Chapter 9

ODEs: Initial Value Problems

Outline of Section

- Predictor-Corrector method
- Multi-step methods
- Runge-Kutta methods
- Implicit methods

So far we have only seen one finite difference method (FDM) for integrating ODEs; the Euler method

$$y_{n+1} = y_n + f_n(x_n, y_n)\, h.$$

It uses a forward difference scheme (FDS) to approximate y' in $y' = f(x, y)$. Its properties are summarised as follows:

- It is a 1st-order method: the global error $\sim \mathcal{O}(h)$.
- The local truncation error (i.e. error per step/iteration) is $\mathcal{O}(h^2)$.
- It is a consistent method.
- It is conditionally stable; $h \leq 2/|\partial f/\partial y|$ for a single ODE.
- It is unstable if $\partial f/\partial y > 0$ (single ODE).
- It is an **explicit** method.
- It is a **single-step** method.

Explicit methods involve evaluating $f(x, y)$ using known values of y, e.g., y_n. There are also **implicit** methods which involve evaluating $f(x, y)$ using y_{n+1}. **Single-step** methods just need the solution at the current iteration (y_n) to get the next value. **Multi-step** methods require previous values too (e.g. y_{n-1}) in order to get y_{n+1}.

In this section we look at some more explicit FDMs, which are superior to explicit Euler. We also take a brief look at implicit FDMs, concentrating on the implicit Euler method.

Notation: In this section we will drop the '$\sim$' notation for denoting numerical approximate solutions, unless otherwise indicated.

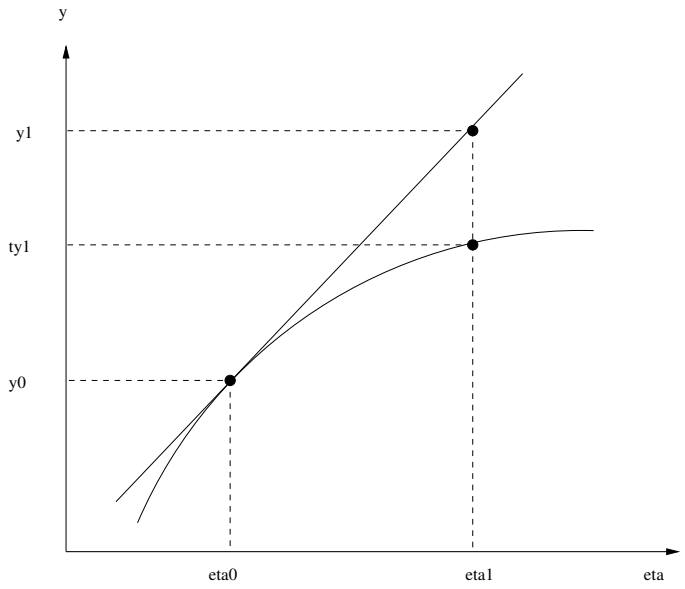

Figure 9.1: The Euler method applied to a non-linear function. The truncation error arises because the gradient is evaluated at the starting point and used to obtain the next point. Here, $\tilde{y}_1$ is the 'approximate' solution according to the Euler method and y_1 (on the curve) is the true solution to the ODE.

9.1 Predictor-Corrector Method

This is classified as an explicit, single-step method. It is a simple improvement to the Euler method. As shown in Fig. 9.1 the Euler method is inaccurate because it uses the gradient evaluated at the initial point to calculate the next point. This only gives a good estimate if the function is linear since the truncation error is quadratic in the step size.

To improve on this we could use the average gradient between the two points

$$y_{n+1} = y_n + \left(\frac{f_n + f_{n+1}}{2}\right) h, \tag{9.1}$$

where $f_n \equiv f(x_n, y_n)$ etc.

