Integration of Ordinary Differential Equations

Com S 477/577

Nov 7, 2002

1 Introduction

The solution of differential equations is an important problem that arises in a host of areas. Many differential equations are too difficult to solve in closed form. Instead, it becomes necessary to employ numerical techniques.

Differential equations have a major application in understanding physical systems that involve aerodynamics, fluid dynamics, thermodynamics, heat diffusion, mechanical oscillations, etc. They are used for developing control algorithms for dynamic simulations. Other applications also include optimization and stochastics.

We will consider ordinary differential equations (ODEs) and focus on two classes of problems: (a) first-order initial-value problems and (b) linear higher-order boundary-value problems. The basic principles we will see in these two classes of problems carry over to more general problems. For instance, higher-order initial-value problems can be rewritten in vector form to yield a set of simultaneous first-order equations. First-order techniques may then be used to solve this system.

Consider a general $n$th order differential equation in the form

$$y^{(n)}(x) = f \left( x, y(x), y'(x), \ldots, y^{(n-1)}(x) \right),$$

where $f$ is a function from $\mathbb{R}^n$ to $\mathbb{R}$. A general solution of this equation will usually contain $n$ arbitrary constants. A particular solution is obtained by specifying $n$ auxiliary constraints. For instance, one might specify the values of $y$ and its derivatives at some point $x = x_0$ as $y(x_0), y'(x_0), \ldots, y^{(n-1)}(x_0)$. Such a problem is called an initial-value problem. In effect, the auxiliary conditions specify all the relevant information at some starting point $x_0$, and the differential equation tells us how to proceed from that point.

If the $n$ auxiliary conditions are specified at different points, then the problem is called a boundary-value problem. In this case, we tie the function (or its derivatives) down at several points, and the differential equation tells us the shape of $y(x)$ between those points. For example, the following $n$ conditions on $y$ are specified at two points $x_0$ and $x_1$:

$$f_1 \left( y(x_0), \ldots, y^{(n-1)}(x_0), y(x_1), \ldots, y^{(n-1)}(x_1) \right) = 0,$$

$$\vdots$$

$$f_n \left( y(x_0), \ldots, y^{(n-1)}(x_0), y(x_1), \ldots, y^{(n-1)}(x_1) \right) = 0.$$
The $n$th order equation (1) can be transformed into a system of first order equations if we introduce $n - 1$ variables $z_i = y^{(i)}$, $i = 1, \ldots, n - 1$. This system consists of $n$ equations:

\[
\begin{align*}
y' &= z_1, \\
z_1' &= z_2, \\
&\vdots \\
z_{n-1}' &= f(x, y, z_1, \ldots, z_{n-1}).
\end{align*}
\]

We will focus on numerical techniques of solving first-order initial-value problems, assuming basic knowledge of analytics techniques. Numerical techniques are useful both for non-linear differential equations and for equations whose analytic solutions consist of complicated integrals and exponentials, etc.

Stated formally, the problem is to recover $y(x)$ from the differential equation

\[
\frac{dy}{dx} = f(x, y)
\]

with initial condition $y(x_0) = y_0$. We will introduce three types of numerical methods to solve the above equation.

2 Taylor’s Algorithm

If the exact solution $y(x)$ has a Taylor series expansion about $x_0$, then we can write

\[
y(x) = y_0 + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2!}y''(x_0) + \cdots
\]

Of course, if we do not know $y(x)$, then we do not explicitly know the values of its derivatives $y'(x_0), y''(x_0), \ldots$. However, if $f$ is sufficiently differentiable, then we can obtain the derivatives of $y$ from $f$:

\[
\begin{align*}
y'(x) &= f(x, y(x)), \\
y'' &= f_x + f_yy', \\
&= f_x + f_yf, \\
y''' &= f_{xx} + f_{xy}f + f_{yx}f + f_{yy}f^2 + f_yf_x + f_y^2f \\
&= f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_xf_y + f_y^2f,
\end{align*}
\]

(2)

From now on we will denote by $y(x_n)$ the true value of $y(x)$ at the point $x = x_n$, and by $y_n$ an approximate value. We will generate a series of points $x_0, x_1, \ldots$ by marching out from $x_0$. We will obtain $y_{n+1}$ from $y_n$ by using a truncated Taylor series, with approximate derivative values obtained from equations (2).

