

## 5 Differential Equations

Our ultimate goal is to solve very general nonlinear partial differential equations of elliptic, hyperbolic, parabolic, or mixed type. However, a variety of basic techniques are required from the solutions of ordinary differential equations. By understanding the basic ideas for computationally solving initial and boundary value problems for differential equations, we can solve more complicated partial differential equations. The development of numerical solution techniques for initial and boundary value problems originates from the simple concept of the Taylor expansion. Thus the building blocks for scientific computing are rooted in concepts from freshman calculus. Implementation, however, often requires ingenuity, insight, and clever application of the basic principles. In some sense, our numerical solution techniques reverse our understanding of calculus. Whereas calculus teaches us to take a limit in order to define a derivative or integral, in numerical computations we take the derivative or integral of the governing equation and go backwards to define it as the difference.

### 5.1 Initial value problems: Euler, Runge-Kutta and Adams methods

The solutions of general partial differential equations rely heavily on the techniques developed for ordinary differential equations. Thus we begin by considering systems of differential equations of the form

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t) \quad (5.1.1)$$

where  $\mathbf{y}$  represents a vector and the initial conditions are given by

$$\mathbf{y}(0) = \mathbf{y}_0 \quad (5.1.2)$$

with  $t \in [0, T]$ . Although very simple in appearance, this equation cannot be solved analytically in general. Of course, there are certain cases for which the problem can be solved analytically, but it will generally be important to rely on numerical solutions for insight. For an overview of analytic techniques, see Boyce and DiPrima [1].

The simplest algorithm for solving this system of differential equations is known as the *Euler method*. The Euler method is derived by making use of the definition of the derivative:

$$\frac{d\mathbf{y}}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Delta \mathbf{y}}{\Delta t}. \quad (5.1.3)$$

Thus over a time span  $\Delta t = t_{n+1} - t_n$  we can approximate the original differential equation by

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t) \quad \Rightarrow \quad \frac{\mathbf{y}_{n+1} - \mathbf{y}_n}{\Delta t} \approx f(\mathbf{y}_n, t_n). \quad (5.1.4)$$

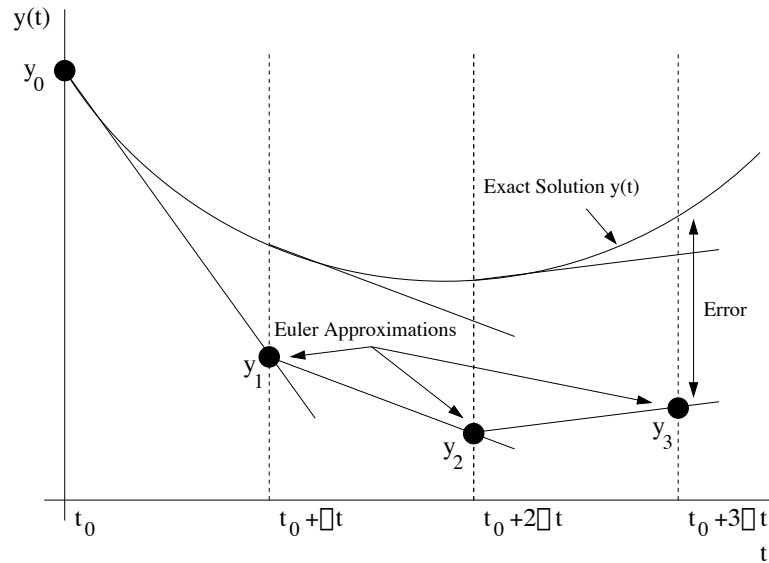


Figure 1: Graphical description of the iteration process used in the Euler method. Note that each subsequent approximation is generated from the slope of the previous point. This graphical illustration suggests that smaller steps  $\Delta t$  should be more accurate.

