the parameters for a second-order Runge-Kutta formula, we write it as

$$y(x+h) = y(x) + h[w_1 f(x,y) + w_2 f(x + c_2 h, y + a_{21} k_1)] \tag{12.1}$$

We then form the Taylor expansion of $f(x + c_2 h, y + a_{21} k_1)$ and find values of the parameters w_1, w_2, c_2, and a_{21}, so that the Runge-Kutta method agrees with the second-order Taylor method.

The Taylor expansion for $f(x + c_2 h, y + a_{21} k_1)$, a function of two variables, is

$$f(x + c_2 h, y + a_{21} k_1) = f(x,y) + c_2 h f_x(x,y) + a_{21} k_1 f_y(x,y)$$

Substituting this into Eq. (12.1) [and abbreviating $f(x,y)$ as f], we have the Runge-Kutta formula

$$y(x+h) = y(x) + h w_1 f + h w_2 f + w_2 c_2 h^2 f_x + w_2 a_{21} k_1 h f_y \tag{12.2}$$

We want this to agree with the second-order Taylor formula for y (a function of one variable):

$$y(x+h) = y(x) + hf + \frac{h^2}{2}[f_x + f_y f] \tag{12.3}$$

where again f, f_x, and f_y are all evaluated at (x,y).

Matching the terms involving f in Eqs. (12.2) and (12.3) gives $w_1 + w_2 = 1$.

Matching the terms with f_x gives $w_2 c_2 h^2 = h^2/2$.

Matching the terms with f_y gives $w_2 a_{21} k_1 h = (h^2/2) f$; since $k_1 = hf$, we have $w_2 a_{21} = 1/2$.

Thus, there are three equations,

$$\begin{aligned} w_1 + w_2 &= 1 \\ w_2 c_2 &= 1/2 \\ w_2 a_{21} &= 1/2 \end{aligned}$$

for the four parameters. If c_2 is taken to be the free parameter, we have the general form

$$\begin{array}{c|cc} 0 & & \\ c_2 & c_2 & \\ \hline & 1 - \dfrac{1}{2c_2} & \dfrac{1}{2c_2} \end{array}$$

Keeping the next higher terms in the Taylor series expansion allows the computation of the local error term. The truncation error for the second-order Runge-Kutta method is

$$T_n = ([\frac{1}{6} - \frac{c_2^2 w_2}{2}] D^2 f + \frac{1}{6} f_y D f) h^3 + O(h^4)$$

where $D = \partial/\partial x + f_n \partial/\partial y$.

Although it is not easy to get good bounds on f and its derivatives, the choice of w_2 that makes the coefficient of D^2 be zero yields the method known as "optimal." Bounds on the truncation error are usually expressed in terms of bounds on f and its various partial derivatives. (See Ralston and Rabinowitz, 1978.)

12.2.2 Third-Order Runge-Kutta Methods

The general form for a third-order Runge-Kutta method is

$$
\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + c_2 h, y_i + a_{21} k_1) \\
k_3 &= hf(x_i + c_3 h, y_i + a_{31} k_1 + a_{32} k_2) \\
y_{i+1} &= y_i + w_1 k_1 + w_2 k_2 + w_3 k_3
\end{aligned}
$$

The array of parameters has the following form:

$$
\begin{array}{c|ccc}
0 & & & \\
c_2 & a_{21} & & \\
c_3 & a_{31} & a_{32} & \\
\hline
 & w_1 & w_2 & w_3
\end{array}
$$

The first row gives the parameters needed to compute k_1 (which is always $k_1 = hf(x_i, y_i)$ so c_1 is always 0), the second row corresponds to k_2, the third row gives parameters to compute k_3, and the row below the line gives the weights used to form y_{i+1}.

The parameters for some well-known third-order methods are given here; others are considered in the exercises.

$$
\begin{array}{c|ccc}
0 & & & \\
1/2 & 1/2 & & \\
1 & -1 & 2 & \\
\hline
 & 1/6 & 2/3 & 1/6
\end{array}
$$

RK3 Kutta's method

$$
\begin{array}{c|ccc}
0 & & & \\
1/2 & 1/2 & & \\
3/4 & 0 & 3/4 & \\
\hline
 & 2/9 & 3/9 & 4/9
\end{array}
$$

RK3 optimal method

$$
\begin{array}{c|ccc}
0 & & & \\
2/3 & 2/3 & & \\
2/3 & 1/3 & 1/3 & \\
\hline
 & 1/4 & 0 & 3/4
\end{array}
$$

RK3 two-thirds rule

The parameters of a third-order Runge-Kutta method satisfy:

$$\begin{aligned} w_1 + w_2 + w_3 &= 1 \\ c_2 w_2 + c_3 w_3 &= \frac{1}{2} \\ c_2^2 w_2 + c_3^2 w_3 &= \frac{1}{3} \\ c_2 a_{32} w_3 &= \frac{1}{6} \\ c_2 &= a_{21} \\ c_3 &= a_{31} + a_{32} \end{aligned}$$

Since there are eight parameters and six equations, two parameters can be chosen to optimize some aspect of the method.

There are two special cases in which the general form of the parameter array simplifies significantly. The first is when $c_2 = c_3 = 2/3$. The array then has the form

$$\begin{array}{c|ccc} 0 & & & \\ \frac{2}{3} & \frac{2}{3} & & \\ \frac{2}{3} & \frac{2}{3} - \frac{1}{4w_3} & \frac{1}{4w_3} & \\ \hline & \frac{1}{4} & \frac{3}{4} - w_3 & w_3 \end{array}$$

The "RK3 two-thirds rule" given above is of this form. Another example of this form, with $w_3 = 3/8$, is given in the exercises.

The other special case is when $c_2 = 2/3; c_3 = 0$. The array then has the form

$$\begin{array}{c|ccc} 0 & & & \\ \frac{2}{3} & \frac{2}{3} & & \\ 0 & -\frac{1}{4w_3} & \frac{1}{4w_3} & \\ \hline & \frac{1}{4} - w_3 & \frac{3}{4} & w_3 \end{array}$$

Butcher (p. 157) also gives the general form for the array of parameters for Runge-Kutta third-order methods (with c_2 and c_3 as the free parameters).

12.2.3 Classic Runge-Kutta Method

Probably the most common form of the Runge-Kutta method is the classic fourth-order method. It uses a linear combination of four function evaluations:

$$
\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + 0.5h, y_i + 0.5k_1) \\
k_3 &= hf(x_i + 0.5h, y_i + 0.5k_2) \\
k_4 &= hf(x_i + h, y_i + k_3)
\end{aligned}
$$

These four equations, together with

$$
y_{i+1} = y_i + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4
$$

make up the method.

The following MATLAB function implements the classic fourth-order Runge-Kutta method. This form requires that the function f be defined as an inline function in the command window before calling `RK4s`.

```
f = inline('x + y', 'x', 'y');
[x,y] = RK4s(f,[0, 1], 2, 5)
```

Classic Runge-Kutta Method

```
function [x , y]  = RK4s(f, tspan, y0, n)
% solve y' = f(x,y) with initial condition y(a) = y0
% using n steps of the classic fourth-order Runge-Kutta method;
a = tspan(1);
b = tspan(2);
h = (b-a)/n;
x = a : h : b;
y(1) = y0;
for i = 1 : n
    k1 = h * f( x(i), y(i) );
    k2 = h * f( x(i)+0.5*h, y(i)+0.5*k1 );
    k3 = h * f( x(i)+0.5*h, y(i)+0.5*k2 );
    k4 = h * f( x(i)+h , y(i)+k3 );
    y(i+1) = y(i) + (k1  + 2*k2  + 2*k3  + k4) / 6;
end
```

For more complicated ODE whose right-hand side cannot be expressed as an inline function, the statements of the form `f(x(i), y(i))` in `RK4s` must be replaced by `feval(f, x(i), y(i))`. The name of the function defining the ODE is then passed as an input argument to the function `RK4`.

EXAMPLE 12.5 Using the Classic Runge-Kutta Method

Consider again the differential equation

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

with

$$a = 0, \quad b = 1, \quad n = 5, \quad \text{and} \quad h = 0.2$$

For comparison with the results of Euler's method, observe that this choice of n will require approximately the same number of evaluations of $f(x, y)$ as are used in Euler's method with $n = 20$.

The graphs of the approximate and exact solutions are almost indistinguishable. The table of values shows that the error is much less than for Euler's method. The computed points and the exact solution, $Y(x) = 3e^x - x - 1$, are shown in Figure 12.7, and tabulated (to four decimal places) in the table below.

x_i	$Y(x_i)$	y_i	$\lvert Y(x_i) - y_i \rvert$
0.2	2.4642	2.4642	8.2745e-06
0.4	3.0755	3.0755	2.0213e-05
0.6	3.8664	3.8663	3.7032e-05
0.8	4.8766	4.8766	6.0308e-05
1.0	6.1548	6.1548	9.2076e-05

Solving $y' = x + y$ with classic Runge-Kutta.

