Education has produced a vast population able to read but unable to distinguish what is worth reading.

— G.M. TREVELYAN

Chapter 6

Ordinary differential equations - Initial value problems

In this chapter we develop algorithms for solving systems of linear and nonlinear ordinary differential equations of the *initial value type*. Such models arise in describing *lumped parameter, dynamic* models. Entire books (Lapidus & Seinfeld, 1971; Lambert, 1973) are devoted to the development of algorithms for such problems. We will develop only elementary concepts of *single* and *multistep* methods, *implicit* and *explicit* methods, and introduce concepts of numerical stability and *stiffness*.

General purpose routines such as LSODE that implement several of advanced features such as automatic step size control, error control *etc.* are available from NETLIB.

6.1 Model equations and initial conditions

Ordinary differential equations of the initial type are represented typically as a system of *first order* equations of the form,

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}, t) \tag{6.1}$$

$$\boldsymbol{y}(t = t_0) = \boldsymbol{y}_0 \tag{6.2}$$

where $\boldsymbol{y}(t)$ is a vector containing elements $\boldsymbol{y} = \{y_1(t), y_2(t), \cdots, y_n(t)\}$ and our objective is to construct an approximate representation of the function $\boldsymbol{y}(t)$ over some interval of interest $t \in [t_0, t_f]$ that satisfy the

initial conditions, given by another vector $\boldsymbol{y}_0 = \{y_{1,0}, y_{2,0}, \cdots, y_{n,0}\}$. If the functions $\boldsymbol{f}(\boldsymbol{y}, t)$ depend on t explicitly, then the equations are called *non-autonomous*; otherwise they are called an *autonomous* system of equations.

6.1.1 Higher order differential equations

A higher order differential equation (say of order n) can be converted into an equivalent system of (n) first order equations. Consider the equation,

$$a_n \frac{d^n\theta}{dt^n} + a_{n-1}\frac{d^{n-1}\theta}{dt^{n-1}} + \cdots a_1\frac{d\theta}{dt} + a_0\theta = b \tag{6.3}$$

subject to a set of n initial conditions at t_0 of the form,

$$\begin{aligned} \left.\frac{d^{n-1}\theta}{dt^{n-1}}\right|_{t_0} &= c_{n-1} \\ \left.\frac{d^{n-2}\theta}{dt^{n-2}}\right|_{t_0} &= c_{n-2} \\ \vdots &= \vdots \\ \left.\frac{d\theta}{dt}\right|_{t_0} &= c_1 \\ \theta|_{t_0} &= c_0 \end{aligned} \tag{6.4}$$

Since all of these conditions are given at t_0, this remains an *initial value problem*. Equation (6.3) can be recast into a system of n first order equations of the form (6.1) as follows. Let us define θ and all of its $(n-1)$ successive higher derivatives as

$$y_1(t) = \theta(t), \quad y_2(t) = \frac{d\theta}{dt}, \quad y_3(t) = \frac{d^2\theta}{dt^2}, \quad \cdots \quad y_n(t) = \frac{d^{n-1}\theta}{dt^{n-1}}$$

Then we have,

$$\begin{aligned} \frac{dy_1}{dt} &= y_2, \qquad y_1(t_0) = c_0 \\ \frac{dy_2}{dt} &= y_3, \qquad y_2(t_0) = c_1 \\ \vdots &= \vdots \\ \frac{dy_{n-1}}{dt} &= y_n, \qquad y_{n-1}(t_0) = c_{n-2} \\ \frac{dy_n}{dt} &= \frac{1}{a_n}[b - a_0y_1 - a_1y_2 - \cdots - a_{n-1}y_n],, \qquad y_n(t_0) = c_{n-1} \end{aligned} \tag{6.5}$$

where the last equation has been obtained from the n-th order equation (6.3). Also shown in equations (6.5), are the transformed initial conditions from equation (6.4) in terms of the new variable set $\boldsymbol{y}$.

Note that the coefficients $\{a_0, a_1, \cdots a_n, b\}$ in equation (6.3) can in general be nonlinear functions of θ and its derivatives. This nonlinearity will reflect in equations (6.5),since the coefficients $\{a_0, a_1, \cdots a_n, b\}$ will be functions of the transformed variables $\{y_1, y_2 \cdots y_n\}$.

6.2 Taylor series expansion

Consider the differential equation,

$$\frac{dy}{dt} = f(y), \qquad y(t_0) = y_0 \tag{6.6}$$

Our task is to construct a sequence $\{y_n | n = 0, 1, \cdots\}$ that represents an approximate solution to $y(t)$ at a discrete set of points $\{t_n | n = 0, 1, \cdots\}$. We can achieve this by constructing a Taylor series expansion for $y(t)$ around t_n with a step size of h as,

$$y(t_n + h) = y(t_n) + y'(t_n)h + y''(t_n)\frac{h^2}{2} + \cdots$$

Truncating after the linear term and recognizing that $y'(t_n) = f(y_n)$, we have the Euler scheme for generating the solution sequence,

$$\boxed{y_{n+1} = y_n + hf(y_n) + \underbrace{\mathcal{O}(h^2)}_{\textit{local error}} \qquad n = 0, 1, \cdots} \tag{6.7}$$

which is *single-step*, *explicit* scheme with a local truncation error of order $\mathcal{O}(h^2)$. It is called a *single-step* method because it requires only the value at y_n to predict the value at next step y_{n+1}. It is *explicit* because the right hand side terms $[y_n + hf(y_n)]$ can be computed explicitly using known value of y_n.

