Chapter 3

A Survey of Methods

In the last chapter we looked at the forward and backward Euler method for approximating the solution to the first order IVP (2.2). Although both methods are simple to understand and program, they converge at a linear rate which is often prohibitively slow. In this chapter we provide a survey of schemes which have higher than linear accuracy. Also in Example 2.8 we saw that the forward Euler method gave unreliable results for some choices of time steps so we need to investigate when this numerical instability occurs so it can be avoided.

The standard methods for approximating the solution of (2.2) fall into two broad categories which are **single-step/one-step methods** and **multistep methods**. Each category consists of families of explicit and families of implicit methods of varying degrees of accuracy. Both the forward and backward Euler methods are single-step methods because they only use information from one previously computed solution, that is, at t_n, to compute the solution at t_{n+1}. We have not encountered multistep methods yet but they use information at several previously calculated points; for example, a two-step method uses information at t_n and t_{n-1} to predict the solution at t_{n+1}. Using previously calculated information is how multistep methods improve on the linear accuracy of the Euler method. One-step methods improve the accuracy by performing intermediate approximations in $(t_n, t_{n+1}]$ which are then discarded. We will see that there are advantages and disadvantages to each class of methods.

Instead of simply listing common single-step and multistep methods, we want to understand how these methods can be derived so that we gain a better understanding. For one-step methods we begin by using Taylor series expansions to derive the Euler methods and see how this approach can be easily extended to get higher order accurate methods. However, we will see that these methods require repeated differentiation of the given slope $f(t, y)$ and so are not, in general, practical. To obtain methods which don't require repeated differentiation we investigate how numerical quadrature rules and interpolating polynomials can be used to derive methods. In these approaches we integrate the differential equation from t_n to t_{n+1} and either

use a quadrature rule to evaluate the integral of $f(t, y)$ or first replace $f(t, y)$ by an interpolating polynomial and then integrate. We will also introduce a systematic approach to deriving schemes called the method of undetermined coefficients. In this approach we let the coefficients in the scheme be undetermined parameters and determine conditions on the parameters so that the scheme has as high accuracy as possible. The Runge-Kutta methods discussed in § 3.1.3 are the most popular one-step methods. After introducing Runge-Kutta methods we investigate how to determine if a single-step method is stable for all choices of time step; if it is not, we show how to obtain conditions on the time step which guarantee stability.

Multistep methods can be derived in analogous ways. However, when we integrate the equation or use an interpolating polynomial we need to include points such as t_{n-1}, t_{n-2}, etc. Backward difference methods are discussed in § 3.2.1 and the widely used Adams-Bashforth and Adams-Moulton families of multistep methods are presented in § 3.2.2. Because multistep methods rely on previously computed values the stability analysis is more complicated than for single-step methods so we will simply summarize the conditions for stability.

For both multistep and single-step methods we provide numerical results and demonstrate that the numerical rate of convergence agrees with the theoretical results. Efficient implementation of the methods is discussed.

In § 3.3 we see how using Richardson extrapolation can take a sequence of approximations from a lower order method and generate more accurate approximations without additional function evaluations. This approach can be used with single-step or multistep methods. We will discuss how the currently popular Burlisch-Stoer extrapolation method exploits certain properties of methods to provide a robust algorithm with high accuracy.

In Chapter 2 we saw that implicit methods were inherently more costly to implement than explicit methods due to the fact that they typically require the solution of a nonlinear equation at each time step. In § 3.4 we investigate an efficient way to implement an implicit method by pairing it with an explicit method to yield the so-called Predictor-Corrector methods.

3.1 Single-Step Methods

In this section we look at the important class of numerical methods for the IVP (2.2) called single-step or one-step methods which only use information at one previously calculated point, t_n to approximate the solution at t_{n+1}. Both the forward and backward Euler methods are one-step methods with linear accuracy. To improve on the accuracy of these methods, higher order explicit single-step methods compute additional approximations in the interval (t_n, t_{n+1}) which are used to approximate the solution at t_{n+1}; these intermediate approximations are then discarded.

We first look at Taylor series methods which are easy to derive but impractical to use because they require repeated differentiation of the given slope $f(t, y)$. Then we demonstrate how other one-step methods can be derived by first integrating the differential equation and using a numerical quadrature rule to approximate the integral of the slope. One shortcoming of this approach is that for each quadra-

ture rule we choose we obtain a different method whose accuracy must then be obtained. An alternate approach to deriving methods is to form a general explicit or implicit method, assuming a fixed number of additional function evaluations, and then determine the coefficients in the scheme so that one has as high an accuracy as possible. This approach results in families of methods which have a given accuracy and so eliminates the tedious local truncation error calculations. Either approach leads to the Runge-Kutta family of methods which we discuss in § 3.1.3.

3.1.1 Taylor series methods

Taylor series is an extremely useful tool in numerical analysis, especially in deriving and analyzing difference methods. In this section we demonstrate that it is straightforward to derive the forward Euler method using Taylor series and then to generalize the approach to derive higher order accurate schemes. Unfortunately, these schemes are not very practical because they require repeated differentiation of the given slope $f(t, y)$. This means that software implementing these methods will be very problem dependent and require several additional routines to be provided by the user for each problem. Additionally, $f(t, y)$ may not possess higher order derivatives.

To derive explicit methods using Taylor series we use the differential equation evaluated at t_n so we need an approximation to $y'(t_n)$. Thus we expand $y(t_n + \Delta t)$ about t_n to get

$$y(t_n + \Delta t) = y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2!} y''(t_n) + \cdots + \frac{(\Delta t)^k}{k!} y^{[k]}(t_n) + \cdots . \quad (3.1)$$

This is an infinite series so if we truncate it then we have an approximation to $y(t_n + \Delta t)$ from which we can approximate $y'(t_n)$. For example, if we truncate the series at the term which is $\mathcal{O}(\Delta t)$ we have

$$y(t_n+\Delta t) \approx y(t_n)+\Delta t y'(t_n) = y(t_n)+\Delta t f(t_n, y(t_n)) \Rightarrow y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{\Delta t}$$

which leads to the forward Euler method when we substitute into the differential equation evaluated at t_n, i.e., $y'(t_n) = f\big(t_n, y(t_n)\big)$. Alternately, we can view this as giving the difference equation directly.