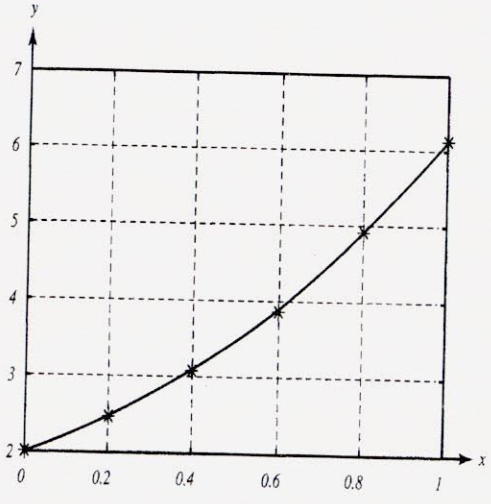

FIGURE 12.7: $y' = x + y$ with classic Runge-Kutta method, $n = 5$.

EXAMPLE 12.6 Another Example Using the Classic Runge-Kutta Method

Consider once more the differential equation

$$y' = f(x, y) = \begin{cases} y(-2x + 1/x) & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}$$

on the interval $0 \leq x \leq 2$ with initial value $y(0) = 0.0$. The values of the exact solution, $Y(x) = x\exp(-x^2)$, and the approximate solution at the mesh points (with $n = 10$) are shown in the following table and graphed in Figure 12.8.

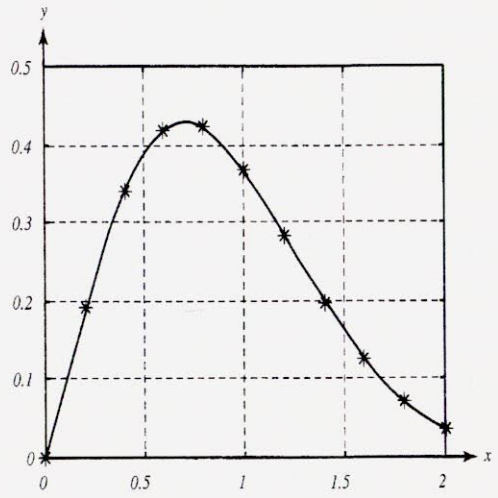

FIGURE 12.8: $y' = y(-2x + 1/x)$, fourth-order Runge-Kutta method ($n = 10$).

x_i	$Y(x_i)$	y_i	$\lvert Y(x_i) - y_i \rvert$
0.2	0.19216	0.19215	1.2288e-05
0.4	0.34086	0.34093	7.1314e-05
0.6	0.41861	0.41872	1.1072e-04
0.8	0.42183	0.42195	1.1763e-04
1.0	0.36788	0.36798	1.0280e-04
1.2	0.28431	0.28439	7.9678e-05
1.4	0.19720	0.19727	6.5398e-05
1.6	0.12369	0.12376	7.0105e-05
1.8	0.070495	0.070584	8.9035e-05
2.0	0.036631	0.036738	1.0716e-04

$y' = y(-2x + 1/x)$, fourth-order Runge-Kutta method ($n = 10$).

12.2.4 Fourth-Order Runge-Kutta Methods

The Runge-Kutta methods described in the previous sections are only the mos common, simple forms of a very extensive field of study. For fourth-order methods

$$
\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + c_2 h, y_i + a_{21} k_1) \\
k_3 &= hf(x_i + c_3 h, y_i + a_{31} k_1 + a_{32} k_2) \\
k_4 &= hf(x_i + c_4 h, y_i + a_{41} k_1 + a_{42} k_2 + a_{43} k_3) \\
y_{i+1} &= y_i + w_1 k_1 + w_2 k_2 + w_3 k_3 + w_4 k_4
\end{aligned}
$$

The parameters may be shown in an array:

$$
\begin{array}{c|cccc}
0 & & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
c_4 & a_{41} & a_{42} & a_{43} & \\
\hline
 & w_1 & w_2 & w_3 & w_4
\end{array}
$$

The entries in the first column are the coefficients of h; the other elements in each row are the coefficients of k_2, k_3, and k_4 that are used to determine the value of y used to compute the next k. Finally, the entries below the line are the weights used to take the linear combination of the k's to compute the value of the next y.

The parameter arrays for two fourth-order methods are as follows:

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Classic Runge-Kutta method

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{3} & \frac{1}{3} & & & \\
\frac{2}{3} & -\frac{1}{3} & 1 & & \\
1 & 1 & -1 & 1 & \\
\hline
 & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8}
\end{array}
$$

Kutta's fourth-order method

The following array of parameters gives another fourth-order method, which will be used in combination with a higher-order method in the next section (Ralston and Rabinowitz, pp. 218, 219).

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & & \\ 1 & 0 & -1 & 2 & \\ \hline & \frac{1}{6} & 0 & \frac{2}{3} & \frac{1}{6} \end{array}$$

This is similar to one of the special forms in Butcher (p. 166), which is given as

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{4} & \frac{1}{4} & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & & \\ 1 & 1 & -2 & 2 & \\ \hline & \frac{1}{6} & 0 & \frac{2}{3} & \frac{1}{6} \end{array}$$

Eight equations must be satisfied for a fourth-order Runge-Kutta method. Except for certain special cases, the equations can be solved for the other parameters in terms of c_2 and c_3. In all cases, $c_4 = 1$.

$$\begin{aligned} w_1 + w_2 + w_3 + w_4 &= 1 \\ c_2 w_2 + c_3 w_3 + c_4 w_4 &= \frac{1}{2} \\ c_2^2 w_2 + c_3^2 w_3 + c_4^2 w_4 &= \frac{1}{3} \\ c_2 a_{32} w_3 + (c_2 a_{42} + c_3 a_{43}) w_4 &= \frac{1}{6} \\ c_2^3 w_2 + c_3^3 w_3 + c_4^3 w_4 &= \frac{1}{4} \\ c_2^2 a_{32} w_3 + (c_2^2 a_{42} + c_3^2 a_{43}) w_4 &= \frac{1}{12} \\ c_2 c_3 a_{32} w_3 + c_4 (c_2 a_{42} + c_3 a_{43}) w_4 &= \frac{1}{8} \\ c_2 a_{32} a_{43} w_4 &= \frac{1}{24} \end{aligned}$$

12.2.5 Fifth-Order Runge-Kutta Methods

Runge-Kutta methods of order higher than fourth require more stages than th order of the method. A fifth-order method requires six stages, i.e., six functio evaluations.

A fifth-order Runge-Kutta method has the form

$$
\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + c_2 h, y_i + a_{21} k_1) \\
&\vdots \\
k_6 &= hf(x_i + c_6 h, y_i + a_{61} k_1 + a_{62} k_2 + \cdots + a_{65} k_5) \\
y_{i+1} &= y_i + w_1 k_1 + w_2 k_2 + \cdots + w_6 k_6
\end{aligned}
$$

Two fifth-order methods are given here. The first, given by Butcher, p. 90, is similar to a method given by Jain, which he attributes to Larson.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$				
$\frac{1}{2}$	0	0	$\frac{1}{2}$			
$\frac{3}{4}$	$\frac{3}{16}$	$-\frac{3}{8}$	$\frac{3}{8}$	$\frac{9}{16}$		
1	$-\frac{3}{7}$	$\frac{8}{7}$	$\frac{6}{7}$	$-\frac{12}{7}$	$\frac{8}{7}$	
	$\frac{7}{90}$	0	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$

Butcher, p. 174, refers to the following as "one of Kutta's formulas as corrected by E. J. Nystrom" (a slightly different form, from Milne, p. 73, is given in the exercises).

0						
$\frac{1}{3}$	$\frac{1}{3}$					
$\frac{2}{5}$	$\frac{4}{25}$	$\frac{6}{25}$				
1	$\frac{1}{4}$	-3	$\frac{15}{4}$			
$\frac{2}{3}$	$\frac{2}{27}$	$\frac{10}{9}$	$-\frac{50}{81}$	$\frac{8}{81}$		
$\frac{4}{5}$	$\frac{7}{30}$	$\frac{3}{5}$	$-\frac{1}{6}$	$\frac{2}{15}$	0	
	$\frac{1}{4}$	0	$\frac{125}{192}$	0	$-\frac{27}{64}$	$\frac{25}{48}$

12.2.6 Runge-Kutta-Fehlberg Methods

As we saw for numerical integration, it is helpful to have an estimate of the error in finding a numerical solution of an ODE, as long as it does not take too much extra effort to compute the estimate.

The Runge-Kutta-Fehlberg methods use a pair of Runge-Kutta methods to obtain both the computed solution and an estimate of the truncation error. The estimate of the error can be used in programs of variable step size to decide when to adjust the step size. The idea is to find a pair of methods, typically of order n and $n+1$, where the computations for the lower-order method are used again in the higher-order method. We refer the embedded Runge-Kutta pairs as RKF methods, although the methods developed by Fehlberg are only one of several groups of methods of this type.