6.2.1 Alternate derivation using interpolation polynomials

Rewriting the differential equation (6.6) as,

$$\int_{y_n}^{y_{n+1}} dy = \int_{t_n}^{t_{n+1}} f(y)dt \tag{6.8}$$

and using Newton forward and backward interpolating polynomials to approximate the function $f(y)$ we can recover, not only the Euler scheme, but develop a mechanism for obtaining a whole class of *implicit* and *multistep* methods. First let us use the m-th degree Newton forward polynomial from equation (5.11), viz.

$$f(y) \approx P_m(t_n + \alpha h) = \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2!}\Delta^2 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-m+1)}{(m)!}\Delta^m \right] f_n + \mathcal{O}(h^{m+1})$$

where t_n has been used as the reference point, f_n means $f(y_n)$ and h is the step size. Since $t = t_n + \alpha h$ we have $dt = h d\alpha$. Using a one term expansion in equation (6.8), (*i.e.*, $m = 0$) results in,

$$\begin{aligned} y_{n+1} - y_n &= \int_{t_n}^{t_{n+1}} P_0(t_n + \alpha h) dt + \underbrace{\int_{t_n}^{t_{n+1}} \mathcal{O}(h) dt}_{\textit{local truncation error}} \\ &= \int_0^1 P_0(t_n + \alpha h)\ h d\alpha + \int_0^1 \mathcal{O}(h) h d\alpha \\ &= \int_0^1 f_n\ h d\alpha + \mathcal{O}(h^2) \\ &= f_n\ h + \mathcal{O}(h^2) \end{aligned}$$

which is the same equation as (6.7). This approach, however, lends itself naturally to further development of higher order methods. For example a two-term expansion (*i.e.*, $m = 1$) results in,

$$\begin{aligned} y_{n+1} - y_n &= \int_{t_n}^{t_{n+1}} P_1(t_n + \alpha h) dt + \underbrace{\int_{t_n}^{t_{n+1}} \mathcal{O}(h^2) dt}_{\textit{local truncation error}} \\ &= \int_0^1 P_1(t_n + \alpha h)\ h d\alpha + \int_0^1 \mathcal{O}(h^2) h d\alpha \\ &= \int_0^1 [f_n + \alpha\Delta f_n]\ h d\alpha + \mathcal{O}(h^3) \\ &= h\left[\alpha\ f_n + \Delta f_n \frac{\alpha^2}{2}\right]_0^1 + \mathcal{O}(h^3) \\ &= h\left[f_n + (f_{n+1} - f_n)\frac{1}{2}\right] + \mathcal{O}(h^3). \end{aligned}$$

Hence we have the final form of the modified Euler scheme as,

$$\boxed{y_{n+1} = y_n + \frac{h}{2}\left[f_n + f_{n+1}\right] + \underbrace{\mathcal{O}(h^3)}_{\textit{local error}} \qquad n = 0, 1, \cdots} \tag{6.9}$$

Both the Euler method given in equation (6.7) and the modified Euler scheme given by equation (6.9) are *single-step* methods since only y_n is required to predict y_{n+1}. The modified Euler method is an *implicit* scheme since we need to compute f_{n+1} which depends on y_{n+1}. Note that implicit schemes requires the solution of a nonlinear algebraic equation at every time step. Thus to calculate y_{n+1} from equation (6.9) we need to use an *iterative method* that involves providing an initial guess for y_{n+1} and using equation (6.9) as a fixed point iteration scheme until y_{n+1} converges to desired accuracy. At a first glance, this might appear to be a disadvantage of the *implicit* schemes. However, implicit schemes have the ability to anticipate sharp changes in the solution between y_n and y_{n+1} and hence are suitable (in fact required) for solving the so called *stiff differential equations*.

This initial guess could be provided by the Euler method (viz. equation (6.7)). When an explicit scheme is combined with an implicit scheme in this manner, we have the so called *predictor-corrector* scheme. The Euler and modified Euler predictor-corrector pair is,

$$\boxed{y^P_{n+1} = y_n + hf(y_n)} \qquad \text{and} \qquad \boxed{y^C_{n+1} = y_n + \frac{h}{2}\left[f(y_n) + f(y^P_{n+1})\right]} \tag{6.10}$$

where the superscript P represents the predicted value from an explicit scheme and C represents the corrected value from an implicit scheme.

It should be clear that extending the Newton forward polynomial to a three-term expansion will not be fruitful, since that would involve not only f_{n+1}, but also f_{n+2}. We can, however, use Newton backward polynomials to develop higher order methods as will be done in section §6.3. But, let us explore first the reason for and the circumstances under which *implicit* schemes are useful.