Suppose we choose a step size $h > 0$. Our points of interest are

\[
x_0, x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots
\]
Let us define a truncated Taylor series operator:

\[ T_k(x, y) = f(x, y) + \frac{h}{2!}f'(x, y) + \cdots + \frac{h^{k-1}}{k!}f^{(k-1)}(x, y), \quad \text{for each } k = 1, 2, \ldots, \]

where

\[ f^{(j)}(x, y) = \frac{d^j}{dx^j}f(x, y(x)) = y^{(j+1)}(x). \]

Taylor’s series tells us that

\[ y(x + h) = y(x) + hT_k(x, y) + r_k, \]

where \( r_k \) is a remainder that depends on \( y^{(k+1)} \) for each \( k \). This formula provides us with a numerical method for obtaining \( y_{n+1} \) from \( y_n \).

Taylor’s algorithm of order \( k \) finds an approximate solution to the differential equation

\[ y'(x) = f(x, y), \quad y(a) = y_0, \]

over the interval \( [a, b] \). It starts with discretizing \( [a, b] \) into \( N \) subintervals of equal length \( h = \frac{b-a}{N} \) and set \( x_n = a + nh, n = 0, 1, \ldots, N \). Then it carries out the following iteration:

\[ y_{n+1} \leftarrow y_n + hT_k(x_n, y_n), \quad \text{for } n = 0, 1, \ldots, N - 1. \]

The calculation of \( y_{n+1} \) uses information about \( y \) and its derivatives that comes from a single point, namely from \( x_n \). For this reason, Taylor’s algorithm is a one-step method.

Taylor’s algorithm of order 1, also known as Euler’s method, has the basic update rule:

\[ y_{n+1} \leftarrow y_n + hf(x_n, y_n). \]

Unfortunately, this method is not very accurate. It requires very small step sizes to obtain good accuracy. And there are stability problems (such as error accumulation). Nevertheless, the basic idea of adding small increments to previous estimates to obtain new estimates leads to more advanced methods.

3 Error Estimation

We would like to understand the quality of our differential equation solvers by estimating the error between \( y(x_n) \) and \( y_n \). There are three types of errors:

1. **Local discretization error** This is the error introduced in a single step of the equation solver as it moves from \( x_n \) to \( x_{n+1} \). In other words, it is the error in the estimate \( y_{n+1} \) that would result if \( y(x_n) \) were known perfectly.

2. **Full discretization error** This is the net error between \( y(x_n) \) and \( y_n \) at step \( n \). This error is the sum of the local discretization errors, plus any numerical roundoff errors.

3. **Numerical roundoff error** Limited machine precision can introduce errors. This type of errors will be ignored by us.

In general, an algorithm is said to be of order \( k \) if its local discretization error is \( O(h^{k+1}) \), where \( h \) is the step size.
3.1 Local Discretization Error

Taylor’s theorem tells us that the local error for Taylor’s algorithm of order \( k \) is simply

\[
\frac{h^{k+1} f^{(k)}(\xi, y(\xi))}{(k+1)!},
\]
or equivalently,

\[
\frac{h^{(k+1)} y^{(k+1)}(\xi)}{(k+1)!},
\]

where \( \xi \) is some point in the interval \((x_n, x_n + h)\). Thus Taylor’s algorithm of order \( k \) is indeed of order \( k \). And Euler’s algorithm is of order 1.

3.2 Full Discretization Error

This type of error can be very difficult to estimate. Let us therefore illustrate the approach with Euler’s method. First, we define

\[
e_n = y(x_n) - y_n.
\]

Taylor’s theorem says that

\[
y(x_{n+1}) = y(x_n) + h y'(x_n) + \frac{h^2}{2} y''(\xi_n), \quad \text{for some } \xi_n \in (x_n, x_{n+1}).
\]

Recall that

\[
y_{n+1} = y_n + hf(x_n, y_n).
\]

Subtract this equation from the previous one:

\[
e_{n+1} = e_n + h \left( f\left(x_n, y(x_n)\right) - f(x_n, y_n)\right) + \frac{h^2}{2} y''(\xi_n).
\]

Now, we apply the mean-value theorem to \( f \) and obtain

\[
e_{n+1} = e_n + hf_y(x_n, \bar{y}_n) \left(y(x_n) - y_n\right) + \frac{h^2}{2} y''(\xi_n)
\]

\[
= e_n \left(1 + hf_y\left(x_n, \bar{y}_n\right)\right) + \frac{h^2}{2} y''(\xi_n),
\]

(3)

for some \( \bar{y}_n \) between \( y(x_n) \) and \( y_n \).