Midpoint-Kutta

We begin with a simple combination of a second-order and a third-order method. The second-order method is the midpoint method:

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \\ y_{i+1} &= y_i + k_2 \end{aligned}$$

The third-order method is Kutta's third-order method

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \\ k_3 &= hf(x_i + h, y_i - k_1 + 2k_2) \\ y_{i+1} &= y_i + (k_1 + 4k_2 + k_3)/6 \end{aligned}$$

The estimate of the local truncation error L can be found from k_1, k_2, and k_3:

$$L = \frac{1}{6}k_1 + \frac{2}{3}k_2 + \frac{1}{6}k_3 - k_2 = \frac{1}{6}k_1 - \frac{1}{3}k_2 + \frac{1}{6}k_3$$

In order to see why this is the case, let Y be the true solution at x_{n+1}, y the solution computed from the midpoint method, and z be the solution computed from Kutta's third-order method. We write the truncation error for the midpoint method as $Y - y = B(x_i)h^3 + O(h^4)$ and the truncation error for Kutta's third-order method as $Y - z = C(x_i)h^4 + O(h^5)$, where $B(x_i)$ and $C(x_i)$ depend on $f(x, y)$ and its derivatives. Subtracting, we find

$$z - y = B(x_i)h^3 + O(h^4)$$

Neglecting the higher-order terms, we can solve for $B(x_i)h^3 = |z - y|$ and substitute in the equation for the truncation error for the midpoint method to find the estimate of the local truncation error L as

$$L = |Y - y| \approx B(x_i)h^3 \approx |z - y|$$

Another RKF23

Here is another simple combination of a second-order and a third-order method. The second-order method is the trapezoid method:

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + h, y_i + k_1) \\ y_{i+1} &= y_i + \frac{1}{2}k_1 + \frac{1}{2}k_2 \end{aligned}$$

The third-order method is

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + h, y_i + k_1) \\ k_3 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{4}k_1 + \frac{1}{4}k_2) \\ y_{i+1} &= y_i + \frac{1}{6}k_1 + \frac{1}{6}k_2 + \frac{2}{3}k_3 \end{aligned}$$

The estimate of the local truncation error L can be found from k_1, k_2, and k_3:

$$\begin{aligned} L &= \frac{1}{6}k_1 + \frac{1}{6}k_2 + \frac{2}{3}k_3 - \frac{1}{2}k_1 - \frac{1}{2}k_2 \\ &= -\frac{1}{3}k_1 - \frac{1}{3}k_2 + \frac{2}{3}k_3 \end{aligned}$$

The parameters for embedded Runge-Kutta methods are often given in a single table. Here is the table for the embedded method given above.

$$\begin{array}{c|ccc} 0 & & & \\ 1 & 1 & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ \hline & \frac{1}{2} & \frac{1}{2} & \\ & \frac{1}{6} & \frac{1}{6} & \frac{4}{6} \\ - & \frac{1}{-3} & \frac{1}{-3} & \frac{2}{3} \end{array}$$

Below the solid line, the coefficients are given for computing y_{i+1} in the second-order method, followed by the coefficients for computing it in the third-order method, and below the dashed line, the difference of those coefficients, which gives the estimate of the error in using the lower-order computation.

ODE23

The following is a simplified version of the method used in the ODE23 routine in MATLAB. It is also described (as BS23) in Moler (available online). It is originally due to Bogacki and Shampine.

$$
\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \\
k_3 &= hf(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_2) \\
y_{i+1} &= y_i + \frac{2}{9}k_1 + \frac{3}{9}k_2 + \frac{4}{9}k_3 \\
k_4 &= hf(x_i + h, y_{i+1}) \\
e_{i+1} &= h(-\frac{5}{72}k_1 + \frac{6}{72}k_2 + \frac{8}{72}k_3 - \frac{9}{72}k_4)
\end{aligned}
$$

Although there is one additional computation, k_4 used to find the error, it becomes the k_1 for the next step; this is an example of embedded methods known as LSAF (last same as first). The third-order method used to compute y_{i+1} is the method listed previously as "optimal." Note that

$$k_4 = hf(x_i + h, y_i + \frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3)$$

although it would be inefficient to recompute it after finding y_{i+1}.

The coefficients for finding y_{i+1} using the lower-order method can be deduced from the error formula if desired (the lower-order method uses all four of the k's), but as long as the higher-order computation for y_{i+1} is propagated forward to the next step, it is not necessary to actually compute both forms of y_{i+1}. The equations given above can be summarized in tabular form as follows:

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{3}{4}$	0	$\frac{3}{4}$		
1	$\frac{2}{9}$	$\frac{3}{9}$	$\frac{4}{9}$	
	$\frac{2}{9}$	$\frac{3}{9}$	$\frac{4}{9}$	
	$-\frac{5}{72}$	$\frac{6}{72}$	$\frac{8}{72}$	$-\frac{9}{72}$

Runge-Kutta-Merson

Another Runge-Kutta type method that provides an error estimate is the Runge-Kutta-Merson method. It uses two fourth-order methods, each with five stages based on the same coefficients, but different weights, in order to obtain the new value of the solution, y. The formulas are

$$
\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + \frac{1}{3}h, y_i + \frac{1}{3}k_1) \\
k_3 &= hf(x_i + \frac{1}{3}h, y_i + \frac{1}{6}k_1 + \frac{1}{6}k_2) \\
k_4 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{8}k_1 + \frac{3}{8}k_3) \\
k_5 &= hf(x_i + h, y_i + \frac{1}{2}k_1 - \frac{3}{2}k_3 + 2k_4) \\
y_{i+1} &= y_i + \frac{1}{6}k_1 + \frac{2}{3}k_4 + \frac{1}{6}k_5 \\
e_{i+1} &= \frac{1}{30}(2k_1 - 9k_3 + 8k_4 - k_5)
\end{aligned}
$$

This error is the more-accurate result minus the less-accurate result, and can be interpreted as an estimate of the local error associated with the less-accurate formula.

$$
\begin{array}{c|ccccc}
0 & & & & & \\
\frac{1}{3} & \frac{1}{3} & & & & \\
\frac{1}{3} & \frac{1}{6} & \frac{1}{6} & & & \\
\frac{1}{2} & \frac{1}{8} & 0 & \frac{3}{8} & & \\
1 & \frac{1}{2} & 0 & -\frac{3}{2} & 2 & \\
\hline
 & \frac{1}{6} & 0 & 0 & \frac{2}{3} & \frac{1}{6} \\
 & \frac{1}{10} & 0 & \frac{3}{10} & \frac{2}{5} & \frac{1}{5} \\
- & - & - & - & - & - \\
 & -\frac{2}{30} & 0 & -\frac{9}{30} & \frac{8}{30} & -\frac{1}{30}
\end{array}
$$

A Simple RKF45

Most RKF methods use a combination of a fourth-order and a fifth-order method; we start with a simple example. The fourth-order method was presented in the previous section. The parameters for both methods are given in the table below (Ralston and Rabinowitz, pp. 218, 219). The lines below the dashed line are the weights for the fourth-order method, the weights for the fifth-order method, and the weights for the error estimate (fourth-order result minus the fifth-order result).

0						
$\frac{1}{2}$	$\frac{1}{2}$					
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$				
1	0	-1	2			
$\frac{2}{3}$	$\frac{7}{27}$	$\frac{10}{27}$	0	$\frac{1}{27}$		
$\frac{1}{5}$	$\frac{28}{625}$	$-\frac{1}{5}$	$\frac{546}{625}$	$\frac{54}{625}$	$-\frac{378}{625}$	
	$\frac{1}{6}$	0	$\frac{2}{3}$	$\frac{1}{6}$		
	$\frac{1}{24}$	0	0	$\frac{5}{48}$	$\frac{27}{56}$	$\frac{125}{336}$
—	$\frac{1}{8}$	0	$\frac{2}{3}$	$\frac{1}{16}$	$-\frac{27}{56}$	$-\frac{125}{336}$

The error estimate T_n is found by subtracting the value of y computed by the fifth-order method from the value computed using the fourth-order method. Sometimes the results of the fourth-order computation are used for the value of y_{n+1}. In other settings, the results of the higher-order method are used as the computed value of y_{n+1} (it may be viewed as a local extrapolation); in either case, it is not necessary to actually compute the estimate that is not to be used. The error estimate is computed from the weights in the last line of the table above.

$$T_n = \frac{1}{8}k_1 + \frac{2}{3}k_3 + \frac{1}{16}k_4 - \frac{27}{56}k_5 - \frac{125}{336}k_6$$

Classic Runge-Kutta-Fehlberg Method

The following parameters give the classic Runge-Kutta-Fehlberg method.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$
—	—	—	—	—	—	—
	$\frac{1}{360}$	0	$-\frac{128}{4275}$	$-\frac{2197}{75240}$	$\frac{1}{50}$	$\frac{2}{55}$

The estimated error is the value computed from the fifth-order method minus the value computed from the fourth-order method; it can be expressed directly as

$$\text{error} = \frac{1}{360}k_1 + 0k_2 + \frac{-128}{4275}k_3 + \frac{-2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6$$

(See Atkinson, 1989, p. 430, for further discussion.)