6.2.2 Stability limits

Let us consider a model, linear equation,

$$\frac{dy}{dt} = \lambda\, y, \qquad y(t = 0) = 1$$

which has the analytical solution,

$$y(t) = e^{\lambda t}$$

For $\lambda < 0$ the exact solution decreases monotonically to zero as $t \to \infty$. Let us examine the sequence $\{y_n | n = 0, 1, \cdots\}$ generated by the explicit, Euler scheme and the implicit, modified Euler scheme. Note that in this model problem the function $f(y) = \lambda y$. The Euler equation is,

$$y_{n+1} = y_n + hf(y_n) = y_n + h\lambda y_n = [1 + h\lambda]y_n$$

Thus the sequence is,

$$\begin{aligned} y_1 &= [1 + h\lambda]y_0 \\ y_2 &= [1 + h\lambda]y_1 = [1 + h\lambda]^2 y_0 \\ y_3 &= [1 + h\lambda]y_2 = [1 + h\lambda]^3 y_0 \\ \vdots &= \vdots \end{aligned}$$

leading to the general solution,

$$y_n = [1 + h\lambda]^n y_0$$

When the step size h is chosen to be too large (more specifically $|h\lambda| > 2$ in this case), the sequence will diverge, while the exact solution remains bounded. This phenomenon is called *numerical instability* caused by the discretization. Explicit methods in general have such a stability bound on the step size h.

Let us examine the behavior of an implicit scheme - viz. the modified Euler scheme.

$$y_{n+1} = y_n + \frac{h}{2}[f_n + f_{n+1}] = y_n + \frac{h}{2}[\lambda y_n + \lambda y_{n+1}]$$

Note that y_{n+1} appears on both sides. Solving for y_{n+1} we get,

$$y_{n+1} = \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right] y_n$$

Thus the sequence is,

$$\begin{aligned} y_1 &= \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right] y_0 \\ y_2 &= \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right] y_1 = \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right]^2 y_0 \\ y_3 &= \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right] y_2 = \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right]^3 y_0 \\ \vdots &= \vdots \end{aligned}$$

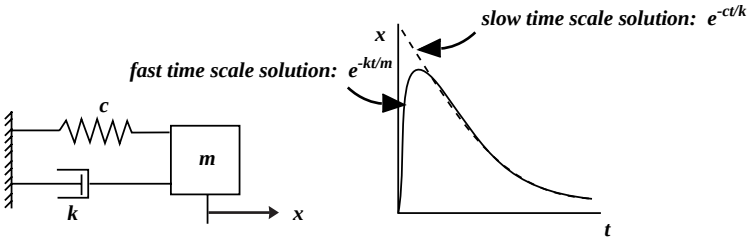

Figure 6.1: Spring and dash pot model

leading to the general solution,

$$y_n = \left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right]^n y_0$$

It is clear that for $\lambda < 0$, the ratio $\left[\frac{1+h\lambda/2}{1-h\lambda/2}\right] < 1$ for any choice of step size h. Thus the *implicit* scheme is *absolutely stable*. Hence, for explicit schemes, the choice of h is governed by both *stability* and *truncation error* considerations while for *implicit* schemes only *truncation error* considerations dictate the choice of step size, h.

6.2.3 Stiff differential equations

The physical interpretation for λ in the above model problem is that it represents the *characteristic time scale* of the problem. For a second order equation (or equivalently a system of two first-order equations), there will be two such time scales λ_1 and λ_2. If the time scales are widely separated in magnitude then we have a stiff system of differential equations. Consider the spring and dash pot model shown in figure 6.1. The displacement x is modelled by the force balance equation,

$$m\frac{d^2x}{dt^2} + k\frac{dx}{dt} + cx = 0$$

where c is the spring constant, k is the damping factor, m is the mass and x is the displacement. Let us assume that it is subject to the initial conditions $x(t = 0) = 0$ and $x'(t = 0) = constant$. We can write the characteristic equation as,

$$\frac{m}{k}\lambda^2 + \lambda + \frac{c}{k} = 0$$

and hence the two roots are given by,

$$\lambda = \frac{-1 \pm \sqrt{1 - 4mc/k^2}}{(2m/k)}$$

In the limit of $m \to 0$ we can approximate these as,

$$\lambda_1 = -\frac{k}{m} \qquad \text{and} \qquad \lambda_2 = -\frac{c}{k}$$

where L'Hopitals rule is used to obtain the second root. Clearly as $m \to 0$, we have $\lambda_1 >> \lambda_2$ and this limit corresponds to the stiff behavior of the solution. In general the *stiffness ratio* is defined as the ratio of the largest to the smallest eigenvalues. In this example the stiffness ratio is (k^2/mc) and it becomes large as m is made small. The solution satisfying the first initial condition is,

$$x(t) = A_1[\underbrace{e^{-kt/m}}_{fast} - \underbrace{e^{ct/k}}_{slow}]$$

where the fast and slow response terms are as shown. The sketch in figure 6.1 also shows the fast and slow response solutions. Note that if $m = 0$, the order of the differential equation drops by one and λ_1 is the only time scale for the problem. This kind of phenomena also occurs in a number of chemical reaction systems, where some of the reactions can occur on a rapid time scale while others take place on a longer time scale. The ozone decomposition model discussed in section §1.4.2 is another example of stiff differential equations.