So theoretically, if we keep additional terms in the series expansion for $y(t_n+\Delta t)$ then we get a higher order approximation. To see how this approach works, we now keep three terms in the expansion and thus have a remainder term of $\mathcal{O}(\Delta t^3)$. From (3.1) we have

$$y(t_n + \Delta t) \approx y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2!} y''(t_n) \,,$$

so we expect a local error of $\mathcal{O}(\Delta t^3)$ which leads to an expected global error of $\mathcal{O}(\Delta t^2)$. Now the problem we have to address when we use this expansion is what to do with $y''(t_n)$ because we only know $y'(t) = f(t, y)$. If our function is smooth enough, we can differentiate this equation with respect to t to get $y''(t)$. To do this

recall that we have to use the chain rule because f is a function of t and y where y is also a function t. Specifically, we have

$$y'(t) = f(t,y) \Rightarrow y''(t) = \frac{\partial f}{\partial t}\frac{dt}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} = f_t + f_y f\,.$$

Substituting this into the expression for $y(t_n + \Delta t)$ gives

$$y(t_n+dt) \approx y(t_n)+\Delta t f\big(t_n, y(t_n)\big)+\frac{(\Delta t)^2}{2!}\Big[\big(f_t(t_n, y(t_n))+f\big(t_n, y(t_n)\big)\big(f_y(t_n, y(t_n))\big)\Big]$$

which gives the second order explicit Taylor series method.

Second order explicit Taylor series method

$$Y^{n+1} = Y^n + \Delta t f(t_n, Y^n) + \frac{(\Delta t)^2}{2}\Big[f_t(t_n, Y^n) + f(t_n, Y^n) f_y(t_n, Y^n)\Big] \quad (3.2)$$

To implement this method, we must provide function routines not only for $f(t,y)$ but also $f_t(t,y)$ and $f_y(t,y)$. In some cases this will be easy, but in others it can be tedious or even not possible. The following example applies the second order Taylor scheme to a specific IVP and in the exercises we explore a third order Taylor series method.

Example 3.1. SECOND ORDER TAYLOR METHOD

Approximate the solution to

$$y'(t) = 3yt^2 \quad y(0) = \frac{1}{3}$$

using (3.2) and verify the quadratic convergence. The exact solution is $\frac{1}{3}e^{t^3}$.

Before writing a code for a particular method, it is helpful to first perform some calculations by hand so it is clear that the method is completely understood and also to have some results with which to compare the numerical simulations for debugging. To this end, we first calculate Y^1 and Y^2 using $\Delta t = 0.1$. Then we provide numerical results at $t = 1$ for several choices of Δt and compare with a first order explicit Taylor series method, i.e., with the forward Euler method.

From the discussion in this section, we know that we need f_t and f_y so

$$f(t,y) = 3yt^2 \Rightarrow f_t = 6ty \quad \text{and} \quad f_y = 3t^2\,.$$

Substitution into the difference equation (3.2) gives the expression

$$Y^{n+1} = Y^n + 3\Delta t Y^n t_n^2 + \frac{(\Delta t)^2}{2}\left(6t_n Y^n + 9t_n^4 Y^n\right)\,. \quad (3.3)$$

For $Y^0 = 1/3$ we have

$$Y^1 = \frac{1}{3} + 0.1(3)\left(\frac{1}{3}\right)0 + \frac{(.1)^2}{2}(0) = \frac{1}{3}$$

and

$$Y^2 = \frac{1}{3} + 0.1(3)\left(\frac{1}{3}\right)(.1)^2 + \frac{(.1)^2}{2}\left(6(.1)\frac{1}{3} + 9(.1)^4\frac{1}{3}\right) = 0.335335\,.$$

The exact solution at $t = 0.2$ is 0.336011 which gives an error of $0.675862\ 10^{-3}$.

To implement this method in a computer code we modify our program for the forward Euler method to include the $\mathcal{O}(\Delta t^2)$ terms in (3.2). In addition to a function for $f(t, y)$ we also need to provide function routines for its first partial derivatives f_y and f_t; note that in our program we code the general equation (3.2), not the equation (3.3) specific to our problem. We perform calculations with decreasing values of Δt and compare with results at $t = 1$ using the forward Euler method. When we compute the numerical rate of convergence we see that the rate of convergence is $\mathcal{O}(\Delta t^2)$, as expected whereas the forward Euler is only linear. For this reason when we compare the global errors at a fixed, small time step we see that the error is much smaller for the second order method because it is converging to zero faster than the Euler method.

Δt	Error in Euler	Numerical rate	Error in second order Taylor	Numerical rate
$1/4$	$3.1689\ 10^{-1}$		$1.2328\ 10^{-1}$	
$1/8$	$2.0007\ 10^{-1}$	0.663	$4.1143\ 10^{-2}$	1.58
$1/16$	$1.1521\ 10^{-1}$	0.796	$1.1932\ 10^{-2}$	1.79
$1/32$	$6.2350\ 10^{-2}$	0.886	$3.2091\ 10^{-3}$	1.89
$1/64$	$3.2516\ 10^{-2}$	0.939	$8.3150\ 10^{-4}$	1.95
$1/128$	$1.6615\ 10^{-2}$	0.969	$2.1157\ 10^{-4}$	1.97

Implicit Taylor series methods can be derived in an analogous manner. In this case we use the differential equation evaluated at t_{n+1}, i.e., $y'(t_{n+1}) = f\big(t_{n+1}, y(t_{n+1})\big)$. Consequently we need an approximation to $y'(t_{n+1})$ instead of $y'(t_n)$ so we use the expansion

$$y(t_n) = y(t_{n+1} - \Delta t) = y(t_{n+1}) - \Delta t y'(t_{n+1}) + \frac{(\Delta t)^2}{2!}y''(t_{n+1}) + \cdots\,. \quad (3.4)$$

Keeping terms through $\mathcal{O}(\Delta t)$ gives the backward Euler method. In the exercises you are asked to derive a second order implicit Taylor series method.