Note that, in general, one would not need to actually compute the fourth-order approximation, since the fifth-order approximation is more accurate, and the error estimate can be computed directly, as given above (Kincaid and Cheney, p. 586).

Since the fifth-order method requires six stages, five stages are used for the fourth-order method.

Runge-Kutta-Cash-Karp Method

An alternative set of coefficients (Cash and Karp, 1990) are recommended as giving a more efficient method with better error properties. The parameters for the fourth- and fifth-order methods are given in the following table (both methods use six stages).

0						
$\frac{1}{5}$	$\frac{1}{5}$					
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				
$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			
1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		
$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	
	$\frac{37}{378}$	0	$\frac{250}{621}$	$\frac{125}{594}$	0	$\frac{512}{1771}$
	$\frac{2825}{27648}$	0	$\frac{18575}{48384}$	$\frac{13525}{55296}$	$\frac{277}{14336}$	$\frac{1}{4}$

The weights for the fifth-order method are given in the first line below the horizontal line; the weights for the fourth-order method are given in the last line. The estimated error is the value computed from the fifth-order method minus the value computed from the fourth-order method (the coefficients for the error estimate are not shown explicitly). These give a more efficient method than the original, with better error properties. (See Press et al., 1992, p. 717.)

Adaptive Step-Size Runge-Kutta Methods

The error estimate from a Runge-Kutta-Fehlberg type method can be used to provide a criterion for adjusting the step size, in order to keep the error below a specified tolerance. To see how this could be done, let us return to the simple midpoint-Kutta method introduced at the beginning of this section. We found the estimate of the local truncation error L as

$$L = |Y - y| \approx |z - y|$$

where z was the solution computed by the third-order Kutta method, y was the solution computed by the second-order midpoint method, and Y is the exact solution. In addition, this local truncation error was also expressed as $L \approx B\,h_0^3$, where B depends on $f(x, y)$ and its derivatives and h_0 is the current step size. The estimate L can be used to control the step size in order to keep the local truncation error per unit step below a given tolerance, T; that is, we want $L/h_0 < T$ or

$$L \approx |z_{i+1} - y_{i+1}| \leq h_0 T$$

If this is satisfied, then the step is successful.

If the estimate of L does not satisfy the inequality, we seek a new step size, h_n so that we expect

$$L \approx B\,h_n^3 \leq h_n T$$

Using the fact that $B = |z_{i+1} - y_{i+1}|/h_0^3$, we have the relationship between the new step size h_n and the current step size h_0 given as

$$\frac{|z_{i+1} - y_{i+1}|}{h_0^3}\,h_n^2 \leq T$$

Solving for h_n gives

$$h_n^2 \approx \frac{h_0^3\,T}{|z_{i+1} - y_{i+1}|}$$

Since there are approximations involved in these calculations, an adjustment factor α is usually introduced for safety.

$$h_n = \alpha \frac{{h_0}^{3/2} T^{1/2}}{|z_{i+1} - y_{i+1}|^{1/2}}$$

If the pair of RK methods are of order p and $p + 1$, the step adjustment equation becomes

$$h_{\mathrm{n}} = \frac{\alpha h_0 (h_0\,T)^{1/p}}{|z_{i+1} - y_{i+1}|^{1/p}}$$

where z_{i+1} is the solution computed with the $(p+1)$-order method and y_{i+1} is the solution computed with the p-order method.

A typical value for the adjustment factor is $\alpha = 0.9$. Note that

$$L = |z_{i+1} - y_{i+1}|$$

can be computed directly from the k's without actually forming either z_{i+1} or y_{i+1}.

Adaptive RKF45

The general idea behind using the error estimate for step-size control is the following. If we have a pair of RK methods, of order p and $p+1$, the error scales as h^{p+1}, so if a step size h_1 produces an error D_1, and another step size h_0 produces an error D_0, the step sizes are related as

$$h_0 = h_1 |D_0/D_1|^{\frac{1}{(p+1)}}$$

We take D_0 to be the desired error tolerance, and reduce the step size when the computed error estimate is larger than the tolerance; conversely, we may increase the step size if the error estimate is significantly smaller than the tolerance.

If we have a fourth-order/fifth-order pair, and the error is Ch^5, we would expect doubling the step size to be safe as long as $C(2h)^5 < \text{tol}/4$; this means we could double the step size when the error estimate is less than tol/128. Requiring the estimate for the new error to be less than tol/4 is intended to prevent doubling that would lead to having to reduce the step size again (Kincaid and Cheney, p. 587).

As an algorithm, this becomes

```
If |error| > tol
    cut step size in half
    repeat the step
    If |error| < tol/128
        double step size and proceed
```

Press et al., 1992, give a very good description of a more sophisticated procedure. The essence is that if error > tol, the step size is reduced by a factor of

$$S|\text{tol}/\text{error}|^{0.25}$$

whereas, if error < tol, the step size is increased by the factor

$$S|\text{tol}/\text{error}|^{0.2}$$

This is a conservative approach which allows for the possibility of having the desired tolerance depend on the step size (a situation which requires the larger exponent). The factor S is a safety factor, slightly less than 1.

Thus, if we let $R = |\text{tol}/\text{error}|$, we have

$$h_n = \begin{cases} Sh_0R^{1/5}, & \text{if} \quad R \geq 1 \\ Sh_0R^{1/4}, & \text{if} \quad R < 1 \end{cases}$$

12.3 MULTISTEP METHODS

In this section, we continue our study of methods of solving the initial-value problem, $y' = f(x, y)$, $y(a) = y_0$, on the interval $[a, b]$ by finding an approximate solution at the node points, $x_0 = a, x_1 = a + h, \ldots, x_n = b$, where $h = (b - a)/n$.

Runge-Kutta methods use evaluations of the function $f(x, y)$ at one or more intermediate points between the current point, (x_i, y_i), and the point (x_{i+1}, y_{i+1}) where the solution is desired, in order to achieve higher order than Euler's method. Another approach is to use previous approximate solution values or evaluation of $f(x, y)$ at previous points; such methods are known as "multistep methods." In this section, we consider multistep methods (including one-step methods) that use evaluations of f only at the node points, i.e., not at any intermediate points between (x_i, y_i) and (x_{i+1}, y_{i+1}). To simplify the notation, we write $f_i = f(x_i, y_i)$.

A one-step method uses information at the current point (x_i, y_i) and the next point (x_{i+1}, y_{i+1}) to compute the new solution value y_{i+1}. The general form of a one-step method is

$$y_{i+1} = y_i + h[b_0 f_{i+1} + b_1 f_i]$$

A two-step method uses values of y and/or f at the current point (x_i, y_i) and the previous point (x_{i-1}, y_{i-1}) (as well as the next point (x_{i+1}, y_{i+1}):

$$y_{i+1} = a_1 y_i + a_2 y_{i-1} + h[b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1}]$$

In addition to the number of steps utilized, multistep methods are also distinguished according to whether the coefficient of the f_{i+1} term is zero. Multistep methods in which the coefficient of the f_{i+1} term is zero are known as *explicit methods*. If the coefficient is not zero, then the unknown y_{i+1} appears on the right-hand side of the equation. Such methods are called *implicit methods*. Implicit methods have better truncation error and better stability properties than explicit methods. However, it is usually not practical to solve directly for y_{i+1} using f_{i+1}, which involves the unknown y_{i+1}. For this reason, multistep methods are usually used in pairs, with an explicit method predicting the next value of y, which is then used in the right-hand side of the corresponding implicit method. The explicit method is a "predictor" and the implicit method is a "corrector."

Multistep methods require starting values, in addition to the initial condition specified for the differential equation. For a two-step method, y_1 must be found by some other method, such as a Runge-Kutta solution; for a k-step method, the first $k - 1$ values must be computed by another method.

There are a variety of methods for finding the coefficients that determine the method. Methods that use only one previous value of y can be found by using a numerical integration routine. The coefficients can also be found by requiring that the method give exact results for all polynomials of degree less than or equal to a specified value.

We first consider a few explicit methods and a few implicit methods before turning our attention to the predictor-corrector pairs, which are the primary focus of this section.

Trapezoid Method

As an example of a one-step method, we see that

$$y_{i+1} = y_i + h[\frac{1}{2}f_{i+1} + \frac{1}{2}f_i]$$

corresponds to

$$y_{i+1} = y_i + h[b_0 f_{i+1} + b_1 f_i]$$

with $b_0 = b_1 = 1/2$. This is an implicit one-step method that can be derived using the Newton-Cotes closed integration formula for the points (x_i, f_i) and (x_{i+1}, f_{i+1}), i.e., the trapezoid rule for integration.