It should now be clear that stiffness phenomena corresponds to large eigenvalues and fast response regions where the solution changes rapidly. In a system of n first order equations there will be n characteristic roots or eigenvalues. If λ_{max} is the largest eigenvalue, then explicit schemes will typically have a numerical stability limit of the form $|h\lambda_{max}| < constant$. Hence explicit schemes require that extremely small step size h be used in regions where the system responds very rapidly; otherwise the integration sequence will diverge. Implicit schemes that are absolutely stable have no such restrictions. The integration sequence using implicit schemes will remain bounded. The choice of step size is determined only by the desired accuracy of the solution. Stability analysis for a variety of explicit and implicit methods are discussed in greater detail by Lapidus and Seinfeld (1971).

6.3 Multistep methods

6.3.1 Explicit schemes

Consider approximating the function $f(y)$ in equation (6.8) by the following m-th degree Newton backward polynomial from equation (5.13),

$$f(y) \approx P_m(t_n + \alpha h) = \left[1 + \alpha\nabla + \frac{\alpha(\alpha+1)}{2!}\nabla^2 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha+1)(\alpha+2)\cdots(\alpha+m-1)}{m!}\nabla^m \right] f_n + \mathcal{O}(h^{m+1})$$

Here, t_n has been used as the reference point. Since this polynomial involves only points at earlier times such as $\{f_n, f_{n-1}, f_{n-2}\cdots\}$, we can develop a class of *explicit* schemes of high orders. These are called Adams-Bashforth schemes. Consider a three-term expansion (*i.e.*, $m = 2$). Equation (6.8) becomes,

$$\begin{aligned}
y_{n+1} - y_n &= \int_{t_n}^{t_{n+1}} P_2(t_n + \alpha h)dt + \underbrace{\int_{t_n}^{t_{n+1}} \mathcal{O}(h^3)dt}_{\textit{local truncation error}} \\
&= \int_0^1 P_2(t_n + \alpha h)\ h d\alpha + \int_0^1 \mathcal{O}(h^3) h d\alpha \\
&= \int_0^1 \left[f_n + \alpha\nabla f_n + \frac{\alpha(\alpha+1)}{2!}\nabla^2 f_n\right]\ h d\alpha + \mathcal{O}(h^4) \\
&= h\left[\alpha\ f_n + \frac{\alpha^2}{2}\ \nabla f_n + \frac{1}{2!}\ \left\{\frac{\alpha^3}{3} + \frac{\alpha^2}{2}\right\}\ \nabla^2 f_n\right]_0^1 + \mathcal{O}(h^4) \\
&= h\left[f_n + \frac{1}{2}(f_n - f_{n-1}) + \frac{5}{12}(f_n - 2f_{n-1} + f_{n-2})\right] + \mathcal{O}(h^4).
\end{aligned}$$

which can be rearranged into the form,

$$y_{n+1} = y_n + \frac{h}{12}\left[23\ f_n - 16\ f_{n-1} + 5\ f_{n-2}\right] + \underbrace{\mathcal{O}(h^4)}_{\textit{local error}} \qquad n = 2, 3, 4, \cdots \tag{6.11}$$

The following points should be observed on the above equation.

- This is a *multistep scheme* since it requires (y_n, y_{n-1}, y_{n-2}) to predict y_{n+1}.
- Hence it is not a self-starter! Typically, in a well posed initial value problem, we know only y_0. Hence y_1 and y_2 must be generated

from other single-step methods before we can switch to the above multistep scheme.

- It is an *explicit* method as y_{n+1} does not appear on the right hand side.
- As a consequence it cannot anticipate sharp changes in $y(t)$ - *i.e.*, not suitable for *stiff differential equations*.
- Makes good use of previous calculations to give low truncation error.
- requires only one function evaluation per step.

The next higher order scheme can be developed from a four-term expansion (*i.e.*, $m = 3$) . This is called 5-th order Adams-Bashforth scheme. viz.

$$\begin{aligned} y_{n+1} - y_n &= \int_{t_n}^{t_{n+1}} P_3(t_n + \alpha h)dt + \underbrace{\int_{t_n}^{t_{n+1}} \mathcal{O}(h^4)dt}_{\textit{local truncation error}} \\ &= \int_0^1 P_3(t_n + \alpha h)\ h d\alpha + \int_0^1 \mathcal{O}(h^4) h d\alpha \\ &= \int_0^1 \left[f_n + \alpha \nabla f_n + \frac{\alpha(\alpha+1)}{2!}\nabla^2 f_n + \frac{\alpha(\alpha+1)(\alpha+2)}{3!}\nabla^3 f_n \right] h d\alpha + \mathcal{O}(h^5) \\ &= h\left[f_n + \frac{1}{2}\ \nabla f_n + \frac{5}{12}\nabla^2 f_n + \frac{3}{8}\nabla^3 f_n \right] + \mathcal{O}(h^5) \end{aligned}$$

which can be rearranged into the form,

$$y_{n+1} = y_n + \frac{h}{24}\left[55\ f_n - 59\ f_{n-1} + 37\ f_{n-2} - 9 f_{n-3}\right] + \underbrace{\mathcal{O}(h^5)}_{\textit{local error}} \qquad n = 3, 4, \cdots \tag{6.12}$$