Taylor series methods are single step methods. Although using these methods results in methods with higher order accuracy than the Euler methods, they are not considered practical because of the requirement of repeated differentiation of $f(t, y)$. For example, the first full derivative has two terms and the second has five terms. So even if $f(t, y)$ can be differentiated, the methods become unwieldy. To implement the methods on a computer the user must provide routines for all the partial derivatives so the codes become very problem dependent. For these reasons we look at other approaches to derive higher order schemes.

3.1.2 Derivation of methods which don't require repeated differentiation

Taylor series methods are not practical because they require repeated differentiation of the solution so it is important to obtain methods which don't need to do this. If we integrate the differential equation then the left-hand side can be evaluated easily but, in general, we won't be able to integrate the given slope. One approach is to use a numerical quadrature rule to approximate this integral. A second approach is to use an interpolating polynomial in $[t_n, t_{n+1}]$ to approximate the slope and then integrate. Because many quadrature rules are interpolatory in nature, these two approaches often yield equivalent schemes. The use of a quadrature rule is more general so we concentrate on that approach. The shortcoming of this approach is that for each method we derive, we have to demonstrate its accuracy. An approach which generates families of methods of a prescribed accuracy is illustrated in § 3.1.2.

Using numerical quadrature

We first derive one-step methods by integrating the differential equation from t_n to t_{n+1} and then approximating the integral of the slope using a numerical quadrature rule. When we integrate $(2.2\mathrm{a})$ from t_n to t_{n+1} we have

$$\int_{t_n}^{t_{n+1}} y'(t)\,dt = \int_{t_n}^{t_{n+1}} f(t,y)\,dt\,. \tag{3.5}$$

The integral on the left hand side can be evaluated exactly by the Fundamental Theorem of Calculus to get $y(t_{n+1}) - y(t_n)$. However, in general, we must use numerical quadrature to approximate the integral on the right hand side. Recall from calculus that one of the simplest approximations to an integral is to use either a left or right Riemann sum, i.e., if the integrand is nonnegative then we are approximating the area under the curve by a rectangle. If we use a left sum for the integral we approximate the integral by a rectangle whose base is Δt and whose height is determined by the integrand evaluated at the left endpoint of the interval; i.e., we use the formula

$$\int_a^b g(x) \approx g(a)(b-a)\,.$$

Using the left Riemann sum to approximate the integral of $f(t,y)$ gives

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t,y)\,dt \approx \Delta t f\big(t_n, y(t_n)\big)$$

which leads us to the forward Euler method. In the exercises you will explore the implications of using a right Riemann sum. Clearly different approximations to the integral of $f(t,y)$ yield different methods.

Numerical quadrature rules for single integrals have the general form

$$\int_a^b g(t)\,dt \approx \sum_{i=1}^{Q} w_i g(q_i)$$

where the scalars w_i are called the quadrature weights, the points q_i are the quadrature points in $[a, b]$ and Q is the number of quadrature points used. One common numerical integration rule is the midpoint rule where, as the name indicates, we evaluate the integrand at the midpoint of the interval; specifically the midpoint quadrature rule is

$$\int_a^b g(t)\, dt \approx (b-a) g\left(\frac{a+b}{2}\right).$$

Using the midpoint quadrature rule to approximate the integral of $f(t, y)$ in (3.5) gives

$$y(t_{n+1}) - y(t_n) \approx \Delta t f\big(t_n + \frac{\Delta t}{2}, y(t_n + \frac{\Delta t}{2})\big).$$

The problem with this approximation is that we can't evaluate $f\big(t_n+\frac{\Delta t}{2}, y(t_n+\frac{\Delta t}{2})\big)$ because we don't know y evaluated at the midpoint. Our only recourse is to use an approximation. If we use the forward Euler method starting at t_n and take a step of length $\Delta t/2$ then this produces an approximation to y at the midpoint i.e.,

$$y(t_n + \frac{\Delta t}{2}) \approx y(t_n) + \frac{\Delta t}{2} f\big(t_n, y(t_n)\big).$$

Thus we can view our method as having two parts; first we approximate y at the midpoint using Euler's method and then use it to approximate $y(t_{n+1})$. Combining these into one equation allows the scheme to be written as

$$Y^{n+1} = Y^n + \Delta t f\big(t_n + \frac{\Delta t}{2}, Y^n + \frac{1}{2}\Delta t f(t_n, Y^n)\big).$$

However, the method is usually written in the following way for clarity and to emphasize the fact that there are two function evaluations required.

Midpoint Rule

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + \tfrac{\Delta t}{2}, Y^n + \tfrac{1}{2}k_1) \\ Y^{n+1} &= Y^n + k_2 \end{aligned} \tag{3.6}$$

The midpoint rule has a simple geometrical interpretation. Recall that for the forward Euler method we used the tangent line at t_n to extrapolate the solution at t_{n+1}. In the midpoint rule we use the tangent line at $t_n + \Delta t/2$ to extrapolate the solution at t_{n+1}. Heuristically we expect this to give a better approximation than the tangent line at t_n.

The midpoint rule is a single step method because it only uses one previously calculated solution, i.e., the solution at t_n. However, it requires one more function evaluation than the forward Euler method but, unlike the Taylor series methods, it does not require additional derivatives of the slope $f(t, y)$. Because we are doing more work than the Euler method, we would like to think that the scheme

converges faster. In the next example we demonstrate that the local truncation error of the midpoint method is $\mathcal{O}(\Delta t^3)$ so that we expect the method to converge with a global error of $\mathcal{O}(\Delta t^2)$. The steps in estimating the local truncation error for the midpoint method are analogous to the ones we performed for determining the local truncation error for the forward Euler method except now we need to use a Taylor series expansion in two independent variables for $f(t, y)$ because of the term $f(t_n + \frac{\Delta t}{2}, Y^n + \frac{1}{2}k_1)$.