Two-Step Methods

The general form of a two-step method is

$$y_{i+1} = a_1 y_i + a_2 y_{i-1} + h[b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1}]$$

Three-Step Methods

The general form of a three-step method is

$$y_{i+1} = a_1 y_i + a_2 y_{i-1} + a_3 y_{i-2} + h[b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1} + b_3 f_{i-1}]$$

General Multistep Methods

The general multistep method (k steps) has the form

$$y_{i+1} = \sum_{j=1}^{k} a_j y_{i-j+1} + h \sum_{j=0}^{k} b_j f_{i-j+1}$$

Adams Multistep Methods

The Adams multistep methods (k steps) have the form

$$y_{i+1} = y_i + h \sum_{j=0}^{k} b_j f_{i-j+1}$$

The explicit Adams methods, known as Adams-Bashforth methods, have $b_0 = 0$. The implicit Adams methods, known as Adams-Moulton methods, have $b_0 \neq 0$. Conditions for the coefficients are discussed in Sec. 12.4.2.

12.3.1 Adams-Bashforth Methods

Among the most popular explicit multistep methods are the Adams-Bashforth methods.

The general Adams-Bashforth method (k steps) has the form

$$y_{i+1} = y_i + h\sum_{j=1}^{k} b_j f_{i-j+1}$$

Adams methods are characterized by the fact that $a_1 = 1$ and all other $a_i = 0$. Adams-Bashforth methods are explicit, so $b_0 = 0$.

For Adams-Bashforth methods, the number of steps is the same as the order of the method. The local truncation error is one order higher than the overall order of the method. The computational procedures for the second-order and third-order Adams-Bashforth methods are summarized here.

Second-Order Adams-Bashforth Method

For the second-order Adams-Bashforth method:

y_0 is given by the initial condition for the differential equation.

y_1 is found from a one-step method, such as a Runge-Kutta technique.

Then, for $i = 1, \ldots, n-1$,

$$y_{i+1} = y_i + \frac{h}{2}[3f_i - f_{i-1}]$$

Comparing this formula with the general form of a two-step method,

$$y_{i+1} = a_1 y_i + a_2 y_{i-1} + h[b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1}]$$

we see that $a_1 = 1$, $a_2 = 0$, $b_0 = 0$, $b_1 = 3/2$, and $b_2 = -1/2$.

Third-Order Adams-Bashforth Method

For the third-order Adams-Bashforth method:

y_0 is given.

y_1 and y_2 are found from a one-step method.

Then, for $i = 2, \ldots, n-1$,

$$y_{i+1} = y_i + \frac{h}{12}[23f_i - 16f_{i-1} + 5f_{i-2}]$$

The primary use of the explicit Adams-Bashforth methods presented in this section is in conjunction with the implicit Adams-Moulton methods that we consider next. Especially for the higher-order Adams-Bashforth methods, stability requirements severely limit the step size for which the method gives reasonable results. The coefficients, and local truncation error terms, for several Adams-Bashforth methods are given in the following table.

Adams-Bashforth Methods

$$y_{i+1} = y_i + h\sum_{j=1}^{k} b_j f_{i-j+1}$$

Steps: k, order: $p = k$, local truncation error $C_{p+1}h^{p+1}f^{(p)}(\eta)$

p	k	b_1	b_2	b_3	b_4	b_5	b_6	C_{p+1}
2	2	$\frac{3}{2}$	$\frac{-1}{2}$					$\frac{5}{12}$
3	3	$\frac{23}{12}$	$\frac{-16}{12}$	$\frac{5}{12}$				$\frac{3}{8}$
4	4	$\frac{55}{24}$	$\frac{-59}{24}$	$\frac{37}{24}$	$\frac{-9}{24}$			$\frac{251}{720}$
5	5	$\frac{1901}{720}$	$\frac{-2774}{720}$	$\frac{2616}{720}$	$\frac{-1274}{720}$	$\frac{251}{720}$		$\frac{95}{288}$
6	6	$\frac{4277}{1440}$	$\frac{-7923}{1440}$	$\frac{9982}{1440}$	$\frac{-7298}{1440}$	$\frac{2877}{1440}$	$\frac{-475}{1440}$	$\frac{19087}{60480}$

The following MATLAB function for the third-order Adams-Bashforth method uses the RK3 optimal method to compute the first two steps. Evaluations of $f(x, y)$ which are used in successive steps are saved in the vector `z`. The interval on which the solution is desired is given in the input vector `tspan = [a, b]`. This function requires an inline function be defined for `f` prior to calling `AB3_RK3`. The use of the function is illustrated in the next example.

Third-Order Adams-Bashforth Method

```
function [x, y]  = AB3_RK3(f, y0, tspan, n)
% use inline function f(x,y)
a = tspan(1);       b = tspan(2);     h = (b-a) / n;      k = h/12;
x = (a : h : b);    y(1) = y0;
for i = 1:2              %  Use RK3 optimal method to start
    z(i) = f(x(i), y(i));
    k1 = h*z(i);
    k2 = h*f( x(i) + 0.5 *h, y(i) + 0.5 *k1 );
    k3 = h*f( x(i) + 0.75*h, y(i) + 0.75*k2 );
    y(i+1) = y(i) + (2*k1 + 3*k2 + 4*k3)/9;
end
for i = 3 : n            % Use AB3 method
    z(i) = f( x(i), y(i));
    y(i+1) = y(i) + k*(23*z(i) - 16*z(i-1) + 5*z(i-2));
end
```

EXAMPLE 12.7 Using the AB3 Method

The commands to use the preceding MATLAB function for the Adams-Bashforth three-step method to solve the simple ODE

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

are given in the script

```
% s_AB3_12_7
f = inline('x+y');
tspan = [ 0, 1];
n = 5;   ya = 2;
[x, y]  = AB3_RK3_opt_2(f, ya, tspan, n)
plot(x, y, '*');   grid on;  hold on
w = 3*exp(x) - x - 1;
plot(x,w);  hold off
error = w - y
```

The following table lists the computed values (y), the exact values (w), and the error $w - y$.

$x =$	0	0.2	0.4	0.6	0.8	1
$y =$	2.0000	2.4640	3.0750	3.8633	4.8696	6.1423
$w =$	2	2.4642	3.0755	3.8664	4.8766	6.1548
error =	0	0.0002	0.0005	0.0031	0.0070	0.0126

The computed values are graphed, together with the exact solution, $w = 3e^x - x - 1$, in Figure 12.9.

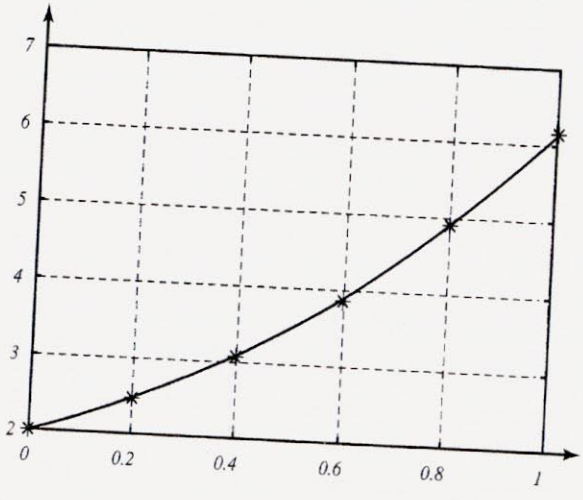

FIGURE 12.9: Solution to simple ODE using Adams-Bashforth third-order method.

12.3.2 Adams-Moulton Methods

Among the most popular implicit multistep methods are the Adams-Moulton methods. We first give the formulas for several of these. We then show how they are used together with the Adams-Bashforth methods in practice.

The coefficients of the error term are, in general, smaller for an implicit method than for the corresponding explicit method of the same order. Thus implicit methods have less round-off error than do explicit methods. The error terms are given in the table of Adams-Moulton methods below. An Adams-Moulton method (global truncation error) is one order higher than the number of steps involved in the method. As with other numerical methods for ODE, the local truncation error is one order higher than the global error.

Second-Order Adams-Moulton Method

The second-order Adams-Moulton method is a one-step method, so

y_0 is given by the initial condition for the differential equation.

Then, for $i = 0, \ldots, n-1$,

$$y_{i+1} = y_i + \frac{h}{2}[f_{i+1} + f_i]$$

Third-Order Adams-Moulton Method

The third-order Adams-Moulton method is a two-step method, so

y_0 is given; y_1 is found from a one-step method; then, for $i = 1, \ldots, n-1$,

$$y_{i+1} = y_i + \frac{h}{12}[5f_{i+1} + 8f_i - f_{i-1}]$$

Adams-Moulton Methods

$$y_{i+1} = y_i + h\sum_{j=0}^{k} b_j f_{i-j+1}$$

Steps k; order: $p = k + 1$, local truncation error $C_{p+1}h^{p+1}f^{(p)}(\eta)$

p	k	b_0	b_1	b_2	b_3	b_4	b_5	C_{p+1}
2	1	$\frac{1}{2}$	$\frac{1}{2}$					$\frac{-1}{12}$
3	2	$\frac{5}{12}$	$\frac{8}{12}$	$\frac{-1}{12}$				$\frac{-1}{24}$
4	3	$\frac{9}{24}$	$\frac{19}{24}$	$\frac{-5}{24}$	$\frac{1}{24}$			$\frac{-19}{720}$
5	4	$\frac{251}{720}$	$\frac{646}{720}$	$\frac{-264}{720}$	$\frac{106}{720}$	$\frac{-19}{720}$		$\frac{-3}{160}$
6	5	$\frac{475}{1440}$	$\frac{1427}{1440}$	$\frac{-798}{1440}$	$\frac{482}{1440}$	$\frac{-173}{1440}$	$\frac{27}{1440}$	$\frac{-863}{60480}$

12.3.3 Adams Predictor-Corrector Methods

In order to take advantage of the beneficial properties of the implicit multistep methods while avoiding the difficulties inherent in solving the implicit equation, an explicit and implicit method can be combined. The explicit method is used to predict a value of y_{i+1}, which we denote y^*_{i+1}. This value is then used in the right hand side of the implicit method, which produces an improved, or corrected, value of y_{i+1}. In addition, the error terms on the predictor and corrector can be used to find an estimate of the local error.