6.3.2 Implicit schemes

In order to construct implicit schemes we need to construct backward polynomial approximations with t_{n+1} as the reference point. viz.

$$\begin{aligned} f(y) \approx P_m(t_{n+1} + \alpha h) = \Bigg[1 + \alpha\nabla + \frac{\alpha(\alpha+1)}{2!}\nabla^2 + \cdots \\ \frac{\alpha(\alpha+1)(\alpha+2)\cdots(\alpha+m-1)}{m!}\nabla^m \Bigg] f_{n+1} + \mathcal{O}(h^{m+1}) \end{aligned}$$

In this manner f_{n+1} is introduced on the right hand side. This class of implicit schemes are called Adams-Moulton schemes. We are still integrating one step from t_n to t_{n+1}. Since $t = t_{n+1} + \alpha h$ and the limits of integration in α become $(-1, 0)$. A four-term expansion results in,

$$\begin{aligned} y_{n+1} - y_n &= \int_{t_n}^{t_{n+1}} P_3(t_{n+1} + \alpha h)dt + \underbrace{\int_{t_n}^{t_{n+1}} \mathcal{O}(h^4)dt}_{\textit{local truncation error}} \\ &= \int_{-1}^{0} P_3(t_n + \alpha h)\ h d\alpha + \int_{-1}^{0} \mathcal{O}(h^4) h d\alpha \\ &= \int_{-1}^{0} \left[f_{n+1} + \alpha \nabla f_{n+1} + \frac{\alpha(\alpha+1)}{2!}\nabla^2 f_{n+1} + \frac{\alpha(\alpha+1)(\alpha+2)}{3!}\nabla^3 f_{n+1} \right] h d\alpha + \mathcal{O}(h^5 \\ &= h\left[f_{n+1} - \frac{1}{2}\ \nabla f_{n+1} - \frac{1}{12}\nabla^2 f_{n+1} - \frac{1}{24}\nabla^3 f_{n+1} \right] + \mathcal{O}(h^5) \end{aligned}$$

which can be expanded and rearranged into the form,

$$y_{n+1} = y_n + \frac{h}{24}\left[9\ f_{n+1} + 19\ f_n - 5\ f_{n-1} + f_{n-2}\right] + \underbrace{\mathcal{O}(h^5)}_{\textit{local error}} \qquad n = 2, 4, \cdots \tag{6.13}$$

The pair of explicit-implicit schemes given by (6.12,6.13) respectively can be used as a *predictor-corrector* pair.

6.3.3 Automatic stepsize control

Some of the start up and step size control issues are illustrated in figure 6.2 for the 5-th order Adams schemes developed in the last section. Note that only y_0 is given and hence (y_1, y_2, y_3) must be generated using some other single step methods with a step size of h before the 5-th order Adams scheme given by equations (6.12,6.13) can be used. In doing so, it is important to realize that any error introduced in these three steps are likely to be propagated during the transient phase of the simulation. Hence if lower order schemes are used to generate (y_1, y_2, y_3), then smaller step sizes must be used. The difference between the predicted and corrected values could be used as a measure of the truncation error. If this error is below an acceptable tolerance, then we can choose to double the next step size. But this can begin only after y_6 has been computed, because we need four previous values at equal intervals of $(2h)$ - *i.e.*, (y_0, y_2, y_4, y_6). If at any time during the intergration process, the difference between the predicted and corrected values is above the

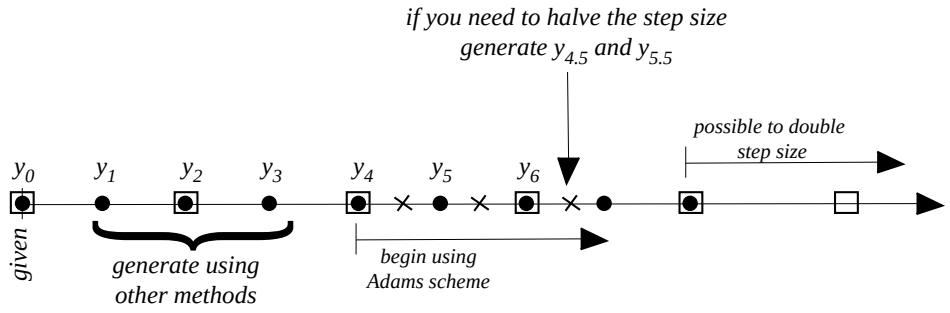

Figure 6.2: Stepsize control strategies for multistep methods

tolerance, then we must halve the step size and repeat the calculation for that step. In so doing, we need to generate intermediate values at intervals of $(h/2)$. For example if the result for y_7 does not meet the tolerance, then we repeat the calculation from y_6 with a step size of $h/2$. We need to generate intermediate values at $y_{4.5}$ and $y_{5.5}$. This can be done using the Newton backward interpolation polynomials; but the truncation errors in the interpolating polynomials should be of the same order and the Adams scheme. Specifically the interpolation rules are:

$$y_{n-\frac{1}{2}} = \frac{1}{128}[35y_n + 140y_{n-1} - 70y_{n-2} + 28y_{n-3} - 5y_{n-4}]$$
$$y_{n-\frac{3}{2}} = \frac{1}{64}[-y_n + 24y_{n-1} + 54y_{n-2} - 16y_{n-3} + 3y_{n-4}]$$