Example 3.2. LOCAL TRUNCATION ERROR FOR THE MIDPOINT RULE

Show that the local truncation error for the midpoint rule is exactly $\mathcal{O}(\Delta t^3)$.

Recall that the local truncation error is the remainder when the exact solution is substituted into the difference equation. For the midpoint rule the local truncation error τ_{n+1} at t_{n+1} is

$$\tau_{n+1} = y(t_{n+1}) - \left[y(t_n) + \Delta t f\big(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f(t_n, y(t_n))\big) \right] . \tag{3.7}$$

As before, we expand $y(t_{n+1})$ with a Taylor series but this time we keep the terms through $(\Delta t)^3$ because we want to demonstrate that terms in the expression for the truncation error through $(\Delta t)^2$ cancel but terms involving $(\Delta t)^3$ do not; this way we will demonstrate that the local truncation is *exactly* $\mathcal{O}(\Delta t^3)$ rather than it is *at least* $\mathcal{O}(\Delta t^3)$. We have

$$y(t_{n+1}) = y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2} y''(t_n) + \frac{(\Delta t)^3}{3!} y'''(t_n) + \mathcal{O}(\Delta t^4) . \tag{3.8}$$

Substituting this into (3.7) yields

$$\begin{aligned} \tau_{n+1} \quad = \quad & \left[y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2} y''(t_n) + \frac{(\Delta t)^3}{3!} y'''(t_n) + \mathcal{O}(\Delta t^4) \right] \\ & - \left[y(t_n) + \Delta t f\big(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f(t_n, y(t_n))\big) \right] . \end{aligned} \tag{3.9}$$

Now all terms are evaluated at t_n except the term $f(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f(t_n, y(t_n)))$. Because this term is a function of two variables instead of one we need to use the Taylor series expansion given in (18). To use this formula we note that the change in the first variable t is $h = \Delta t/2$ and the change in the second variable y is $k = (\Delta t/2) f\big(t_n, y(t_n)\big)$. Thus we have

$$\begin{aligned} \Delta t f\big(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f(t_n, y(t_n))\big) = \Delta t \Big[& f + \frac{\Delta t}{2} f_t + \frac{\Delta t}{2} f f_y \\ & + \frac{(\Delta t)^2}{4 \cdot 2!} f_{tt} + \frac{(\Delta t)^2}{4 \cdot 2!} f^2 f_{yy} + 2 \frac{(\Delta t)^2}{4 \cdot 2!} f f_{ty} + \mathcal{O}(\Delta t^3) \Big] . \end{aligned}$$

All terms involving f or its derivatives on the right-hand side of this equation are evaluated at $(t_n, y(t_n))$ and we have omitted this explicit dependence for brevity. Substituting this expansion into the expression (3.9) for τ_{n+1} and collecting terms involving each power of Δt yields

$$\begin{aligned} \tau_{n+1} \quad = \quad & \Delta t \Big[y' - f \Big] + \Delta t^2 \Big[\frac{1}{2} y'' - \frac{1}{2} (f_t + f f_y) \Big] \\ & + \Delta t^3 \big(\frac{1}{3!} y''' - \frac{1}{8} \Big[f_{tt} + f^2 f_{yy} + 2 f f_{ty} \Big] + \mathcal{O}(\Delta t^4) . \end{aligned} \tag{3.10}$$

Clearly $y' = f$ so the term involving Δt cancels but to see if the other terms cancel we need to write y'' and y''' in terms of f and its derivatives. To write $y''(t)$ in terms of $f(t,y)$ and its partial derivatives we have to differentiate $f(t,y)$ with respect to t which requires the use of the chain rule. We have

$$y''(t) = \frac{\partial f}{\partial t}\frac{\partial t}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} = f_t + f_y y' = f_t + f_y f\,.$$

Similarly,

$$\begin{aligned} y'''(t) &= \frac{\partial (f_t + f_y f)}{\partial t}\frac{\partial t}{\partial t} + \frac{\partial (f_t + f_y f)}{\partial y}\frac{\partial y}{\partial t} \\ &= f_{tt} + f_{yt} f + f_y f_t + \big(f_{ty} + f_{yy} f + f_y^2\big) f \\ &= f_{tt} + 2 f_{yt} f + f_y f_t + f_{yy} f^2 + f f_y^2\,. \end{aligned}$$

Thus the terms in (3.10) involving Δt and $(\Delta t)^2$ cancel but the terms involving $(\Delta t)^3$ do not. Consequently the local truncation error converges cubically and we expect the global convergence rate to be quadratic.

If we use a Riemann sum or the midpoint rule to approximate an integral $\int_a^b g(t)dt$ where $g(t) \geq 0$ on $[a,b]$ then we are using a rectangle to approximate the area. An alternate approach is to use a trapezoid to approximate the area. The trapezoidal integration rule is found by calculating the area of the trapezoid with base $(b-a)$ and height determined by the line passing through $(a, g(a))$ and $(b, g(b))$; specifically the rule is

$$\int_a^b g(t)\, dt \approx \frac{(b-a)}{2}\Big(g(a) + g(b)\Big)\,.$$

Integrating the differential equation $(2.2\mathrm{a})$ from t_n to t_{n+1} and using this quadrature rule gives

$$y(t_{n+1}) - y(t_n) \approx \frac{\Delta t}{2}\Big[f\big(t_n, y(t_n)\big) + f\big(t_{n+1}, y(t_{n+1})\big)\Big]\,.$$

Trapezoidal Rule $$Y^{n+1} = Y^n + \frac{\Delta t}{2}\big[f(t_n, Y^n) + f(t_{n+1}, Y^{n+1})\big] \qquad (3.11)$$

However, like the backward Euler method this is an implicit scheme and thus for each t_n we need to solve a nonlinear equation for most choices of $f(t,y)$. This can be done, but there are better approaches for using implicit schemes in the context of ODEs as we will see in § 3.4.

Other numerical quadrature rules on the interval from $[t_n, t_{n+1}]$ lead to additional explicit and implicit one-step methods. The Euler method, the midpoint rule and the trapezoidal rule all belong to a family of methods called **Runge-Kutta methods** which we discuss in § 3.1.3.