To show that this method is second order accurate (i.e. global error $\sim \mathcal{O}(h^2)$) we can start with the Taylor expansion of y_{n+1}

$$y_{n+1} = y_n + f_n\, h + f'_n \frac{h^2}{2} + f''_n(\xi)\, \frac{h^3}{3!}$$

where as usual $x_n < \xi < x_n + h$. Above, y_{n+1} is the exact value (at least until the remainder term is dropped). We can replace f'_n in the above Taylor expansion by a forward difference approximation plus a remainder term

$$f'_n = \frac{f_{n+1} - f_n}{h} - f''_n(\zeta)\frac{h}{2},$$

to get

$$y_{n+1} = y_n + f_n\, h + \left(\frac{f_{n+1} - f_n}{h}\right) \frac{h^2}{2} - \left(f''_n(\zeta)\frac{h}{2}\right)\, \frac{h^2}{2} + \frac{1}{3!} f''_n(\xi)\, h^3\,.$$

Expanding out and grouping terms of $\mathcal{O}(h^3)$ we get

$$y_{n+1} = y_n + \frac{f_{n+1} + f_n}{2}\, h + \mathcal{O}(h^3), \tag{9.2}$$

So the local error (truncation) is 3rd order. Since the integration requires $\mathcal{O}(1/h)$ evaluations the **global error is 2nd order** hence Predictor-Corrector is a **2nd-order method**. Using (9.1) will

give an approximate solution to the ODE since the remainder terms in the Taylor expansion of y_{n+1} and in using the forward difference approximation of f'_n have been discarded.

The only problem with this method is that we don't have the value y_{n+1} to use in $f_{n+1} = f(x_{n+1}, y_{n+1})$ in order to carry out the step. However we can use a first Euler step to *predict* y_{n+1} and use that to calculate f_{n+1} to use in the *corrected* Euler step.

$$\begin{aligned} &\text{step 1} \quad \text{(predict):} && y^\star_{n+1} = y_n + f_n\, h, \\ &&& f^\star_{n+1} = f(x_{n+1}, y^\star_{n+1}), && \text{(9.3a)} \\ &\text{step 2} \quad \text{(correct):} && y^{\text{new}}_{n+1} = y_n + \frac{f^\star_{n+1} + f_n}{2}\, h. && \text{(9.3b)} \end{aligned}$$

The above boxed equation is the algorithm for the predictor-corrector method, for a single 1st-order ODE.

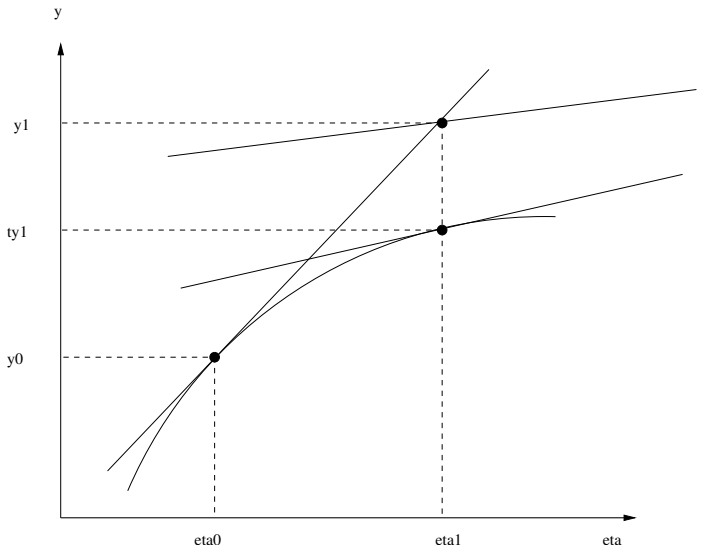

Figure 9.2: (Predictor-corrector) Ideally we would like to use the gradient at y_{n+1} to get the average gradient in the interval. However we do not have the true value y_{n+1} so we use the gradient calculated with the Euler estimate $\tilde{y}_{n+1}$. (Nb: $n = 0$ is used in the plot.)