Example

MATLAB has several built-in functions for solving initial value problems. The functions named ADAMS and GEAR, use multistep methods. All of the ODE solvers in MATLAB are part of the SIMULINK toolbox. Hence the m-files that define the problem must have a special structure. In this section we illustrate how to use these functions to solve the ozone decomposition model. Recall that the equations are,

$$\frac{dy_1}{dt} = f_1(y_1, y_2) = -y_1 - y_1y_2 + \epsilon\kappa y_2$$
$$\frac{dy_2}{dt} = f_2(y_1, y_2) = (y_1 - y_1y_2 - \epsilon\kappa y_2)/\epsilon$$

The initial compositions are $\underline{y}(t = 0) = [1.0, 0.0]$. The parameters are $\epsilon = 1/98$ and $\kappa = 3.0$. The equations are clearly nonlinear. The m-file

```
function [ydot,y0]=ozone(t,y,u,flag)
k=3.0;epsilon=1/98;

if abs(flag) == 1
  ydot(1) = -y(1) - y(2)*y(1) + k*epsilon*y(2);
  ydot(2) = (y(1)-y(1)*y(2)-epsilon*k*y(2))/epsilon;
elseif flag == 0
  ydot=[2,0,0,0,0,0]; %first element=number of equations
  y0=[1 0]; %initial conditions
else
 ydot=[];
end
```

Figure 6.3: MATLAB implementation of ozone decomposition model

named `ozone.m` is shown in figure 6.3. This function should be written in such a way that it should return the derivatives $y'(t)$ in the variable `ydot` when `iflag==1` and it should return the number of equations and the initial condition when `iflag==0` as shown in the figure 6.3.

To use the GEAR and ADAMS functions and integrate the ozone model to a final time of 3.0 do the following during an interactive MATLAB session.

»tf=3.0;	% Define final time
»type ozone	% test that ozone.m exists
»[t,y]=gear('ozone',tf)	% Integrate using GEAR
»[t,y]=adams('ozone',tf)	% Integrate using ADAMS

The independent variable `t` and the solution `y` at the same time values are returned in the corresponding varaibles. These results are shown graphically in figure 6.4. Observe that $y_2(t)$ increases rapidly from an initial condition of zero and hence the system is very stiff during the early times.

6.4 Runge-Kutta Methods

While multi-step methods achieve high order accuracy by making efficient use of previously generated results, Runge-Kutta methods achieve

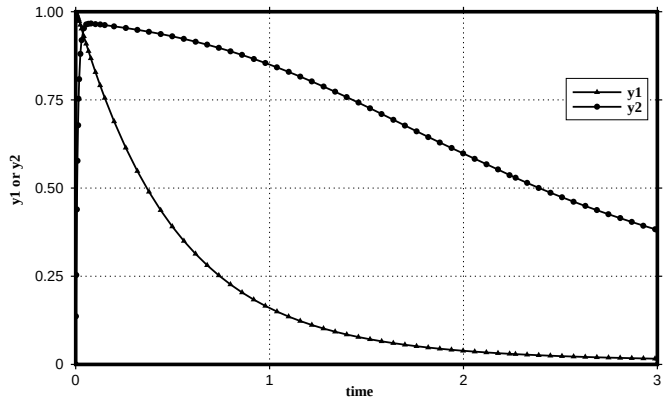

Figure 6.4: Results of ozone decomposition model shows a stiff system behavior

the same goal in a single step, but at the expense of requiring many function evaluations per step. Being single-step schemes, they are self-starters. They are also classified as *explicit, semi-implicit* and *implicit* schemes. Implicit schemes require solution of a set on non-linear algebraic equations at every time step, but they are suitable for *stiff differential equations*.

6.4.1 Explicit schemes

Explicit schemes have the general form,

$$y_{n+1} = y_n + \sum_{i=1}^{v} w_i\, k_i \tag{6.14}$$

$$k_i = h\, f\left(t_n + c_i,\; y_n + \sum_{j=1}^{i-1} a_{ij}\, k_j\right), \qquad c_1 = 0, \qquad i = 1, 2, \cdots v$$

In these equations, $\{c_i, w_i, a_{ij}\}$ are all parameters. The development of a specific scheme entails determining the best possibile values for these constants by matching the expansion of this formula with a Taylor series expansion. Often these parameter values are given in tabular form as,

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

or

$$\begin{array}{c|c} \boldsymbol{c} & \boldsymbol{A} \\ \hline & \boldsymbol{w} \end{array}$$

For explicit methods, $\boldsymbol{A}$ is a lower triangular matrix.