To solve the recurrence relation (3), we assume that \( f_y \) and \( y'' \) are both bounded on \([a, b]\), say,

\[
\left| f_y(x, y(x)) \right| \leq L,
\]

\[
\left| y''(x) \right| \leq Y,
\]

for \( x \in [a, b] \). Then it follows that

\[
|e_{n+1}| \leq (1 + hL)|e_n| + \frac{h^2}{2} Y.
\]

Since \( e_0 = 0 \) and \( 1 + hL > 1 \), we can prove by induction that \( |e_n| \leq \delta_n \), where \( \delta_n \) satisfies the recurrence

\[
\delta_{n+1} = (1 + hL)\delta_n + \frac{h^2}{2} Y.
\]
This recurrence has the solution
\[ \delta_n = \frac{hY}{2L} \left( (1 + hL)^n - 1 \right). \]

Consequently,
\[ |e_n| \leq \delta_n \leq \frac{hY}{2L} \left( e^{nhL} - 1 \right) = \frac{hY}{2L} \left( e^{(x_n-x_0)L} - 1 \right). \] (4)

From the above analysis we see that the full discretization error approaches zero as the step size \( h \) is made smaller. That the error is \( O(h) \) implies that the convergence may be slow.

Example 1. Consider the differential equation
\[ y' = -y^2, \quad \text{with} \quad y(1) = 1. \]

The exact solution is \( y(x) = \frac{1}{x} \).

Suppose we use Euler’s method to obtain a numerical solution over the interval \([1, 2]\). The step size we choose is \( h = 0.1 \). Euler’s method says
\[ y_{n+1} = y_n + hf(x_n, y_n), \quad n = 0, 1, \ldots. \]

In this problem \( y_0 = 1 \) and \( f(x_n, y_n) = -y_n^2 \).

First, we obtain the local error as
\[ |E| \leq \max_{1 \leq x \leq 2} \frac{h^2}{2} |y''(x)| = \max_{1 \leq x \leq 2} \frac{h^2}{2} \frac{2}{x^3} = 0.01. \]

To obtain full error, note that \( y''(x) = \frac{2}{x^3} \) and \( f_y(x, y(x)) = -2y = -\frac{2}{x} \). So we use
\[ Y = 2 \quad \text{and} \quad L = 2. \]

Therefore it follows from (4) that
\[ |e_n| \leq \frac{h}{2} \left( e^{2(x_n-1)} - 1 \right), \]
and
\[ \max_n |e_n| \leq \frac{h}{2} (e^2 - 1) \approx 0.3195. \]

So we should expect about 1 decimal digit of accuracy.
The following table compares the actual results obtained against those yielded from the exact solution.

<table>
<thead>
<tr>
<th>$x_n$</th>
<th>$y_n$</th>
<th>$f(x_n, y_n)$</th>
<th>$y(x_n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>−1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.1</td>
<td>0.9</td>
<td>−0.810</td>
<td>0.90909</td>
</tr>
<tr>
<td>1.2</td>
<td>0.819</td>
<td>−0.67076</td>
<td>0.83333</td>
</tr>
<tr>
<td>1.3</td>
<td>0.7519</td>
<td>−0.56539</td>
<td>0.76923</td>
</tr>
<tr>
<td>1.4</td>
<td>0.69539</td>
<td>−0.48356</td>
<td>0.71429</td>
</tr>
<tr>
<td>1.5</td>
<td>0.64703</td>
<td>−0.41865</td>
<td>0.66667</td>
</tr>
<tr>
<td>1.6</td>
<td>0.60516</td>
<td>−0.36622</td>
<td>0.625</td>
</tr>
<tr>
<td>1.7</td>
<td>0.56854</td>
<td>−0.32324</td>
<td>0.58824</td>
</tr>
<tr>
<td>1.8</td>
<td>0.53622</td>
<td>−0.28753</td>
<td>0.55556</td>
</tr>
<tr>
<td>1.9</td>
<td>0.50746</td>
<td>−0.25752</td>
<td>0.52632</td>
</tr>
<tr>
<td>2.0</td>
<td>0.48171</td>
<td>−0.23205</td>
<td>0.5</td>
</tr>
</tbody>
</table>