Many quadrature rules are interpolatory in nature; that is, the integrand is approximated by an interpolating polynomial which can then be integrated exactly. For example, for $\int_a^b g(x)\,dx$ we could use a constant, a linear polynomial, a quadratic polynomial, etc. to approximate $g(x)$ in $[a,b]$ and then integrate it exactly. We want to use a Lagrange interpolating polynomial instead of a Hermite because the latter requires derivatives. If we use $f(t,y(t)) \approx f(t_n, y(t_n))$ in $[t_n, t_{n+1}]$ then we get the forward Euler method and if we use $f(t,y(t)) \approx f(t_n + \Delta t/2, y(t_n + \Delta t/2))$ then we get the midpoint rule. So to derive some single-step methods we can use interpolation but only using points in the interval $[t_n, t_{n+1}]$. However there are many quadrature rules, such as Gauss quadrature, which are not interpolatory and so using numerical quadrature as an approach to deriving single-step methods is more general.

Using the method of undetermined coefficients

One problem with deriving methods using numerical integration or interpolation is that once we have obtained a method then we must determine its local truncation error which is straightforward but often tedious. A systematic approach to deriving methods is to assume the most general form of the desired method and then determine constraints on the coefficients in the general method so that it has a local truncation error which is as high as possible.

Suppose we want an explicit one-step scheme and we are only willing to perform one function evaluation which is at (t_n, Y^n). The forward Euler method is such a scheme; we want to determine if there are any others, especially one which converges at a higher rate. The most general form of an explicit one-step scheme is

$$Y^{n+1} = \alpha Y^n + b_1 \Delta t f(t_n, Y^n)\,;$$

note that the forward Euler method has $\alpha = b_1 = 1$. To determine the local truncation error we substitute the exact solution into the difference equation and calculate the remainder. We have

$$\begin{aligned}
\tau_{n+1} &= y(t_{n+1}) - \alpha y(t_n) - b_1 \Delta t f\big(t_n, y(t_n)\big) \\
&= \Big[y(t_n) + \Delta t y'(t_n) + \frac{\Delta t^2}{2!} y''(t_n) + \frac{\Delta t^3}{3!} y'''(\xi_n)\Big] - \alpha y(t_n) - b_1 \Delta t y'(t_n) \\
&= y(t_n)\big[1-\alpha\big] + y'(t_n)\Delta t\big[1-b_1\big] + y''(t_n)\Delta t^2\big[\frac{1}{2}\big] + \frac{\Delta t^3}{3!} y'''(\xi_n)\,,
\end{aligned}$$

where we have expanded $y(t_{n+1})$ in a Taylor series with remainder and used the fact that $y'(t) = f(t,y)$ in the second step. In the last step we have grouped the constant terms, the terms involving Δt, Δt^2 and Δt^3. For the constant term to disappear we require that $\alpha = 1$; for the linear term in Δt we require that $b_1 = 1$. The term involving Δt^2 can not be made to disappear so the only explicit method with one function evaluation which has a local truncation $\mathcal{O}(\Delta t^2)$ is the forward Euler method. No explicit method using only the function evaluation at t_n with a local truncation error greater than $\mathcal{O}(\Delta t^2)$ is possible. Note that α must always be one to cancel the $y(t_n)$ term in the expansion of $y(t_n + \Delta t)$ so in the sequel there is no need for it to be unknown.

The following example illustrates this approach if we want an explicit single-step method where we are willing to perform one additional function value in the interval $[t_n, t_{n+1}]$. We already know that the midpoint rule is a second order method which requires the additional function evaluation at $t_n + \Delta t/2$. However, this example demonstrates that there is an infinite number of such methods.

Example 3.3. DERIVATION OF A SECOND ORDER EXPLICIT SINGLE-STEP METHOD

We now assume that in addition to evaluating $f(t_n, Y^n)$ we want to evaluate the slope at one intermediate point in $(t_n, t_{n+1}]$. Because we are doing an additional function evaluation, we expect that we should be able to make the truncation error smaller if we choose the parameters correctly; i.e., we choose an appropriate point in $(t_n, t_{n+1}]$. A random point may not give us second order accuracy. We must leave the choice of the location of the point as a variable so our general difference equation is

$$Y^{n+1} = Y^n + b_1 \Delta t f(t_n, Y^n) + b_2 \Delta t f\big(t_n + c_2 \Delta t, Y^n + a_{21} \Delta t f(t_n, Y^n)\big) \qquad (3.12)$$

where the general point in $(t_n, t_{n+1}]$ is $t_n + c_2 \Delta t$ and the general approximation to y at this point is $Y^n + a_{21} \Delta t f(t_n, Y^n)$. Recall that we have set $\alpha = 1$ as in the derivation of the forward Euler method.

To determine constraints on the parameters b_1, b_2, c_2 and a_{21} which result in the highest order for the truncation error, we compute the local truncation error and use Taylor series to expand the terms. For simplicity, in the following expansion we have omitted the explicit evaluation of f and its derivatives at the point $(t_n, y(t_n))$; however, if f is evaluated at some other point we have explicitly noted this. We use (18) for a Taylor series expansion in two variables to get

$$\begin{aligned}
\tau_{n+1} &= \left[y + \Delta t y' + \frac{\Delta t^2}{2!} y'' + \frac{\Delta t^3}{3!} y''' + \mathcal{O}(\Delta t^4)\right] \\
&\quad - \Big[y + b_1 \Delta t f + b_2 \Delta t f(t_n + c_2 \Delta t, y + a_{21} \Delta t f)\Big] \\
&= \Big[\Delta t f + \frac{\Delta t^2}{2}\big(f_t + f f_y\big) + \frac{\Delta t^3}{6}\big(f_{tt} + 2 f f_{ty} + f^2 f_{yy} + f_t f_y + f f_y^2\big) + \mathcal{O}(\Delta t)^4)\Big] \\
&\quad - b_1 \Delta t f - b_2 \Delta t \Big[f + c_2 \Delta t f_t + a_{21} \Delta t f f_y \\
&\quad + \frac{c_2^2 (\Delta t)^2}{2} f_{tt} + \frac{a_{21}^2 (\Delta t)^2 f^2}{2} f_{yy} + c_2 a_{21} (\Delta t)^2 f f_{ty} + \mathcal{O}(\Delta t^3)\Big] .
\end{aligned}$$