6.4.2 Euler formula revisited

Let us consider $v = 1$ in equation (6.14) and write the equations explicitly as,

$$\begin{aligned} y_{n+1} &= y_n + w_1\, k_1 \\ k_1 &= h\, f(t_n, y_n) \end{aligned} \tag{6.15}$$

or

$$y_{n+1} = y_n + w_1\, h\, f(t_n, y_n)$$

The procedure to determine w_1 is to match the above equaiton with the Taylor series expansion,

$$y_{n+1} = y_n + hy'_n + \frac{h^2}{2}y''_n + \frac{h^3}{3!}y'''_n + \cdots \tag{6.16}$$

The first term on the right hand side is the same in both equations. Recongnizing that $y' = f$ the second term will also match if we make $w_1 = 1$ and this results in recovering the Euler scheme developed in equation (6.7). We cannot match with any higher order terms and hence the local truncation error is of order $\mathcal{O}(h^2)$.

6.4.3 A two-stage ($v = 2$) Runge-Kutta scheme

Let us consider $v = 2$ in equation (6.14) and write the equations explicitly as,

$$\begin{aligned} y_{n+1} &= y_n + w_1\, k_1 + w_2\, k_2 \\ k_1 &= h\, f(t_n, y_n) \\ k_2 &= h\, f(t_n + c_2 h, y_n + a_{21} k_1) \end{aligned} \tag{6.17}$$

This scheme has four unknown paramters $\{w_1, w_2, c_2, a_{21}\}$ which must be determined by matching the Taylor series expansion of equation (6.17) with the equation (6.16). Expanding equation (6.17) we get,

$$y_{n+1} = y_n + w_1\ h\ f(t_n, y_n) + w_2\ h\ f(t_n + c_2 h, y_n + a_{21} k_1)$$

or,

$$y_{n+1} = y_n + w_1\ h\ f_n + w_2\ h\ \left[f + \frac{\partial f}{\partial t}(c_2 h) + \frac{\partial f}{\partial y} a_{21}(hf) \right]_n + \cdots \tag{6.18}$$

Substituting for y' and its higher derivatives in terms of f in equation (6.16) and expanding, we get,

$$y_{n+1} = y_n + h f_n + \frac{h^2}{2}\frac{df}{dt} + \mathcal{O}(h^3)$$

or

$$y_{n+1} = y_n + h f_n + \frac{h^2}{2}\left[\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} \right] + \mathcal{O}(h^3) \tag{6.19}$$

Now comparing the f_n terms between equations (6.18) and (6.19) we require that

$$w_1 + w_2 = 1$$

for the two equations to match. Next comparing $\frac{\partial f}{\partial t}$ terms, we require,

$$w_2 c_2 = 1/2$$

Finally comparing $\frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$ we require,

$$w_2 a_{21} = 1/2$$

Thus, we have matched all terms of order $\mathcal{O}(h^2)$ leaving a truncation error or order $\mathcal{O}(h^3)$. In that process we have developed 3 constraint equations on the four unknonws $\{w_1, w_2, c_2, a_{21}\}$ appearing in the 2-stage RUnge-Kutta scheme (6.17). Any choice of values for $\{w_1, w_2, c_2, a_{21}\}$ that satisfies the above three constraints will result in a 3-rd order, 2-stage Runge-Kutta scheme. Since there are four variables and only three equations, we have one extra degree of freedom. Hence the solution is not unique. Two sets of results are:

$$w_1 = 2/3, \quad w_2 = 1/3, \quad c_2 = 3/2, \quad a_{21} = 3/2$$

and

$$w_1 = 1/2, \quad w_2 = 1/2, \quad c_2 = 1, \quad a_{21} = 1$$

The later is the equivalent of the predictor-corrector pair using Euler and modified Euler schemes developed in equations (6.10). In summary this scheme is a 2-stage RK method since it requires two function evaluations per step. It is explicit and has a local truncation error of $\mathcal{O}(h^3)$.

Using the first set of parameter values in equation (6.17), we have,

$$\begin{aligned} y_{n+1} &= y_n + \frac{2}{3}\, k_1 + \frac{2}{3}\, k_2 \\ k_1 &= h\, f(t_n, y_n) \\ k_2 &= h\, f(t_n + \frac{3}{2}h, y_n + \frac{3}{2}k_1) \end{aligned}$$

or in tabular form,

0		
3/2	3/2	
	2/3	2/3

A fourth order Runge-Kutta scheme

Higher order Runge-Kutta methods can be developed by carrying out the matching process with the Taylor series expansion to higher order terms. An explicit fourth-order form that matches with the Taylor series to h^4 terms (and hence has a truncation error of $\mathcal{O}(h^5)$ is,

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	2/6	2/6	1/6

Which in expanded form,

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}\,[k_1 + 2k_2 + 2k_3 + k_4] \\ k_1 &= h\, f(t_n, y_n) \\ k_2 &= h\, f(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= h\, f(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= h\, f(t_n + h, y_n + k_3) \end{aligned} \tag{6.20}$$

As with all explicit schemes, it is good for non-stiff differential equations.