The approximation can easily be rearranged to give

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot f(\mathbf{y}_n, t_n). \quad (5.1.5)$$

Thus the Euler method gives an iterative scheme by which the future values of the solution can be determined. Generally, the algorithm structure is of the form

$$\mathbf{y}(t_{n+1}) = F(\mathbf{y}(t_n)) \quad (5.1.6)$$

where  $F(\mathbf{y}(t_n)) = \mathbf{y}(t_n) + \Delta t \cdot f(\mathbf{y}(t_n), t_n)$ . The graphical representation of this iterative process is illustrated in Fig. 1 where the slope (derivative) of the function is responsible for generating each subsequent approximation to the solution  $\mathbf{y}(t)$ . Note that the Euler method is exact as the step size decreases to zero:  $\Delta t \rightarrow 0$ .

The Euler method can be generalized to the following iterative scheme:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot \phi. \quad (5.1.7)$$

where the function  $\phi$  is chosen to reduce the error over a single time step  $\Delta t$  and  $\mathbf{y}_n = \mathbf{y}(t_n)$ . The function  $\phi$  is no longer constrained, as in the Euler scheme, to make use of the derivative at the left end point of the computational step.

Rather, the derivative at the mid-point of the time-step and at the right end of the time-step may also be used to possibly improve accuracy. In particular, by generalizing to include the slope at the left and right ends of the time-step  $\Delta t$ , we can generate an iteration scheme of the following form:

$$\mathbf{y}(t + \Delta t) = \mathbf{y}(t) + \Delta t [Af(t, \mathbf{y}(t)) + Bf(t + P \cdot \Delta t, \mathbf{y}(t) + Q\Delta t \cdot f(t, \mathbf{y}(t)))] \quad (5.1.8)$$

where  $A, B, P$  and  $Q$  are arbitrary constants. Upon Taylor expanding the last term, we find

$$f(t + P \cdot \Delta t, \mathbf{y}(t) + Q\Delta t \cdot f(t, \mathbf{y}(t))) = f(t, \mathbf{y}(t)) + P\Delta t \cdot f_t(t, \mathbf{y}(t)) + Q\Delta t \cdot f_y(t, \mathbf{y}(t)) \cdot f(t, \mathbf{y}(t)) + O(\Delta t^2) \quad (5.1.9)$$

where  $f_t$  and  $f_y$  denote differentiation with respect to  $t$  and  $\mathbf{y}$  respectively, use has been made of (5.1.1), and  $O(\Delta t^2)$  denotes all terms that are of size  $\Delta t^2$  and smaller. Plugging in this last result into the original iteration scheme (5.1.8) results in the following:

$$\begin{aligned} \mathbf{y}(t + \Delta t) = & \mathbf{y}(t) + \Delta t(A + B)f(t, \mathbf{y}(t)) \\ & + PB\Delta t^2 \cdot f_t(t, \mathbf{y}(t)) + BQ\Delta t^2 \cdot f_y(t, \mathbf{y}(t)) \cdot f(t, \mathbf{y}(t)) + O(\Delta t^3) \end{aligned} \quad (5.1.10)$$

which is valid up to  $O(\Delta t^2)$ .

To proceed further, we simply note that the Taylor expansion for  $\mathbf{y}(t + \Delta t)$  gives:

$$\begin{aligned} \mathbf{y}(t + \Delta t) = & \mathbf{y}(t) + \Delta t \cdot f(t, \mathbf{y}(t)) + \frac{1}{2}\Delta t^2 \cdot f_t(t, \mathbf{y}(t)) \\ & + \frac{1}{2}\Delta t^2 \cdot f_y(t, \mathbf{y}(t))f(t, \mathbf{y}(t)) + O(\Delta t^3). \end{aligned} \quad (5.1.11)$$

Comparing this Taylor expansion with (5.1.10) gives the following relations:

$$A + B = 1 \quad (5.1.12a)$$

$$PB = \frac{1}{2} \quad (5.1.12b)$$

$$BQ = \frac{1}{2} \quad (5.1.12c)$$

which yields three equations for the four unknowns  $A, B, P$  and  $Q$ . Thus one degree of freedom is granted, and a wide variety of schemes can be implemented. Two of the more commonly used schemes are known as *Heun's method* and *Modified Euler-Cauchy* (second order Runge-Kutta). These schemes assume  $A = 1/2$  and  $A = 0$  respectively, and are given by:

$$\mathbf{y}(t + \Delta t) = \mathbf{y}(t) + \frac{\Delta t}{2} [f(t, \mathbf{y}(t)) + f(t + \Delta t, \mathbf{y}(t) + \Delta t \cdot f(t, \mathbf{y}(t)))] \quad (5.1.13a)$$

$$\mathbf{y}(t + \Delta t) = \mathbf{y}(t) + \Delta t \cdot f\left(t + \frac{\Delta t}{2}, \mathbf{y}(t) + \frac{\Delta t}{2} \cdot f(t, \mathbf{y}(t))\right). \quad (5.1.13b)$$