The maximum error is just under 0.02, well within our estimates. But it is still not small enough.

4 Runge-Kutta Methods

Euler’s method is unacceptable because it requires small step sizes. Higher-order Taylor algorithms are unacceptable because they require higher-order derivatives.

Runge-Kutta methods attempt to obtain greater accuracy than, say, Euler’s method, without explicitly evaluating higher order derivatives. These methods evaluate $f(x, y)$ at selected points in the interval $[x_n, x_{n+1}]$.

The idea is to combine the estimates of $y$ resulting from these selected evaluations in such a way that error terms of order $h, h^2, \ldots, \text{etc.}$, are canceled out, to the desired accuracy.

For instance, the Runge-Kutta method of order 2 tries to cancel terms of order $h$ and $h^2$, leaving a local discretization error of order $h^3$. Similarly, the Runge-Kutta method of order 4 tries to cancel terms of order $h, h^2, h^3, h^4$, leaving an $O(h^5)$ local discretization error.

5 Runge-Kutta Method of Order 2

We wish to evaluate $f(x, y)$ at two points in the interval $[x_n, x_{n+1}]$, then combine the results to obtain an error of order $h^3$. The basic step of the algorithm is

$$y_{n+1} = y_n + ak_1 + bk_2,$$

where

$$k_1 = hf(x_n, y_n),$$
$$k_2 = hf(x_n + ah, y_n + \beta k_1).$$

In the above, $a, b, \alpha,$ and $\beta$ are fixed constants. They will be chosen in such a way as to obtain the $O(h^3)$ local discretization error.

Intuitively, the basic step first evaluates $y'$ at $(x_n, y_n)$ by computing $y'_n = f(x_n, y_n)$. The algorithm then tentatively uses this approximation to $y'$ to step to the trial point $(x_n + ah, y_n + \beta k_1)$. At this trial point, the algorithm reevaluates the derivative $y'$, then uses both derivative estimates
to compute $y_{n+1}$. By using two points to compute $y'$, the algorithm should be obtaining second derivative information, hence be able to reduce the size of the discretization error.

Let us derive the constants $a, b, \alpha$ and $\beta$. First, we use Taylor’s expansion in $x$:

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \frac{h^3}{3!}y'''(x_n) + \cdots$$

$$= y(x_n) + hf(x_n, y_n) + \frac{h^2}{2}(f_x + f_fy)\|_{(x_n, y_n)}$$

$$+ \frac{h^3}{6}(f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_xf_y + f_y^2f)\|_{(x_n, y_n)} + O(h^4). \quad (5)$$

Meanwhile, Taylor’s expansion in two variables $x$ and $y$ says that

$$f(x_n + \alpha h, y_n + \beta k_1) = f(x_n, y_n) + \alpha hf_x + \beta k_1 f_y + \frac{\alpha^2 h^2}{2}f_{xx} + \frac{\beta^2 k_1^2}{2}f_{yy} + \alpha hf_{x} + O(h^3).$$

Subsequently, we have

$$y_{n+1} = y_n + ak_1 + bk_2$$

$$= y_n + (a + b)hf + bh^2(\alpha f_x + \beta f_y) + bh^3\left(\frac{\alpha^2}{2}f_{xx} + \beta^2 f_{yy} + \alpha \beta f_{xy}\right) + O(h^4). \quad (6)$$

We want $y_{n+1}$ to estimate $y(x_{n+1})$ with $O(h^3)$ local error. Let us equate the formulas (5) with (6) and match the terms in $h$ and $h^2$, respectively. This yields the following three constraint equations for $a, b, \alpha$, and $\beta$:

$$a + b = 1,$$

$$\alpha b = \frac{1}{2},$$

$$\beta b = \frac{1}{2}.$$ 

Note that for the purpose of local discretization error, $y(x_n)$ matches $y_n$. The three constraint equations on four constants gives us a family of second-order Runge-Kutta methods.