Embedded forms

The embedded forms of Runge-Kutta algorithms provide a pair of schemes that use a common set of function evaluations to predict two estimates of the solution at y_{n+1}. Typically a lower order scheme is embedded within a higher order scheme. The motivation for developing such schemes is to have a convenient estimate of the local truncation error at every time step, which can then be used to develop a step size control stragegy. A popular scheme, called RK45, is given below.

k_1	0							
k_2	$\frac{1}{4}$	$\frac{1}{4}$						
k_3	$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$					
k_4	$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$				
k_5	1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$\frac{-845}{410}$			
		$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$		$\leftarrow y_{n+1}^{(4)}$
k_6	$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$		
		$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$	$\leftarrow y_{n+1}^{(5)}$

MATLAB has an implimentation of this scheme both as a built-in function called `rk45.m` and a m-file called `ode45.m`.

6.4.4 Semi-implicit & implicit schemes

In the general form given in equation (6.14), we fixed $c_1 = 0$ and $\boldsymbol{A}$ to be lower triangular. These constraints on the paramters ensure that each of the k_i could be computed explicitly without the need for iterative solution. Let us now relax these constraints and write the general form as,

$$\begin{aligned} y_{n+1} &= y_n + \sum_{i=1}^{v} w_i\, k_i \qquad (6.21)\\ k_i &= h\, f\left(t_n + c_i,\; y_n + \sum_{j=1}^{v} a_{ij}\, k_j\right), \qquad i = 1, 2, \cdots v \end{aligned}$$

Thus, for a two-stage process ($v = 2$), we will have,

$$y_{n+1} = y_n + w_1\, k_1 + w_2\, k_2 \qquad (6.22)$$

$$\begin{aligned} k_1 &= h\, f(t_n + c_1 h, y_n + a_{11}k_1 + a_{12}k_2) \\ k_2 &= h\, f(t_n + c_2 h, y_n + a_{21}k_1 + a_{22}k_2) \end{aligned}$$

or in compact tabular form,

c_1	a_{11}	a_{12}
c_2	a_{21}	a_{22}
	w_1	w_2

Note that in order to compute k_1 and k_2 in the above form, we need to solve two sets of nonlinear algebraic equations simulataneously. On the positive side of such schemes, fully implicit nature of the algorithm results in numerically stable schemes making them suitable for *stiff differential equations*. Also, a two-stage scheme ($v = 2$) has eight parameters, and hence we can match these equations with the Taylor series expansion to higher order terms. Hence more accurate formulas can be constructed. An example of a fully implicit, 2-stage, 4-th order accurate scheme is the Gauss form given by,

$(3-\sqrt{3})/6$	$1/4$	$(3-2\sqrt{3})/12$
$(3+\sqrt{3})/6$	$(3-2\sqrt{3})/12$	$1/4$
	$1/2$	$1/2$

For an extensive catalogue of such schemes see Lapidus and Seinfeld (1971).

6.4.5 Semi-Implicit forms of Rosenbrock

While the fully-implicit forms of the last section §6.4.4, have desirable stability properties, they are computationally demanding since a system of non-linear algebraic equations must be solved iteratively at every time step. In an effort to reduce the computational demand while retaining the stability characteristics, Rosenbrock (1963) proposed a special form of the algorithm. These are suitable for autonomous system of equations of the form,

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}) \qquad \boldsymbol{y}(t = t_0) = \boldsymbol{y}_0$$

A 2-stage, 3-rd order scheme is shown below.

$$\begin{aligned} \boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + w_1\, \boldsymbol{k}_1 + w_2\, \boldsymbol{k}_2 \\ \boldsymbol{k}_1 &= h\, [\boldsymbol{I} - h a_1 \boldsymbol{J}(\boldsymbol{y}_n)]^{-1} \boldsymbol{f}(\boldsymbol{y}_n) \\ \boldsymbol{k}_2 &= h\, [\boldsymbol{I} - h a_2 \boldsymbol{J}(\boldsymbol{y}_n + c_1 \boldsymbol{k}_1)]^{-1} \boldsymbol{f}(\boldsymbol{y}_n + b_1 \boldsymbol{k}_1) \end{aligned} \tag{6.23}$$

The parameters are,

$$a_1 = 1 + \sqrt{6}/6, \qquad a_2 = 1 - \sqrt{6}/6$$

$$w_1 = -0.41315432, \qquad w_2 = 1.41315432$$

$$b_1 = c_1 = \frac{-6 - \sqrt{6} + \sqrt{58 + 20\sqrt{6}}}{6 + 2\sqrt{6}}$$

Here $\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}$ is the Jacobian, which must be evaluated at every time step. Note that the main advantage in using equation (6.23) is that $\boldsymbol{k}_1, \boldsymbol{k}_2$ could be computed without the need for iteration, although it requires two matrix inverse computations per step.

6.5 Dynamical systems theory