ABM2 Predictor-Corrector Method

As the simplest example, we use the second-order Adams-Bashforth method as a predictor with the second-order Adams-Moulton method as a corrector:

y_0 is given by the initial condition; y_1 is found from a one-step method. Then, for $i = 1, \ldots, n-1$,

$$y^*_{i+1} = y_i + \frac{h}{2}[3f_i - f_{i-1}]$$
$$y_{i+1} = y_i + \frac{h}{2}[f^*_{i+1} + f_i]$$

We now consider how the local truncation error terms for the predictor and corrector can be used to get a computable estimate of the local error. The predictor and corrector, with their local errors, are

$$y^*_{i+1} = y_i + \frac{h}{2}[3f_i - f_{i-1}] + \frac{5}{12}h^3y^{(3)}(\eta_i)$$
$$y_{i+1} = y_i + \frac{h}{2}[f^*_{i+1} + f_i] - \frac{1}{12}h^3y^{(3)}(\eta_i)$$

Let Y be the true solution, so that the error for the predictor is $Y - y^*$:

$$Y - y^* = \frac{5}{12}h^3y^{(3)}(\eta_1)$$

and the error for the corrector is $Y - y$:

$$Y - y = -\frac{1}{12}h^3y^{(3)}(\eta_2)$$

If we assume that h is small enough so that $y^{(3)} \approx C$ is nearly constant, we have $\frac{12}{5}(Y - y^*) \approx h^3C$ and $-12(Y - c) \approx h^3C$, so

$$\frac{12}{5}(Y - y^*) \approx -12(Y - c)$$
$$(Y - y^*) + 5(Y - y) \approx 0$$
$$6Y - 6y \approx -y + y^*$$

Thus, we find a computable estimate of the error, $Y - y$:

$$Y - y \approx -\frac{1}{6}(y - y^*)$$

ABM3 Predictor-Corrector Method

The ABM3 method uses the third-order Adams-Bashforth method as a predictor with the third-order Adams-Moulton method as a corrector:

y_0 given by the initial condition; y_1 and y_2 are found from a one-step method.
Then, for $i = 2, ..., n-1$,

$$y_{i+1}^* = y_i + \frac{h}{12}[23f_i - 16f_{i-1} + 5f_{i-2}]$$
$$y_{i+1} = y_i + \frac{h}{12}[5f_{i+1}^* + 8f_i - f_{i-1}]$$

With the error terms, we have

$$y_{i+1}^* = y_i + \frac{h}{12}[23f_i - 16f_{i-1} + 5f_{i-2}] + \frac{3}{8}h^4y^{(4)}(\eta)$$
$$y_{i+1} = y_i + \frac{h}{12}[5f_{i+1}^* + 8f_i - f_{i-1}] + \frac{-1}{24}h^4y^{(4)}(\eta)$$

If the true solution is Y, the predictor solution is y^* with error coefficient C_p, and the corrector solution is y with error coefficient C_c, then the estimate of the local truncation error for the corrected solution y is

$$Y - y = \frac{C_c}{C_p - C_c}(y - y^*)$$

For the ABM3 method we have $C_c = -1/24$ and $C_p = 3/8$, so

$$Y - y = \frac{-1}{10}(y - y^*)$$

ABM4 Predictor-Corrector Method

The ABM4 method combines the fourth-order Adams-Bashforth method as a predictor with the fourth-order Adams-Moulton method as a corrector:

y_0 is given; $y_1, \ldots, y_3$ are found from a one-step method.
Then, for $i = 3, \ldots, n-1$,

$$y_{i+1}^* = y_i + \frac{h}{24}[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}]$$
$$y_{i+1} = y_i + \frac{h}{24}[9f_{i+1}^* + 19f_i - 5f_{i-1} + f_{i-2}]$$

With the error terms, we have

$$y_{i+1}^* = y_i + \frac{h}{24}[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}] + \frac{251}{720}h^5y^{(5)}(\eta_1)$$
$$y_{i+1} = y_i + \frac{h}{24}[9f_{i+1}^* + 19f_i - 5f_{i-1} + f_{i-2}] - \frac{19}{720}h^5y^{(5)}(\eta_2)$$

Again, the estimate of the local truncation error for the corrected solution y is

$$Y - y = \frac{C_c}{C_p - C_c}(y - y^*)$$

For the ABM4 method, we have $Y - y = -\frac{19}{270}(y - y^*)$.

The following MATLAB function requires an inline function to specify $f(x, y)$.

Third-Order Predictor-Corrector Method

```
function [x, y]  = ABM3(f, y0, tspan, n)
% Adams_Bashforth_Moulton to solve y' = f(x,y)
% use inline function f(x,y)
a = tspan(1);     b = tspan(2);
h = (b-a) / n;    k = h/12;
x = (a : h : b);
y(1) = y0
%  Use RK3 optimal method to start
for i = 1:2
    z(i) = f(x(i), y(i));
    k1 = h*z(i);
    k2 = h*f( x(i) + 0.50*h, y(i) + 0.50*k1 );
    k3 = h*f( x(i) + 0.75*h, y(i) + 0.75*k2 );
    y(i+1) = y(i) + (2*k1 + 3*k2 + 4*k3)/9;
end
% Use third-order A-B-M method for remaining points
% yy is predicted value, y is corrected
for i = 3 : n
    z(i) = f( x(i), y(i));
    yy(i+1) = y(i) + k*(23*z(i) - 16*z(i-1) + 5*z(i-2));
    zz(i+1) = f(x(i+1), yy(i+1));
    y(i+1) = y(i) + k*(5*zz(i+1) +8*z(i) - z(i-1));
    error_est(i+1) = -0.1*(y(i+1) - yy(i+1));
end
error_est
```

The following script gives the commands to use this function for the example that follows.

```
% s_ABM3_12_8
f = inline('x+y');
tspan = [ 0, 1];
n = 5;     ya = 2;
[x, y]  = ABM3(f, ya, tspan, n)
plot(x, y, '*');   grid on;   hold on
w = 3*exp(x) - x - 1;
plot(x,w)
hold off
error = w - y
```

EXAMPLE 12.8 Using the ABM3 Predictor-Corrector Method

Consider again the differential equation

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

The approximate solution y was computed using the third-order Adams-Bashforth-Moulton predictor-corrector equations with the RK3 optimal method to start. Since the exact solution is $w = 3e^x - x - 1$, we can find the error $E = w - y$ at each value of x where the solution has been approximated. The error estimate computed from the difference between the corrected value and the predicated value is $E^* = -0.1(y - y^*)$. The same problem was worked in Example 12.7 without the corrector.

x	0	0.2	0.4	0.6	0.8	1.0
y	2	2.4640	3.0750	3.8658	4.8761	6.1544
w	2	2.4642	3.0755	3.8664	4.8766	6.1548
E	0	0.0002083	0.0005088	0.0005205	0.0005014	0.0004502
E^*	0	0	0	−0.0002534	−0.0003039	−0.0003736

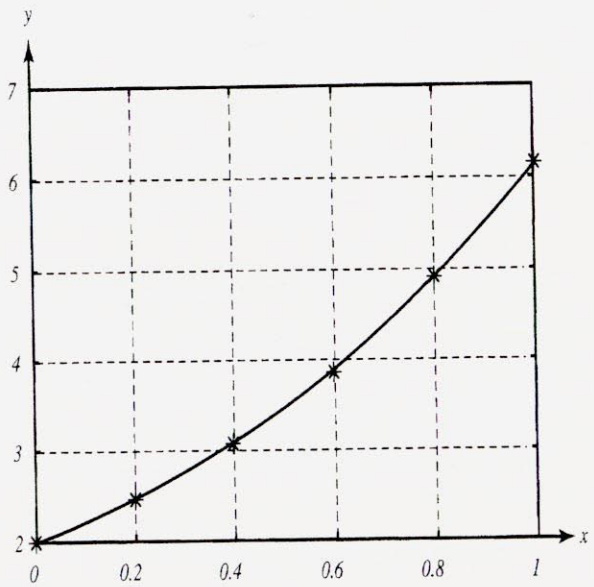

FIGURE 12.10: Solving $y' = x + y$ with the ABM3 method.