We first see if we can determine the parameters so that the scheme has a local truncation error of $\mathcal{O}(\Delta t^3)$; to this end we must determine the equations that the unknown coefficients must satisfy in order for the terms involving $(\Delta t)^1$ and $(\Delta t)^2$ to vanish. We have

$$\begin{aligned}
\Delta t \left[f(1 - b_1 - b_2)\right] &= 0 \\
\Delta t^2 \left[f_t\big(\frac{1}{2} - b_2 c_2\big) + f f_y\big(\frac{1}{2} - b_2 a_{21}\big)\right] &= 0
\end{aligned}$$

where once again we have dropped the explicit evaluation of y and f at $(t_n, y(t_n))$. Thus we have the conditions

$$b_1 + b_2 = 1, \quad b_2 c_2 = \frac{1}{2} \quad \text{and} \quad b_2 a_{21} = \frac{1}{2} . \qquad (3.13)$$

Note that the midpoint method given in (3.6) satisfies these equations with $b_1 = 0, b_2 = 1$, $c_2 = a_{21} = 1/2$. However, any choice of the parameters which satisfy these constraints generates a method with a third order local truncation error.

Because we have four parameters and only three constraints we might ask ourselves if it is possible to choose the parameters so that the local truncation error is one order higher, i.e., $\mathcal{O}(\Delta t^4)$. To see that this is impossible note that in the expansion of $y(t_{n+1})$ the term y''' involves terms such as $f_t f_y$ for which there are no corresponding terms in the expansion of $f\big(t_n + c_2\Delta t, Y^n + a_{21}\Delta t f(t_n, Y^n)\big)$ so these $\mathcal{O}(\Delta t^3)$ terms will remain. Consequently there is no third order explicit one-step method which only performs two function evaluations per time step.

3.1.3 Runge-Kutta methods

Runge-Kutta[1] (RK) methods are a family of one-step methods which include both explicit and implicit methods. They are further characterized by the number of **stages** which is just the number of function evaluations performed at each time step. The forward Euler and the midpoint methods are examples of explicit RK methods; the Euler method is a one-stage method whereas the midpoint method is a two-stage method because it uses information at the midpoint $t_n + \Delta t/2$ as well as at t_n. Both the backward Euler and the trapezoidal methods are examples of implicit RK methods; the Euler method is a one-stage method whereas the trapezoidal method is a two-stage method. The family of RK methods were developed primarily from 1895 to 1925 and involved work by Runge, Heun, Kutta and Nystrom. Interested readers should refer to the paper by J.C. Butcher entitled "A history of Runge-Kutta methods."

The standard approach for the derivation of families of RK methods is to use the method of undetermined coefficients discussed in § 3.1.2 because it gives families of methods with a prescribed local truncation error. This was illustrated in Example 3.3 where we assumed the most general form of an explicit two-stage single step method in (3.12). To illustrate the fact that it is a two-stage method we can also write (3.12) as

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + c_2\Delta t, Y^n + a_{21}k_1) \\ Y^{n+1} &= Y^n + b_1 k_1 + b_2 k_2\,. \end{aligned}$$

We obtained the constraints $b_1 + b_2 = 1$, $b_2 c_2 = 1/2$ and $b_2 a_{21} = 1/2$ and so there is a family of second order accurate two-stage explicit RK methods. The midpoint method which we derived using numerical quadrature is one of the two-stage RK methods. Another commonly used two-stage RK method is the Heun method where the intermediate point is $\big(t_n + \frac{2}{3}\Delta t, Y^n + \frac{2}{3}\Delta t f(t_n, Y^n)\big)$; note that $y(t_n + \frac{2}{3}\Delta t)$ is approximated by taking an Euler step of length $\frac{2}{3}\Delta t$.

[1] Carl David Tolmé Runge (1856-1927) and Martin Wilhelm Kutta (1867-1944) were German mathematicians

Heun Method

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + \tfrac{2}{3}\Delta t, Y^n + \tfrac{2}{3}k_1) \\ Y^{n+1} &= Y^n + \tfrac{1}{4}k_1 + \tfrac{3}{4}k_2 \end{aligned} \tag{3.14}$$

The general form for an explicit s-stage RK method is given below. The coefficient c_1 is always zero because we always evaluate f at the point (t_n, Y^n) from the previous step to get the appropriate cancellation for the Δt term in the local truncation error calculation.

General s-stage explicit RK method

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + c_2\Delta t, Y^n + a_{21}k_1) \\ k_3 &= \Delta t f(t_n + c_3\Delta t, Y^n + a_{31}k_1 + a_{32}k_2) \\ &\vdots \\ k_s &= \Delta t f\big(t_n + c_s\Delta t, Y^n + a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{ss-1}k_{s-1}\big) \\ Y^{n+1} &= Y^n + \textstyle\sum_{j=1}^{s} b_j k_j \end{aligned} \tag{3.15}$$

Once the stage s is set and the coefficients are determined, the method is completely specified; for this reason, the RK explicit s-stage methods are often described by a **Butcher**[2] **tableau** of the form

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{3.16}$$

As an example, a commonly used four-stage RK method is described by the tableau

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{3.17}$$

[2]Named after John C. Butcher, a mathematician from New Zealand.

which uses an approximation at the point t_n, two approximations at the point $t_n + \Delta t/2$, and the fourth approximation at t_{n+1}. This defines the method

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + \tfrac{1}{2}\Delta t, Y^n + \tfrac{1}{2}k_1) \\ k_3 &= \Delta t f(t_n + \tfrac{1}{2}\Delta t, Y^n + \tfrac{1}{2}k_2) \\ k_4 &= \Delta t f(t_n + \Delta t, Y^n + k_3) \\ Y^{n+1} &= Y^n + \tfrac{k_1}{6} + \tfrac{k_2}{3} + \tfrac{k_3}{3} + \tfrac{k_4}{6}\,. \end{aligned}$$

In the examples of RK methods provided, we note that c_i in the term $t_n + c_i\Delta t$ satisfy the property that $c_i = \sum_{j=1}^{i-1} a_{ij}$ which can be demonstrated rigorously. In addition, the weights b_i satisfy $\sum_{i=1}^{s} b_i = 1$. This can be used as a check in a computer code to confirm the coefficients have been entered correctly.