One popular method is to choose $a = b = \frac{1}{2}$ and $\alpha = \beta = 1$. This effectively places the “trial” point at $x_{n+1}$, then moves from $y_n$ to $y_{n+1}$ by averaging the derivatives computed at $x_n$ and $x_{n+1}$. Thus

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

with

$$k_1 = hf(x_n, y_n),$$

$$k_2 = hf(x_n + h, y_n + k_1).$$

The Runge-Kutta method of order 2 has a local discretization error $O(h^3)$. This is superior than Euler’s method, which has local error $O(h^2)$. In order for a Taylor-based method to obtain $O(h^3)$ error, it must compute second derivatives. The Runge-Kutta method, however, does not need to compute second derivatives. Instead it performs two evaluations of the first derivative.
5.1 Runge-Kutta of Order 4

A widely used method is the following order 4 Runge-Kutta method. It produces $O(h^5)$ local discretization error, and thus leads to solutions quickly and accurately. The integration formula has the form

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

where

$$k_1 = hf(x_n, y_n),$$

$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right),$$

$$k_4 = hf(x_n + h, y_n + k_3).$$

The term $k_1$ estimates the derivative $y' = f(x, y)$ at the start point. The term $k_2$ estimates $y'$ at the midpoint between $x_n$ and $x_{n+1}$, using a $y$ value obtained with $k_1$. The term $k_3$ estimates $y'$ at the midpoint $\frac{x_n + x_{n+1}}{2}$, but with an updated $y$ value obtained by using $k_2$. The term $k_4$ estimates $y'$ at the point $x_{n+1}$, using a $y$ value obtained by advancing from $y_n$ by $k_3$.

In the above figure, the curve represents the actual solution to the differential equation $y'(x) = f(x, y)$. If $y(x_n) = y_n$ as in the picture, then we desire that $y_{n+1} = y(x_{n+1})$. The picture shows the process whereby $y_{n+1}$ is computed. Points are labelled by numbers in the order that they are computed. We assume that the vector field given by $y' = f(x, y)$ is continuous, and that other integral curves locally look “like” the one shown.

Example 2. Reconsider the differential equation in Example 1:

$$y' = -y^2 \quad \text{with} \quad y(1) = 1.$$ 

The table below shows the solution of this problem over the interval $[1, 2]$ using the fourth order Runge-Kutta
method with a step size \( h = 0.1 \).

<table>
<thead>
<tr>
<th>( x_n )</th>
<th>( y_n )</th>
<th>( f(x_n, y_n) )</th>
<th>( y(x_n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.1</td>
<td>0.90909</td>
<td>-0.82645</td>
<td>0.90909</td>
</tr>
<tr>
<td>1.2</td>
<td>0.83333</td>
<td>-0.69445</td>
<td>0.83333</td>
</tr>
<tr>
<td>1.3</td>
<td>0.76923</td>
<td>-0.59172</td>
<td>0.76923</td>
</tr>
<tr>
<td>1.4</td>
<td>0.71429</td>
<td>-0.51020</td>
<td>0.71429</td>
</tr>
<tr>
<td>1.5</td>
<td>0.66667</td>
<td>-0.44445</td>
<td>0.66667</td>
</tr>
<tr>
<td>1.6</td>
<td>0.62500</td>
<td>-0.39063</td>
<td>0.625</td>
</tr>
<tr>
<td>1.7</td>
<td>0.58824</td>
<td>-0.34001</td>
<td>0.58824</td>
</tr>
<tr>
<td>1.8</td>
<td>0.55556</td>
<td>-0.30864</td>
<td>0.55556</td>
</tr>
<tr>
<td>1.9</td>
<td>0.52632</td>
<td>-0.27701</td>
<td>0.52632</td>
</tr>
<tr>
<td>2.0</td>
<td>0.50000</td>
<td>-0.25000</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Compare this with the results of Euler’s method shown in Example 1. There is no error to the fifth decimal digits. In fact, the error was less than \( 5 \cdot 10^{-7} \).