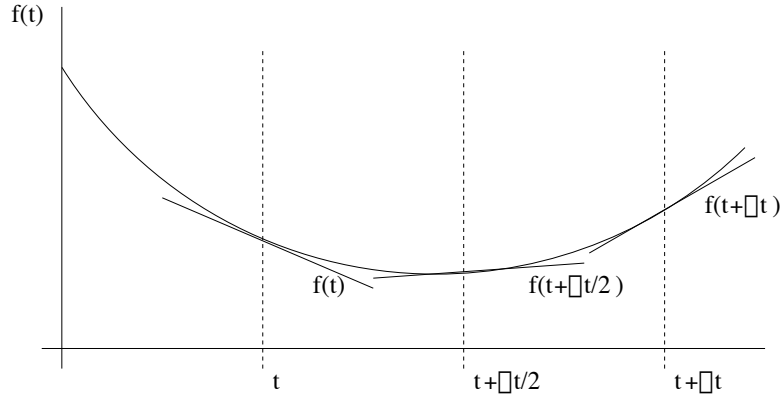


Figure 2: Graphical description of the initial, intermediate, and final slopes used in the 4th order Runge-Kutta iteration scheme over a time  $\Delta t$ .

Generally speaking, these methods for iterating forward in time given a single initial point are known as *Runge-Kutta methods*. By generalizing the assumption (5.1.8), we can construct stepping schemes which have arbitrary accuracy. Of course, the level of algebraic difficulty in deriving these higher accuracy schemes also increases significantly from Heun's method and Modified Euler-Cauchy.

#### 4th-order Runge-Kutta

Perhaps the most popular general stepping scheme used in practice is known as the *4th order Runge-Kutta method*. The term “4th order” refers to the fact that the Taylor series local truncation error is pushed to  $O(\Delta t^5)$ . The total cumulative (global) error is then  $O(\Delta t^4)$  and is responsible for the scheme name of “4th order”. The scheme is as follows:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{6} [f_1 + 2f_2 + 2f_3 + f_4] \quad (5.1.14)$$

where

$$f_1 = f(t_n, \mathbf{y}_n) \quad (5.1.15a)$$

$$f_2 = f\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2} f_1\right) \quad (5.1.15b)$$

$$f_3 = f\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2} f_2\right) \quad (5.1.15c)$$

$$f_4 = f(t_n + \Delta t, \mathbf{y}_n + \Delta t \cdot f_3). \quad (5.1.15d)$$

This scheme gives a local truncation error which is  $O(\Delta t^5)$ . The cumulative (global) error in this case is fourth order so that for  $t \sim O(1)$  then the error

is  $O(\Delta t^4)$ . The key to this method, as well as any of the other Runge-Kutta schemes, is the use of intermediate time-steps to improve accuracy. For the 4th order scheme presented here, a graphical representation of this derivative sampling at intermediate time-steps is shown in Fig. 2.