The gradient calculated with the predicted point will not, in general, be exactly the correct value since it depends on both x and y but it will still be more accurate than the Euler method (see Fig. 9.2). The predictor-corrector method is $\mathcal{O}(h^2)$ accurate but involves two evaluations per step which is less efficient. It is conditionally stable for decaying solutions ($y' = -\alpha y$), but unstable for pure oscillating solutions ($y'' = -\omega^2 y$).

Coupled equations

For a set of m coupled 1st-order ODEs, step 1 should first be applied to each of the m ODEs to get the predicted value $(y^\star_{n+1})^i$ for each dependent variable y^i. Then $(f^\star_{n+1})^i = f^i(x_{n+1}, \vec{y}^\star_{n+1})$ can be evaluated for each ODE. Then step 2 (corrected Euler) can be applied to each ODE. In the compact vector notation, this is

$$\begin{aligned} &\text{step 1} \quad \text{(predict):} && \vec{y}^\star_{n+1} = \vec{y}_n + \vec{f}_n\, h, \\ &&& \vec{f}^\star_{n+1} = \vec{f}(x_{n+1}, \vec{y}^\star_{n+1}), && \text{(9.4a)} \\ &\text{step 2} \quad \text{(correct):} && \vec{y}^{\,\text{new}}_{n+1} = \vec{y}_n + \frac{\vec{f}^\star_{n+1} + \vec{f}_n}{2}\, h. && \text{(9.4b)} \end{aligned}$$

9.2 Multi-Step (Leapfrog) Method

We have seen how the use of a central gradient between two points is more accurate ($\mathcal{O}(h^2)$) than using the gradient at the first point ($\mathcal{O}(h)$). In the previous section we predicted the next point to take an average of the gradient between y_n and y_{n+1}. However since we have presumably evaluated values before y_n in previous steps we could use those to get an update. This requires storing a number of previous points.

The simplest of these methods is the Leapfrog method which uses the y_{n-1} value

$$\boxed{y_{n+1} = y_{n-1} + 2f_n\, h,} \tag{9.5}$$

and requires the storage of one previous step. In this method the point (x_n, y_n) – where the gradient is used – is the midpoint between y_{n-1} and y_{n+1}. The leapfrog algorithm therefore essentially uses a central difference scheme to approximate the derivative y' in the ODE. The leapfrog method is an explicit FDM. The Leapfrog ($m = 1$ order multi-step) method is $\boldsymbol{\mathcal{O}(h^2)}$ **accurate** and only requires one evaluation per step. This makes it more efficient than the Predictor-Corrector method. The method is stable for oscillating and growing solutions but is unstable for decaying solutions.

General multi-step

This kind of multi-step method can use any number of previous values, say 'm'. The Leapfrog is an $m = 1$ order multi-step method. In general, an m^{th}-order multi-step method is an $\boldsymbol{(m+1)^{\text{th}}}$ **order accurate method**.

One way to get an m^{th}-order multi-step method is to fit $P_{m+1}(x)$, an $(m+1)^{\text{th}}$-degree Lagrange polynomial, through the $m+2$ points $y_{n+1}, y_n, y_{n-1}, \ldots, y_{n-m}$ (i.e. including the point to be determined) and then approximate y' in the ODE by P'_{m+1} (for which an analytic expression can easily be obtained, once the coefficients of the interpolating polynomial have been found). Another viable way is to fit $P_m(x)$ to $f(x, y)$ through the $m+1$ points (x_n, f_n), (x_{n-1}, f_{n-1}), $\ldots$, (x_{n-m}, f_{n-m}) and then analytically integrate this interpolated $f(x, y)$ from x_n to x_{n+1} thus advancing the solution to y_{n+1}. This yields the 'Adam-Bashforth' family of methods.

However the drawback of multi-step methods is that $m + 1$ points are needed. So to apply a multi-step method from the outset, values before $x = x_0$ have to be guessed. If the guess is not a good one then the method can suffer from a initial value error.