## 6 Adams-Bashforth Method

Runge-Kutta propagates a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand \( f \)'s), and then using the information obtained to match a Taylor series expansion up to some higher order. It essentially makes several small steps for each major step from \( x_n \) to \( x_{n+1} \).

An alternative of moving from \( x_n \) to \( x_{n+1} \) is to make use of prior information at \( x_{n-1}, x_{n-2}, \ldots \). In other words, rather than evaluating \( f(x, y) \) at several intermediate points, make use of known values of \( f(x, y) \) at several past points. Methods that make use of information at several of the \( x \)'s are called multi-step methods. We will here look at one such method, known as the Adams-Bashforth method.

Suppose we have approximations to \( y(x) \) and \( y'(x) \) at the points \( x_0, \ldots, x_n \). If we integrate the differential equation

\[
y'(x) = f(x, y(x))
\]

from \( x_n \) to \( x_{n+1} \), we obtain

\[
\int_{x_n}^{x_{n+1}} y'(x) \, dx = \int_{x_n}^{x_{n+1}} f(x, y(x)) \, dx.
\]

Hence

\[
y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y(x)) \, dx.
\]

How do we evaluate the integral? The trick is to approximate \( f(x, y(x)) \) with an interpolating polynomial. Of course, we do not know exact values of \( f(x, y(x)) \) anywhere except at \( x_0 \). However, we do have approximate values \( f(x_k, y_k) \) for \( k = 0, 1, \ldots, n \). So we will construct an interpolating polynomial that passes through some of those values.

Specifically, if we are interested in an order \( m + 1 \) method, we will approximate the function \( f(x, y(x)) \) with an interpolating polynomial that passes through the \( m + 1 \) points \( (x_{n-m}, f_{n-m}), \ldots, (x_n, f_n) \). Here \( f_i = f(x_i, y_i) \). We then integrate this polynomial in order to approximate the integral \( \int_{x_n}^{x_{n+1}} f(x, y(x)) \, dx \).
We have seen interpolating polynomials written in terms of divided differences. It is also possible to write interpolating polynomials in terms of forward differences. Forward differences are defined recursively as

$$\Delta^i f_k = \begin{cases} f_k & \text{if } i = 0; \\ \Delta^{i-1} f_{k+1} - \Delta^{i-1} f_k & \text{if } i > 0. \end{cases}$$

A forward difference table looks like this:

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\Delta^0$</th>
<th>$\Delta^1$</th>
<th>$\Delta^2$</th>
<th>$\Delta^3$</th>
<th>$\Delta^4$</th>
<th>$\Delta^5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$f_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>$f_1$</td>
<td>$\Delta^2 f_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>$f_2$</td>
<td>$\Delta^2 f_1$</td>
<td>$\Delta^3 f_0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>$f_3$</td>
<td>$\Delta^2 f_2$</td>
<td>$\Delta^3 f_1$</td>
<td>$\Delta^4 f_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_4$</td>
<td>$f_4$</td>
<td>$\Delta^2 f_3$</td>
<td></td>
<td>$\Delta^3 f_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_5$</td>
<td>$f_5$</td>
<td></td>
<td></td>
<td></td>
<td>$\Delta^4 f_0$</td>
<td></td>
</tr>
</tbody>
</table>

One advantage of forward differences is that no division needs to be performed.

It turns out that one can write the order $m$ interpolating polynomial that passes through the points $(x_i, f_i), i = n - m, \ldots, n,$ as

$$p_m(x) = \sum_{k=0}^{m} (-1)^k \binom{-s}{k} \Delta^k f_{n-k},$$

where

$$s = \frac{x - x_n}{h},$$

$$\binom{-s}{k} = \frac{(-s)(-s-1) \cdots (-s-k+1)}{k}, \quad \text{if } k \geq 1.$$ 