EXAMPLE 12.9 Velocity of Falling Parachutist

Consider the situation described in Application 12-A, in which the velocity of a parachutist with drag coefficient $k/m = 1.5$, $g = 32$, and $v_0 = 0$ is given by

$$m\frac{dv}{dt} = k(-v) - mg$$

The parachutist reaches a terminal velocity of approximately 21 ft/sec after only about 3 seconds.

If we modify the model slightly, so that

$$m\frac{dv}{dt} = k(-v)^{1.1} - mg$$

we find that the terminal velocity is somewhat slower. Figure 12.11 shows the velocity of the parachutist under both models of air resistance.

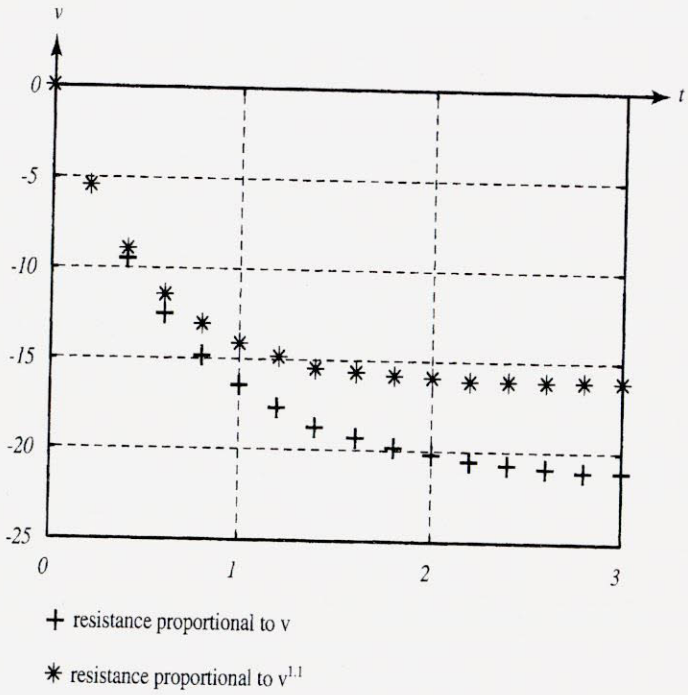

FIGURE 12.11: Velocity of parachutist with air resistance.

12.3.4 Other Predictor-Corrector Methods

Not all predictor-corrector methods are Adams methods. As an example, we consider the use of Simpson's method as a corrector together with Milne's method as the predictor.

Simpson's Method

As an example of a two-step method, we will find the coefficients for the implicit method given by

$$y_{i+1} = y_{i-1} + h[b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1}]$$

To illustrate the idea of undetermined coefficients, which can be used to find the coefficients for a wide variety of methods, we require that

$$\int_{x_{i-1}}^{x_{i+1}} f\,dt = b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1}$$

be exact for any f which is a polynomial of degree 2 or less. To simplify the computation, we shift the interval to $[-1, 1]$ and consider the polynomials $p_0 = 1$, $p_1 = t$, and $p_2 = t^2$. The equation becomes

$$\int_{-1}^{1} p(t)\,dt = b_0 p(1) + b_1 p(0) + b_2 p(-1)$$

For $p_0 = 1$, this gives

$$\int_{-1}^{1} 1\,dt = 2 = b_0 + b_1 + b_2$$

For $p_1 = t$, we have

$$\int_{-1}^{1} t\,dt = 0 = b_0(1) + b_1(0) + b_2(-1) = b_0 - b_2$$

and for $p_2 = t^2$

$$\int_{-1}^{1} t^2\,dt = 2/3 = b_0(1)^2 + b_1(0)^2 + b_2(-1)^2 = b_0 + b_2$$

Thus

$$\begin{aligned} 2 &= b_0 + b_1 + b_2 \\ 0 &= b_0 - b_2 \\ 2/3 &= b_0 + b_2 \end{aligned}$$

which gives $b_2 = 1/3$, $b_0 = 1/3$, $b_1 = 4/3$. This is not surprising, since the method uses the same points that Simpson's rule would use to compute the integral.

Thus, Simpson's implicit two-step method is given by

$$y_{i+1} = y_{i-1} + \frac{h}{3}[f_{i+1} + 4f_i + f_{i-1}]$$

Milne's Method

As an example of an explicit four-step method, consider Milne's method

$$y_{i+1} = y_{i-3} + \frac{4h}{3}(2f_i - f_{i-1} + 2f_{i-2}) + \frac{14}{45}h^5 y^{(n)}(\eta)$$

The derivation uses an open-type quadrature over the interval $[x_{i-3}, x_{i+1}]$.

$$y_{i+1} - y_{i-3} = \int_{x_{i-3}}^{x_{i+1}} f(t, y(t))dt$$

For open quadrature, the points at x_{i-3} and x_i are not used. To simplify the computation, we assume the nodes are at 0, 1, 2, 3, 4; using the Newton-Cotes open formula for $n = 3$ (Sec. 11.2.3) we have

$$y_{i+1} - y_{i-3} = \int_0^4 f(t, y(t))dt = \frac{4h}{3}(2f_1 - f_2 + 2f_3)$$

The general form for an explicit four-step method is

$$y_{i+1} = a_1 y_i + a_2 y_{i-1} + a_3 y_{i-2} + a_4 y_{i-3} = h(b_1 f_i + b_2 f_{i-1} + b_3 f_{i-2} + b_4 f_{i-3})$$

so we see that $a_1 = a_2 = a_3 = 0, a_4 = 1; b_1 = 8/3, b_2 = -4/3, b_3 = 8/3, b_4 = 0$.

Note that explicit methods which are found by quadrature must be found using open-quadrature rules so that the value of f_{i+1} does not appear on the right-hand side of the equation.

Milne-Simpson Predictor-Corrector Method

$$yp_{i+1} = y_{i-3} + \frac{4}{3}h(2f_i - f_{i-1} + f_{i-2}) + \frac{14}{45}h^5 y^{(5)}(\eta)$$

$$yc_{i+1} = y_{i-1} + \frac{1}{3}h(f_{i+1} + 4f_i + f_{i-1}) - \frac{1}{90}h^5 y^{(5)}(\eta)$$

The error expressions come from the open-quadrature rule and Simpson's rule; the open-quadrature rule has error $(14/45)h^5 f''''$; Simpson's rule has error $(-1/90)h^5 f''''$. If Y is the true solution,

then the local error for the predictor is $Y - y^* = \frac{28}{90}h^5 f''''$

and the error for the corrector is $Y - y = -\frac{1}{90}h^5 f''''$

If h is small enough that $f^{(4)}$ is nearly constant, we have

$$\frac{1}{28}(Y - y^*) \approx -(Y - y) \approx \frac{1}{90}f^{(4)}h^5$$

$$(Y - y^*) + 28(Y - y) \approx 0$$

$$29Y - y^* - 28y \approx 0$$

$$29Y - 29y \approx y^* - y$$

Solving for Y, we find

$$Y - y \approx -\frac{1}{29}(y - y^*)$$

This gives a computable estimate of the error.

12.4 FURTHER TOPICS

We begin this section by introducing some of MATLAB's built-in functions for solving ODE. We then consider the concepts of consistency and convergence for numerical methods for ODE. In the next chapter we consider stiff ODE, MATLAB functions designed for stiff ODE, and stability of numerical methods.

12.4.1 MATLAB's Methods

MATLAB includes three functions, `ode23`, `ode45`, and `ode113`, for solving non-stiff ODE. The function `ode23` implements a pair of explicit Runge-Kutta second- and third-order methods (the Bogacki and Shampine pair). According to the MATLAB documentation, it may be more efficient than `ode45` at crude tolerances and in the presence of moderate stiffness.

The function `ode45` implements the Dormand-Prince pair of fourth- and fifth-order methods. It is recommended as a "first try" for most problems.

The function `ode113` is a fully variable step-size ODE solver based on the Adams-Bashforth-Moulton family of formulas of orders 1–12. According to the MATLAB documentation, it may be more efficient than `ode45` at stringent tolerances and when the ODE file function is particularly expensive to evaluate.

The MATLAB functions `ode15s` and `ode23s` (discussed in the next chapter) are designed to solve stiff differential equations. The function `ode15s` implements a variable order method; `ode23s` is a low-order method. By default, the Jacobians needed by each method are generated numerically.

The syntax for the function call to each of the ODE solvers is the same; we illustrate it here for `ode23`. For that function, the basic call is

```
[t, y] = ode23('F', tspan, y0)
```

where the input is

- `'F'` is a string containing the name of an ODE file. The function `F(t, y)` must return a column vector of values.
- `tspan = [ t0  tf ]` is a vector of the initial and final times for the solution. To obtain solutions at specific times $t_0, t_1, ..., t_f$ (all increasing or all decreasing), use `tspan = [t0, t1, ..., tf ]`.
- `y0` is the vector of initial conditions.

The output consists of

- `t`, a vector of the times at which solution is computed.
- `y`, an array in which each row gives the solution at the time in the corresponding entry of vector `t`.