### Adams method: multi-stepping techniques

The development of the Runge-Kutta schemes rely on the definition of the derivative and Taylor expansions. Another approach to solving (5.1.1) is to start with the fundamental theorem of calculus [2]. Thus the differential equation can be integrated over a time-step  $\Delta t$  to give

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t) \quad \Rightarrow \quad \mathbf{y}(t + \Delta t) - \mathbf{y}(t) = \int_t^{t+\Delta t} f(t, y) dt. \quad (5.1.16)$$

And once again using our iteration notation we find

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} f(t, y) dt. \quad (5.1.17)$$

This iteration relation is simply a restatement of (5.1.7) with  $\Delta t \cdot \phi = \int_{t_n}^{t_{n+1}} f(t, y) dt$ . However, at this point, no approximations have been made and (5.1.17) is exact. The numerical solution will be found by approximating  $f(t, y) \approx p(t, y)$  where  $p(t, y)$  is a polynomial. Thus the iteration scheme in this instance will be given by

$$\mathbf{y}_{n+1} \approx \mathbf{y}_n + \int_{t_n}^{t_{n+1}} p(t, y) dt. \quad (5.1.18)$$

It only remains to determine the form of the polynomial to be used in the approximation.

The *Adams-Bashforth* suite of computational methods uses the current point and a determined number of past points to evaluate the future solution. As with the Runge-Kutta schemes, the order of accuracy is determined by the choice of  $\phi$ . In the Adams-Bashforth case, this relates directly to the choice of the polynomial approximation  $p(t, y)$ . A first-order scheme can easily be constructed by allowing

$$p_1(t) = \text{constant} = f(t_n, \mathbf{y}_n), \quad (5.1.19)$$

where the present point and no past points are used to determine the value of the polynomial. Inserting this first-order approximation into (5.1.18) results in the previously found Euler scheme

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot f(t_n, \mathbf{y}_n). \quad (5.1.20)$$

Alternatively, we could assume that the polynomial used both the current point and the previous point so that a second-order scheme resulted. The linear polynomial which passes through these two points is given by

$$p_2(t) = f_{n-1} + \frac{f_n - f_{n-1}}{\Delta t}(t - t_n). \quad (5.1.21)$$

When inserted into (5.1.18), this linear polynomial yields

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} \left( f_{n-1} + \frac{f_n - f_{n-1}}{\Delta t}(t - t_n) \right) dt. \quad (5.1.22)$$

Upon integration and evaluation at the upper and lower limits, we find the following 2nd order Adams-Bashforth scheme

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{2} [3f(t_n, \mathbf{y}_n) - f(t_{n-1}, \mathbf{y}_{n-1})]. \quad (5.1.23)$$

In contrast to the Runge-Kutta method, this is a *two-step algorithm* which requires two initial conditions. This technique can be easily generalized to include more past points and thus higher accuracy. However, as accuracy is increased, so are the number of initial conditions required to step forward one time-step  $\Delta t$ . Aside from the first-order accurate scheme, any implementation of Adams-Bashforth will require a *boot strap* to generate a second “initial condition” for the solution iteration process.

The Adams-Bashforth scheme uses current and past points to approximate the polynomial  $p(t, y)$  in (5.1.18). If instead a future point, the present, and the past is used, then the scheme is known as an *Adams-Moulton method*. As before, a first-order scheme can easily be constructed by allowing

$$p_1(t) = \text{constant} = f(t_{n+1}, \mathbf{y}_{n+1}), \quad (5.1.24)$$

where the future point and no past and present points are used to determine the value of the polynomial. Inserting this first-order approximation into (5.1.18) results in the *backward Euler scheme*

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot f(t_{n+1}, \mathbf{y}_{n+1}). \quad (5.1.25)$$

Alternatively, we could assume that the polynomial used both the future point and the current point so that a second-order scheme resulted. The linear polynomial which passes through these two points is given by

$$p_2(t) = f_n + \frac{f_{n+1} - f_n}{\Delta t}(t - t_n). \quad (5.1.26)$$

Inserted into (5.1.18), this linear polynomial yields

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} \left( f_n + \frac{f_{n+1} - f_n}{\Delta t}(t - t_n) \right) dt. \quad (5.1.27)$$

Upon integration and evaluation at the upper and lower limits, we find the following 2nd order Adams-Moulton scheme

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{2} [f(t_{n+1}, \mathbf{y}_{n+1}) + f(t_n, \mathbf{y}_n)]. \quad (5.1.28)$$

Once again this is a two-step algorithm. However, it is categorically different than the Adams-Bashforth methods since it results in an *implicit scheme*, i.e. the unknown value  $\mathbf{y}_{n+1}$  is specified through a nonlinear equation (5.1.28). The solution of this nonlinear system can be very difficult, thus making *explicit schemes* such as Runge-Kutta and Adams-Bashforth, which are simple iterations, more easily handled. However, implicit schemes can have advantages when considering stability issues related to time-stepping. This is explored further in the notes.