Since $dx = h \, ds$, we obtain that

$$y_{n+1} \approx y_n + h \int_0^1 \sum_{k=0}^{m} (-1)^k \binom{-s}{k} \Delta^k f_{n-k} \, ds$$

$$= y_n + h \left\{ \gamma_0 f_n + \gamma_1 \Delta f_{n-1} + \cdots + \gamma_m \Delta^m f_{n-m} \right\} \quad (7)$$

where

$$\gamma_k = (-1)^k \int_0^1 \binom{-s}{k} \, ds.$$ 

The sequence $\{\gamma_k\}$ are precomputable numbers. The simplest case, obtained by setting $m = 0$ in (7), leads to Euler’s method.
One popular Adams-Bashforth method is the one of order 4. To derive it, we need to compute \( \gamma_i, \ i = 0, 1, 2, 3: \)

\[
\begin{align*}
\gamma_0 &= 1, \\
\gamma_1 &= -\int_0^1 (-s) \, ds = \frac{1}{2}, \\
\gamma_2 &= \int_0^1 \frac{(-s)(-s-1)}{2} \, ds = \frac{5}{12}, \\
\gamma_3 &= -\int_0^1 \frac{(-s)(-s-1)(-s-2)}{6} \, ds = \frac{3}{8}.
\end{align*}
\]

So

\[
y_{n+1} = y_n + h \left( f_n + \frac{1}{2} \Delta f_{n-1} + \frac{5}{12} \Delta^2 f_{n-2} + \frac{3}{8} \Delta^3 f_{n-3} \right).
\]

From the definition of the forward-difference operator \( \Delta \) we derive that

\[
\begin{align*}
\Delta f_{n-1} &= f_n - f_{n-1}, \\
\Delta^2 f_{n-2} &= f_n - 2f_{n-1} + f_{n-2}, \\
\Delta^3 f_{n-3} &= f_n - 3f_{n-1} + 3f_{n-2} - f_{n-3}.
\end{align*}
\]

Substituting the above in (8) and regrouping, we obtain the update formula:

\[
y_{n+1} = y_n + \frac{h}{24} \left( 55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3} \right).
\]

The local error can be determined as

\[
E = \frac{251}{120} h^5 y^{(5)}(\xi), \quad \text{for some } \xi \in (x_{n-3}, x_{n+1}).
\]

**Example 3.** Let us return to the example \( y' = -y^2 \) with \( y(1) = 1 \) to be solved over the interval \([1, 2]\).

In order to start off the order 4 Adams-Bashforth method, we need to know the values of \( f(x, y(x)) \) at the four points \( x_0, x_1, x_2, x_3 \). In general, one obtains these either by knowing them, or by running a highly accurate method that computes approximations to \( y(x_1), y(x_2), \) and \( y(x_3) \). In our case, we will take the values obtained by the Runge-Kutta method in Example 2.

<table>
<thead>
<tr>
<th>( x_n )</th>
<th>( y_n )</th>
<th>( f(x_n, y_n) )</th>
<th>( y(x_n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.1</td>
<td>0.90909</td>
<td>-0.82645</td>
<td>0.90909</td>
</tr>
<tr>
<td>1.2</td>
<td>0.83333</td>
<td>-0.69445</td>
<td>0.83333</td>
</tr>
<tr>
<td>1.3</td>
<td>0.76923</td>
<td>-0.59172</td>
<td>0.76923</td>
</tr>
<tr>
<td>1.4</td>
<td>0.71444</td>
<td>-0.51042</td>
<td>0.71429</td>
</tr>
<tr>
<td>1.5</td>
<td>0.66666</td>
<td>-0.44444</td>
<td>0.66667</td>
</tr>
<tr>
<td>1.6</td>
<td>0.62525</td>
<td>-0.39093</td>
<td>0.625</td>
</tr>
<tr>
<td>1.7</td>
<td>0.58848</td>
<td>-0.34631</td>
<td>0.58824</td>
</tr>
<tr>
<td>1.8</td>
<td>0.55556</td>
<td>-0.30892</td>
<td>0.55556</td>
</tr>
<tr>
<td>1.9</td>
<td>0.52655</td>
<td>-0.27726</td>
<td>0.52632</td>
</tr>
<tr>
<td>2.0</td>
<td>0.50000</td>
<td>-0.25052</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Notice that the maximum error is approximately $2.5 \cdot 10^{-4}$. This is indeed within the bound we obtained for the local error, namely