When implementing RK methods using a fixed time step one could have separate codes for the Euler method, midpoint method, Heun's method, etc. but a more general approach is discussed here. Basically one writes "library" routines which input the coefficients for each s-stage method coded and another routine which advances the solution one time step for any s-stage method; once debugged, these routines never need to be modified so they can be private routines. The user sets the number of stages desired and the problem specific information. Then the driver routine initializes the computation by calling the appropriate routine to set the RK coefficients and then at each time step the routine is called to advance the solution. This code structure can be easily done in an object-oriented manner or simply with conditional statements. Error routines need to be added if the exact solution is known.

The following example compares the results of using explicit RK methods for stages one through four. Note that the numerical rate of convergence matches the stage number for stages one through four but, as we will see, this is not true in general.

Example 3.4. NUMERICAL SIMULATIONS USING EXPLICIT RK METHODS

Consider the IVP

$$y'(t) = ty^2(t) \quad 0 \le t \le 2 \quad y(0) = -1$$

whose exact solution is $y(t) = -2/(t^2+2)$. In the table below we provide numerical results at $t = 2$ using RK methods with stages one through four. The methods were chosen so that they have as high degree of accuracy as possible for the given stage. The global error for each s-stage method at $t = 2$ is tabulated along with the corresponding numerical rates calculated by comparing errors at successive values of the time step.

Δt	stage 1		stage 2		stage 3		stage 4	
	error	rate	error	rate	error	rate	error	rate
$1/5$	$2.38\ 10^{-2}$		$1.36\ 10^{-3}$		$1.29\ 10^{-4}$		$1.17\ 10^{-5}$	
$1/10$	$1.08\ 10^{-2}$	1.14	$3.40\ 10^{-4}$	2.01	$1.48\ 10^{-5}$	3.12	$7.20\ 10^{-7}$	4.02
$1/20$	$5.17\ 10^{-3}$	1.06	$8.38\ 10^{-5}$	2.02	$1.78\ 10^{-6}$	3.05	$4.45\ 10^{-8}$	4.02
$1/40$	$2.53\ 10^{-3}$	1.03	$2.08\ 10^{-5}$	2.01	$2.19\ 10^{-7}$	3.02	$2.77\ 10^{-9}$	4.01

Many RK methods were derived in the early part of the 1900's; initially, the impetus was to find higher order explicit methods. In Example 3.4 we saw that for $s \leq 4$ the stage and the order of accuracy are the same. One might be tempted to generalize that an s-stage method always produces a method with global error $\mathcal{O}(\Delta t^s)$, however, this is *not* the case. In fact, there is an order barrier which is illustrated in the table below. As you can see from the table, a five-stage RK method does not produce a fifth order scheme; we need a six-stage method to produce that accuracy so there is no practical reason to use a five-stage scheme because it has the same accuracy as a four-stage scheme but requires one additional function evaluation.

Order	1	2	3	4	5	6	7	8	9
Min. stage	1	2	3	4	6	7	9	11	11

Unlike explicit RK methods, implicit RK methods do not have this order barrier. The following four-stage implicit RK method has order five so it is more accurate than any four-stage explicit RK method.

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + \tfrac{1}{4}\Delta t, Y^n + \tfrac{1}{8}k_1 + \tfrac{1}{8}k_2) \\ k_3 &= \Delta t f(t_n + \tfrac{7}{10}\Delta t, Y^n - \tfrac{1}{100}k_1 + \tfrac{14}{25}k_2 + \tfrac{3}{20}k_3) \\ k_4 &= \Delta t f\big(t_n + \Delta t, Y^n + \tfrac{2}{7}k_1 + \tfrac{5}{7}k_3\big) \\ Y^{n+1} &= Y^n + \tfrac{1}{14}k_1 + \tfrac{32}{81}k_2 + \tfrac{250}{567}k_3 + \tfrac{5}{54}k_4\,. \end{aligned}$$

Analogous to the general explicit s-stage RK scheme (3.15) we can write a general form of an implicit s-stage RK method. The difference in implicit methods is that in the calculation of k_i the approximation to $y(t_n + c_i\Delta t)$ can be over all values of s whereas in explicit methods the sum only goes through the previous k_j, $j = 1, \cdots, j-1$ terms.

General s-stage implicit RK method

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n + a_{11}k_1 + a_{12}k_2 + \cdots a_{1s}k_s) \\ k_2 &= \Delta t f(t_n + c_2\Delta t, Y^n + a_{21}k_1 + + a_{22}k_2 + \cdots a_{2s}k_s) \\ &\vdots \\ k_s &= \Delta t f(t_n + c_s\Delta t, Y^n + a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{ss}k_s) \\ Y^{n+1} &= Y^n + \textstyle\sum_{j=1}^{s} b_j k_j \end{aligned} \tag{3.18}$$

An implicit RK method has a tableau which is no longer upper triangular

$$\begin{array}{c|cccc} 0 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{3.19}$$

3.1.4 Stability of single-step methods

For stability we want to know that the computed solution to the difference equation remains close to the actual solution of the difference equation and so does not grow in an unbounded manner. We first look at stability of the differential equation for the model problem

$$y'(t) = \lambda y \quad 0 < t \leq T, \lambda \in \mathbb{C}, \tag{3.20}$$

with the initial condition $y(0) = y_0$ and solution $y(t) = y_0 e^{\lambda t}$. Note that in general λ is a complex number but to understand why we look at this particular problem first consider the case when λ is real. If $\lambda > 0$ then small changes in the initial condition can result in the solutions becoming far apart. For example, if we have IVPs (3.20) with two initial conditions $y_1(0) = \alpha$ and $y_2(0) = \beta$ which differ by $\delta = |\beta - \alpha|$ then the solutions $y_1 = \alpha e^{\lambda t}$ and $y_2 = \beta e^{\lambda t}$ differ by $\delta e^{\lambda t}$. Consequently, for large $\lambda > 0$ these solutions can differ dramatically as illustrated in the table below for various choices of δ and λ. However, if $\lambda < 0$ the term $\delta e^{\lambda t}$ approaches zero as $t \to 0$. Therefore for stability of this model IVP when λ is real we require $\lambda < 0$.