Starting off

Typically, what is done is to use a single-step method for the first iteration(s). For instance to start off the leapfrog method, an Euler step can be used to get y_1 from the initial condition y_0. Then there are enough points to continue with the leapfrog scheme:

$$\begin{aligned} y_1 &= y_0 + f_0\, h, \\ y_2 &= y_0 + 2f_1\, h, \\ y_3 &= y_1 + 2f_2\, h\,, \\ &\ldots \\ y_{n+1} &= y_{n-1} + 2f_n\, h\,. \end{aligned} \tag{9.6}$$

To improve the accuracy of the starting step, a better single-step method could be used and/or smaller steps can be used (e.g. use k Euler steps with $h \to h/k$ to get y_1 for starting leapfrog).

9.3 Runge-Kutta Methods

The Runge-Kutta methods are a family of explicit, single-step methods using more than one term in the Taylor series expansion of y_{n+1}. The method generalises the idea, employed by the Predictor-

Corrector scheme, of using a weighted average of gradients, to using m estimates of the gradient calculated at various points in the interval $x_n \le x \le x_{n+1}$ to determine the change in y.

In general the **RKm** method equates to an m^{th}-order Taylor expansion, and is an **m^{th}-order finite difference method**. Thus the local error (truncation) is of $\mathcal{O}(h^{m+1})$ and the global accuracy of the method is $\mathcal{O}(h^m)$. A number of weighting schemes for the averaging of the gradients can be used for each RKm.

RK2

The RK2 class of methods includes the Predictor-Corrector method discussed in section 9.1. The general RK2 scheme is;

$$\boxed{\begin{aligned} y_{n+1} &= y_n + a\,k_1 + b\,k_2, && \text{(9.7a)}\\ k_1 &= h\,f(x_n, y_n), && \text{(9.7b)}\\ k_2 &= h\,f(x_n + \alpha\,h, y_n + \beta\,k_1). && \text{(9.7c)} \end{aligned}}$$

Here a, b, α, and β are coefficients. Different choices for the coefficients give different methods

$$\begin{aligned} &a = 0\,, \quad b = 1\,, \quad \alpha = \tfrac{1}{2}\,, \quad \beta = \tfrac{1}{2} && \text{:Single mid-point}\\ &a = \tfrac{1}{2}\,, \quad b = \tfrac{1}{2}\,, \quad \alpha = 1\,, \quad \beta = 1 && \text{:Predictor-Corrector}\\ &a = \tfrac{1}{3}\,, \quad b = \tfrac{2}{3}\,, \quad \alpha = \tfrac{3}{4}\,, \quad \beta = \tfrac{3}{4} && \text{:Smallest error RK2} \end{aligned} \qquad \text{(9.8)}$$

The RK2 method is illustrated in Fig. 9.3. It uses two stages of gradient estimation. At each stage, the gradient is used to estimate the change in y going from x_n to x_{n+1}. The first estimate is an Euler step yielding an estimated shift in y of $k_1 \equiv y^{(1)}_{n+1} - y_n$. In the second stage, a more accurate value of the gradient is estimated using the function f at a point shifted by an amount $\alpha\,h$ in x and by an amount $\beta\,k_1$ in y. This yields a better estimate in the shift in y, i.e., $k_2 = y^{(2)}_{n+1} - y_n$. The two shifts are then averaged with weights a and b.

Coupled ODEs

For coupled ODEs, the first estimate stage must be carried out for all the dependent variables (e.g. u, v, w in $\vec{y} = \{u, v, w\}$) before moving onto the 2nd estimation stage. The RK2 scheme is then;

$$\begin{aligned} \vec{y}_{n+1} &= \vec{y}_n + a\,\vec{k}_1 + b\,\vec{k}_2, && \text{(9.9a)}\\ \vec{k}_1 &= h\,\vec{f}(x_n, \vec{y}_n), && \text{(9.9b)}\\ \vec{k}_2 &= h\,\vec{f}(x_n + \alpha\,h, \vec{y}_n + \beta\,\vec{k}_1). && \text{(9.9c)} \end{aligned}$$