$$|E| \leq \max_{1 \leq x \leq 2} \frac{251}{750} h^5 |y^{(5)}(x)|$$

$$= \max_{1 \leq x \leq 2} \frac{251}{720} \left(\frac{1}{3}\right)^5 \frac{120}{x^6}$$

$$\leq 4.2 \cdot 10^{-4}.$$  

Both the Runge-Kutta method and the Adams-Bashforth method are methods of order 4, meaning that their local discretization error is $O(h^5)$. However, the constant coefficient hidden in the error term tends to be higher for the Adams-Bashforth method than for the Runge-Kutta method. Consequently, the Runge-Kutta method tends to exhibit greater accuracy. This is shown in Example 3. Adams-Bashforth also has the drawback that it is not “self-starting” — one must supply four initial data points rather than just one.

Both methods require evaluation of $f(x, y(x))$ at four points in order to move from $x_n$ to $x_{n+1}$. However, while Runge-kutta must generate three intermediate points at which to evaluate $f(x, y(x))$, Adams-Bashforth uses information already available. Consequently, Adams-Bashforth requires less computation and is therefore faster.

Multi-step methods use the basic formula:

$$y_{n+1} = y_{n-p} + h \int_{x_n-p}^{x_n+1} \sum_{k=0}^{m} (-1)^k \binom{s}{k} \Delta^k f_{n-k} \, ds.$$  

The integration is from $x_{n-p}$ to $x_{n+1}$ using interpolation at the points $x_{n-m}, \ldots, x_n$. The case $p = 0$ yields Adams-Bashford. Some especially interesting formulas of this type are

$$y_{n+1} = y_{n-1} + 2hf_n, \quad E = \frac{h^3}{6} y'''(\xi), \quad \text{when } m = 1 \text{ and } p = 1.$$  

and

$$y_{n+1} = y_{n-3} + \frac{4h}{3} (2f_n - f_{n-1} + 2f_{n-2}), \quad E = \frac{14}{45} h^5 y^{(5)}(\xi), \quad \text{when } m = 3 \text{ and } p = 3.$$  

Other important techniques and topics of the initial-value problems include

- Adaptive control of the step size.
- Multi-step formulas that integrate over several intervals.
- Predictor-corrector methods (implicit methods).
- Extrapolation to the limit.
- Numerical instability.
- Stiff equations (higher order equations or systems of equations whose solutions have different time scales.)
A Two Theorems about ODEs

It can be shown that the initial value problem involving the differential equation $y' = f(x, y)$ has exactly one solution provided that the function $f$ satisfies a few simple regular conditions.

**Theorem 1** Let $f$ be defined and continuous on the strip $S = \{ (x, y) \mid a \leq x \leq b, y \in \mathbb{R}^n \}$. Furthermore, let there be a constant $L$ such that
\[
\|f(x, y_1) - f(x, y_2)\| \leq L\|y_1 - y_2\| \quad \text{(Lipschitz condition)}
\]
for all $x \in [a, b]$ and all $y_1, y_2 \in \mathbb{R}^n$. Then for every $x_0 \in [a, b]$ and every $y_0 \in \mathbb{R}^n$ there exists exactly one function $y(x)$ such that

(a) $y(x)$ is continuously differentiable on $[a, b]$;
(b) $y'(x) = f(x, y(x))$ for $x \in [a, b]$;
(c) $y(x_0) = y_0$.

The second theorem states that the solution of an initial value problem depends continuously on the initial value.

**Theorem 2** Let $f$ be defined and continuous on the strip $S = \{ (x, y) \mid a \leq x \leq b, y \in \mathbb{R}^n \}$ and satisfy the Lipschitz condition
\[
\|f(x, y_1) - f(x, y_2)\| \leq L\|y_1 - y_2\|, \quad \text{some constant } L,
\]
for all $(x, y_i) \in S, i = 1, 2$. Let $a \leq x_0 \leq b$. Then for the solution of $y(x; s)$ of the initial value problem
\[
y' = f(x, y), \quad y(x_0; s) = s
\]
there holds the estimate
\[
\|y(x; s_1) - y(x; s_2)\| \leq e^{L|x-x_0|}\|s_1 - s_2\|, \quad \text{for } a \leq x \leq b.
\]

References