There are several variations for input and also for output. More details are available in the on-line help provided with each function.

12.4.2 Consistency and Convergence

One of the first requirements for a numerical method for solving an ODE is that the difference equation (DE) which is solved to approximate the solution must be consistent with the ODE. That means that the DE and the ODE must give the same answer for suitably simple problems.

To understand the consistency conditions, consider the approximate solution of

$$y' = f(x, y), \quad y(0) = 1$$

given by the three-step DE

$$y_{i+1} = a_1 y_i + h[b_0 f_{i+1} + b_1 f_i + b_2 f_{i-1}]$$

For the DE to be consistent with the ODE, we require that the DE give the same result as the ODE for $f(x, y) = 0$, $f(x, y) = 1$, $f(x, y) = x$, and $f(x, y) = x^2$.

The corresponding exact solutions are $y = 1$, $y = x + 1$, $y = (1/2)x^2 + 1$, and $y = (1/3)x^3 + 1$. This will give a method with local truncation error of order 4 and total truncation error of order 3 (the same as the degree of the highest polynomial $y(x)$ for which the method is exact).

To simplify the computations, take $h = 1$, so that we consider the DE

$$y_1 = a_1 y_0 + b_0 f_1 + b_1 f_0 + b_2 f_{-1}$$

Taking $f = 0$, so that the solution of $y' = f(x, y) = 0$ is $y = 1$, we have

$$1 = a_1(1) + b_0(0) + b_1(0) + b_2(0) \quad \Rightarrow \quad a_1 = 1$$

Taking $f = 1$, so that $y = x + 1$, and using the fact that $a_1 = 1$, we have

$$2 = 1 + b_0(1) + b_1(1) + b_2(1) \quad \Rightarrow \quad b_0 + b_1 + b_2 = 1$$

Taking $f = x$, so that the solution to $y' = f(x, y) = x$ is $y = (1/2)x^2 + 1$, we have

$$3/2 = 1 + b_0(1) + b_1(0) + b_2(-1) \quad \Rightarrow \quad b_0 - b_2 = 1/2$$

Taking $f = x^2$, so that the solution is $y = (1/3)x^3 + 1$, we have

$$1/3 + 1 = 1 + b_0(1^2) + b_1(0^2) + b_2(-1)^2 \quad \Rightarrow \quad b_0 + b_2 = 1/3$$

So we have

$$\begin{aligned} b_0 + b_1 + b_2 &= 1 \\ b_0 - b_2 &= 1/2 \\ b_0 + b_2 &= 1/3 \end{aligned}$$

Adding the second and third equations gives $b_0 = 5/12$, substituting this in the third equation gives $b_2 = -1/12$, and the first equation then yields $b_1 = 2/3$.

The generalization to a method with more steps, or with more previous values of y, is straightforward.

It can be shown that the consistency conditions for a general multistep method

$$y_{i+1} = \sum_{j=1}^{k} a_j y_{i-j+1} + h \sum_{j=0}^{k} b_j f_{i-j+1}$$

are

$$\sum_{j=1}^{k} a_j = 1$$

and

$$\sum_{j=0}^{k} b_j = \sum_{j=1}^{k} j a_j$$

To ensure that the method has order m, the additional conditions are that

$$r \sum_{j=0}^{k} (-j)^{r-1} b_j + \sum_{j=1}^{k} (-j)^r a_j$$

for $r = 2, \ldots, m$. These can be found by expanding all of the terms in the difference formula as Taylor series at x_{i+1} and requiring that the coefficients of the first m terms are zero. For $k = 3$, and a method of order 3, these equations are

$$\begin{aligned} a_1 + a_2 + a_3 &= 1 \\ b_0 + b_1 + b_2 + b_3 &= a_1 + 2a_2 + 3a_3 \\ 2(b_1 + 2b_2 + 3b_3) &= a_1 + 4a_2 + 9a_3 \\ 3(b_1 + 4b_2 + 9b_3) &= a_1 + 8a_2 + 27a_3 \end{aligned}$$

Atkinson indicates that in practice, one should take the order m of a k-step method to be less than or equal to $k+1$. The additional parameters can be chosen to improve stability, reduce truncation error, give an explicit method, and so on (see Atkinson, p. 381). Rice (p. 278) indicates that all multistep methods with $m > k+2$ (for m even) or $m > k+1$ (for m odd) are unstable.

The concept of convergence for a numerical method (a difference equation or DE) for solving an ODE deals with the behavior of the solution of the DE as the step size goes to zero. In particular, the multistep method

$$y_{i+1} = \sum_{j=1}^{p} a_j y_{i-j+1} + h \sum_{j=0}^{p} b_j f_{i-j+1}$$

is convergent if, as $h \to 0$, $y_n \to Y(x)$, where $Y(x)$ is the true solution of the ODE.

A necessary condition for the method to be convergent is that no root of

$$r^{p+1} - \sum_{i=1}^{p} a_i r^{p-i+1} = 0$$

lies outside the unit circle, and that roots of magnitude 1 are simple. (See Ralston and Rabinowitz, p. 176, for a proof.)

12.5 CHAPTER WRAP-UP

SUMMARY

Euler's Method

$y_{i+1} = y_i + hf_i$ order: 1, local truncation error: $E_T = (1/2)h^2y''(\eta_i)$

Midpoint Method

$y_{i+1} = y_{i-1} + 2hf_i$ order: 2, steps: 2, $E_T = (1/3)h^3y^{(3)}(\eta_i)$

Adams-Bashforth Methods

$y_{i+1} = y_i + h\sum_{j=1}^{k} b_j f_{i-j+1}$ order: p, steps: k, $E_T = C_{p+1}h^{p+1}f^{(p)}(\eta)$

p	k	b_1	b_2	b_3	b_4	b_5	b_6	C_{p+1}
2	2	$\frac{3}{2}$	$\frac{-1}{2}$					$\frac{5}{12}$
3	3	$\frac{23}{12}$	$\frac{-16}{12}$	$\frac{5}{12}$				$\frac{3}{8}$
4	4	$\frac{55}{24}$	$\frac{-59}{24}$	$\frac{37}{24}$	$\frac{-9}{24}$			$\frac{251}{720}$
5	5	$\frac{1901}{720}$	$\frac{-2774}{720}$	$\frac{2616}{720}$	$\frac{-1274}{720}$	$\frac{251}{720}$		$\frac{95}{288}$
6	6	$\frac{4277}{1440}$	$\frac{-7923}{1440}$	$\frac{9982}{1440}$	$\frac{-7298}{1440}$	$\frac{2877}{1440}$	$\frac{-475}{1440}$	$\frac{19087}{60480}$

Adams-Moulton Methods

$y_{i+1} = y_i + h\sum_{j=0}^{k} b_j f_{i-j+1}$ order: p, steps: k, $E_T = C_{p+1}h^{p+1}f^{(p)}(\eta)$

p	k	b_0	b_1	b_2	b_3	b_4	b_5	C_{p+1}
2	1	$\frac{1}{2}$	$\frac{1}{2}$					$\frac{-1}{12}$
3	2	$\frac{5}{12}$	$\frac{8}{12}$	$\frac{-1}{12}$				$\frac{-1}{24}$
4	3	$\frac{9}{24}$	$\frac{19}{24}$	$\frac{-5}{24}$	$\frac{1}{24}$			$\frac{-19}{720}$
5	4	$\frac{251}{720}$	$\frac{646}{720}$	$\frac{-264}{720}$	$\frac{106}{720}$	$\frac{-19}{720}$		$\frac{-3}{160}$
6	5	$\frac{475}{1440}$	$\frac{1427}{1440}$	$\frac{-798}{1440}$	$\frac{482}{1440}$	$\frac{-173}{1440}$	$\frac{27}{1440}$	$\frac{-863}{60480}$

Runge-Kutta Methods

Second-Order Methods

RK2: Midpoint

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$
$$y_{n+1} = y_n + k_2$$

RK2: Trapezoid

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + h, y_n + k_1)$$
$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

RK2: Optimal

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{2}{3}h, y_n + \frac{2}{3}k_1)$$
$$y_{n+1} = y_n + \frac{1}{4}(k_1 + 3k_2)$$

Third-Order Methods

RK3: Kutta's method

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$
$$k_3 = hf(x_n + h, y_n - k_1 + 2k_2)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 4k_2 + k_3)$$

RK3: Optimal method

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$
$$k_3 = hf(x_n + \frac{3}{4}h, y_n + \frac{3}{4}k_2)$$
$$y_{n+1} = y_n + \frac{1}{9}(2k_1 + 3k_2 + 4k_3)$$

RK3: Two-thirds rule

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{2}{3}h, y_n + \frac{2}{3}k_1)$$
$$k_3 = hf(x_n + \frac{2}{3}h, y_n + \frac{1}{3}k_1 + \frac{1}{3}k_2)$$
$$y_{n+1} = y_n + \frac{1}{4}(k_1 + 3k_3)$$

Fourth-Order Methods

Classic Runge-Kutta

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$
$$k_3 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$