RK4

This uses 4 stages of gradient estimation to calculate an average one and is 4^{th}-order accurate.

$$\boxed{\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}\,(k_1 + 2\,k_2 + 2\,k_3 + k_4)\,, && \text{(9.10a)}\\ k_1 &= h\,f(x_n, y_n), && \text{(9.10b)}\\ k_2 &= h\,f(x_n + \frac{1}{2}\,h, y_n + \frac{1}{2}\,k_1), && \text{(9.10c)}\\ k_3 &= h\,f(x_n + \frac{1}{2}\,h, y_n + \frac{1}{2}\,k_2), && \text{(9.10d)}\\ k_4 &= h\,f(x_n + h, y_n + k_3). && \text{(9.10e)} \end{aligned}}$$

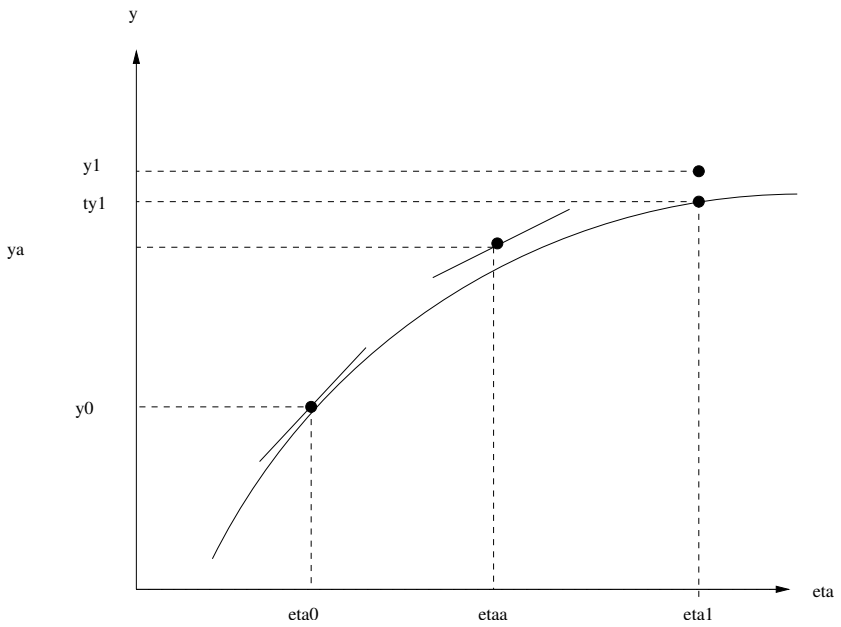

Figure 9.3: The RK2 method. A second gradient is calculated at a shifted position and averaged with the gradient at the starting position to obtain $\tilde{y}_1$, the next value of y. Comparing to fig. 9.1 you can see that $\tilde{y}_1$ is more accurate for RK2 compared to Euler.

Again the first stage is an Euler estimate, as in RK2. The 2nd and 3rd stages estimate the gradient using a fraction $\frac{1}{2}$ of the preceding shifts, k_1 and k_2, respectively. The 4th stage estimates the gradient using the gradient function f evaluated at the best estimate yet of the true value of y_{n+1}, i.e., $y^{(3)}_{n+1} = y_n + k_3$ and x_{n+1}. The values of the weights and shift coefficients are careful chosen to minimise the error.

The advantage of the RK4 method is that, although it requires four evaluations per step, it can take much larger values of h and is therefore more efficient than the Euler and Predictor-Corrector methods. Even higher order RK can be used but these require more (e.g. 11 for RK6) evaluations per step and are therefore slower. Only for $m < 4$ are $\leq m$ evaluations required per step. One practical advantage of RK methods over multi-step methods is that we don't need to store any previous steps. This makes coding it a little easier.