One way to circumvent the difficulties of the implicit stepping method while still making use of its power is to use a *Predictor-Corrector method*. This scheme draws on the power of both the Adams-Bashforth and Adams-Moulton schemes. In particular, the second order implicit scheme given by (5.1.28) requires the value of  $f(t_{n+1}, \mathbf{y}_{n+1})$  in the right hand side. If we can predict (approximate) this value, then we can use this predicted value to solve (5.1.28) explicitly. Thus we begin with a predictor step to estimate  $\mathbf{y}_{n+1}$  so that  $f(t_{n+1}, \mathbf{y}_{n+1})$  can be evaluated. We then insert this value into the right hand side of (5.1.28) and explicitly find the corrected value of  $\mathbf{y}_{n+1}$ . The second-order predictor-corrector steps are then as follows:

$$\text{predictor (Adams-Bashforth): } \mathbf{y}_{n+1}^P = \mathbf{y}_n + \frac{\Delta t}{2} [3f_n - f_{n-1}] \quad (5.1.29a)$$

$$\text{corrector (Adams-Moulton): } \mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{2} [f(t_{n+1}, \mathbf{y}_{n+1}^P) + f(t_n, \mathbf{y}_n)]. \quad (5.1.29b)$$

Thus the scheme utilizes both explicit and implicit time-stepping schemes without having to solve a system of nonlinear equations.

### Higher order differential equations

Thus far, we have considered systems of first order equations. Higher order differential equations can be put into this form and the methods outlined here can be applied. For example, consider the third-order, nonhomogeneous, differential equation

$$\frac{d^3 u}{dt^3} + u^2 \frac{du}{dt} + \cos t \cdot u = g(t). \quad (5.1.30)$$

By defining

$$y_1 = u \quad (5.1.31a)$$

$$y_2 = \frac{du}{dt} \quad (5.1.31b)$$

$$y_3 = \frac{d^2 u}{dt^2}, \quad (5.1.31c)$$

we find that  $dy_3/dt = d^3 u/dt^3$ . Using the original equation along with the definitions of  $y_i$  we find that

$$\frac{dy_1}{dt} = y_2 \quad (5.1.32a)$$

$$\frac{dy_2}{dt} = y_3 \quad (5.1.32b)$$

$$\frac{dy_3}{dt} = \frac{d^3 u}{dt^3} = -u^2 \frac{du}{dt} - \cos t \cdot u + g(t) = -y_1^2 y_2 - \cos t \cdot y_1 + g(t) \quad (5.1.32c)$$

which results in the original differential equation (5.1.1) considered previously

$$\frac{d\mathbf{y}}{dt} = \frac{d}{dt} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ -y_1^2 y_2 - \cos t \cdot y_1 + g(t) \end{pmatrix} = f(\mathbf{y}, t). \quad (5.1.33)$$

At this point, all the time-stepping techniques developed thus far can be applied to the problem. It is imperative to write any differential equation as a first-order system before solving it numerically with the time-stepping schemes developed here.

### MATLAB commands

The time-stepping schemes considered here are all available in the MATLAB suite of differential equation solvers. The following are a few of the most common solvers:

- **ode23**: second-order Runge-Kutta routine
- **ode45**: fourth-order Runge-Kutta routine
- **ode113**: variable order predictor-corrector routine
- **ode15s**: variable order Gear method for stiff problems [3, 4]

## 5.2 Error analysis for time-stepping routines

*Accuracy* and *stability* are fundamental to numerical analysis and are the key factors in evaluating any numerical integration technique. Therefore, it is essential to evaluate the accuracy and stability of the time-stepping schemes developed. Rarely does it occur that both accuracy and stability work in concert. In fact, they often are offsetting and work directly against each other. Thus a highly accurate scheme may compromise stability, whereas a low accuracy scheme may have excellent stability properties.