9.4 Variable step size h

With single-step methods, it is pretty easy to vary h during the calculation to, e.g., keep the error within a specified bound. Classic examples are the so-called 'RK45' methods, which produce a fourth-order global error estimate and a fifth-order function estimate. The adaptive stepping in this case is done either via step doubling, or by embedding lower-order Runge-Kutta formulae in higher-order ones. See the lecture slides for an outline of how to do this, and Numerical Recipes for a detailed discussion.

Note that whilst implementing a dynamic step size is pretty easy in single-step algorithms, it is more difficult in multi-step methods, as this requires back-calculating y_{n-1}, etc., using a good single-step method.

9.5 Implicit Methods

Implicit methods for solving an ODE require evaluation of $f(x, y)$ using the yet unknown value y_{n+1}. The simplest is implicit Euler:

$$\boxed{y_{n+1} = y_n + f(x_{n+1}, y_{n+1})\, h.} \tag{9.11}$$

Implicit Euler is still a 1^{st}-order method, with global accuracy $\mathcal{O}(h)$, just like its explicit cousin. For a linear function $f(x, y) = p(x)y + q(x)$ equation (9.11) can easily be rearranged to give $y_{n+1} = g(x_{n+1}, y_n)$ (where g is a new function) which can easily be solved. For a non-linear function, (9.11) is a non-linear algebraic equation in y_{n+1}, that needs to be solved using something like the bisection, Newton or secant method (see section 2.3).

Stability

What is the advantage then? The advantage of implicit Euler over explicit Euler is that it is **unconditionally stable**. There is no critical step size h, above which numerical instability occurs. Of course, using a large h in implicit Euler will give a very inaccurate solution. The stability analysis carried out in section 8.5 (for explicit Euler) can be applied to implicit Euler, in much the same way, yielding

$$\frac{\epsilon_{n+1}}{\epsilon_n} \approx \left(1 - \frac{\partial f}{\partial y} h\right)^{-1} \tag{9.12}$$

so that for decaying problems ($\partial f/\partial y < 0$) we have $g \leq 1$ for ***any*** (positive) h. What happens to y_{n+1} for large h? It simply tends to zero; the same behaviour as the exact solution. Inspection of equation (9.11) for the linear decay problem ($f = -\beta\, y$) shows that $y_{n+1} = y_n/(1 + \beta\, h) \to 0$ as $h \to \infty$.

Just like we have seen more advanced explicit methods (than Euler), more advanced implicit methods exist and have superior stability characteristics to explicit methods.

Coupled ODEs

Implicit Euler works for coupled, linear, 1st-order ODEs. One has

$$\begin{aligned} & & \vec{y}_{n+1} &= \vec{y}_n + h\mathbf{L} \cdot \vec{y}_{n+1} \\ \therefore \quad & (\mathbf{I} - h\mathbf{L}) \cdot \vec{y}_{n+1} &&= \vec{y}_n \\ \therefore \quad & & \vec{y}_{n+1} &= (\mathbf{I} - h\mathbf{L})^{-1} \cdot \vec{y}_n = \bar{\mathbf{T}} \cdot \vec{y}_n \end{aligned} \tag{9.13}$$

where $\mathbf{L}$ is the same matrix operator as for explicit Euler. $\bar{\mathbf{T}}$ is the update matrix for implicit Euler and is the inverse of the matrix $(\mathbf{I} - h\mathbf{L})$. Hence implicit Euler works if $(\mathbf{I} - h\mathbf{L})$ is a non-singular matrix so that $\bar{\mathbf{T}}$ exists and can be found.

With non-linear coupled ODEs, things are not so easy. In principle one needs to find the solution of a set of (coupled) non-linear algebraic equations. This is at best tricky and at worst insoluble. A way out is to linearise the functions f^i in each ODE about the known 'point' $(x_n, \vec{y}_n)$. But then the update matrix $\bar{\mathbf{T}}$ has to be calculated at each step, since the matrix $\mathbf{L}$ obtained by linearisation depends on $\vec{y}_n$. Thus implicit methods are generally less efficient than explicit methods.