λ	δ	$\lvert y_1(0.5) - y_2(0.5)\rvert$	$\lvert y_1(1) - y_2(1)\rvert$	$\lvert y_1(10) - y_2(10)\rvert$
1	0.01	0.0165	0.0272	220
1	0.1	0.165	0.272	2203
10	0.01	1.48	220	10^{41}
10	0.1	14.8	2203	10^{42}
-1	0.1	$6.07\ 10^{-2}$	$3.68\ 10^{-2}$	$4.54\ 10^{-6}$
-10	0.1	$6.73\ 10^{-4}$	$4.54\ 10^{-6}$	10^{-45}

In general, λ is complex so it can be written as $\lambda = \alpha + i\beta$ where α, β are real numbers and $i = \sqrt{-1}$. The exact solution is

$$y(t) = y_0 e^{\lambda t} = y_0 e^{\alpha t + i\beta t} = y_0 e^{\alpha t} e^{i\beta t}\,.$$

Now $e^{i\beta t} = \cos(\beta t) + i\sin(\beta t)$ so this term does not grow in time; however the term $e^{\alpha t}$ will grow in an unbounded manner if $\alpha > 0$. Consequently we say that the differential equation $y' = \lambda y$ is stable when the real part of λ is less than or equal to zero, i.e., $\mathrm{Re}(\lambda) \leq 0$ or λ is in the left half of the complex plane.

When we approximate the model IVP (3.20) we want to know that small changes, such as those due to round off, do not cause large changes in the solution. Here we are going to look at stability of a difference equation of the form

$$Y^{n+1} = \zeta(\lambda\Delta t)Y^n \tag{3.21}$$

applied to the model problem (3.20). Our single step methods fit into this framework. For example, for the forward Euler method applied to the differential equation $y' = \lambda y$ we have $Y^{n+1} = Y^n + \Delta t\lambda Y^n$ so $\zeta(\lambda\Delta t) = 1 + \Delta t\lambda$. For the backward Euler method we have $Y^{n+1} = Y^n + \Delta t\lambda Y^{n+1}$ so $\zeta(\lambda\Delta t) = 1/(1 - \lambda\Delta t)$. For explicit RK methods $\zeta(z)$ will be a polynomial in z and for implicit RK methods it will be a rational function.

We apply the difference equation (3.21) recursively to get

$$Y^n = \zeta(\lambda\Delta t)Y^{n-1} = \zeta^2(\lambda\Delta t)Y_{i-2} = \cdots = \zeta^i(\lambda\Delta t)Y_0$$

so we can view ζ as an **amplification factor** because the solution at time t_{n-1} is amplified by a factor of ζ to get the solution at t_n, the solution at time t_{n-2} is amplified by a factor of ζ^2 to get the solution at t_n, etc. We know that the magnitude of ζ must be less than or equal to one or else Y^n will become unbounded. This condition is known as **absolute stability**. There are many other definitions of different types of stability; some of these are explored in the exercises.

The region of absolute stability for the difference equation (3.21) is $\{\lambda\Delta t \in C \mid |\zeta(\lambda\Delta t)| \leq 1\}$. A method is called A**-stable** if $|\zeta(\lambda\Delta t)| \leq 1$ for the entire left half plane.

Example 3.5. We want to determine if the forward Euler method and the backward Euler method are A-stable; if not, we want to determine the region of absolute stability. We then discuss our previous numerical results for $y'(t) = -20y(t)$ in light of these results.

For the forward Euler method $\zeta(\lambda\Delta t) = 1 + \lambda\Delta t$ so the condition for A-stability is that $|1 + \lambda\Delta t| \leq 1$ for the entire left plane. Now λ is, in general, complex which we can write as $\lambda = \alpha + i\beta$ but let's first look at the real case, i.e., $\beta = 0$. Then we have

$$-1 \leq 1 + \lambda\Delta t \leq 1 \Rightarrow -2 \leq \lambda\Delta t \leq 0$$

so on the real axis we have the interval $[-2, 0]$. This says that for a fixed real $\lambda < 0$, Δt must satisfy $\Delta t \leq 2/|\lambda|$ and thus the method is not A-stable but has a region $[-2, 0]$ of absolute stability if λ is real. If $\beta \neq 0$ then we have the region of stability as a circle in the complex plane of radius one centered at -1. For example, when $\lambda = -20$ Δt must satisfy $\Delta t \leq 0.1$. In Example 2.8 we plotted results for $\Delta t = 1/4$ and $1/8$ which do not satisfy the stability criteria. In the figure below we plot approximations to the same problem using $\Delta t = 1/20$, $1/40$ and $1/60$. As you can see from the graph, the solution appears to be converging.

For the backward Euler method $\zeta(\lambda\Delta t) = 1/(1-\lambda\Delta t)$. To determine if it is A-stable we see if it satisfies the stability criteria for the entire left plane. As before, we first find the region when λ is real. For $\lambda \leq 0$ have $1 - \lambda\Delta t \geq 1$ so that $\zeta(\lambda\Delta t) \leq 1$ for all Δt and we have the entire left plane. The backward Euler method is A-stable so any choice of Δt provides stable results for $\lambda < 0$.

To be precise, the region of absolute stability for the backward Euler method is actually the region outside the circle in the complex plane centered at one with radius one. Clearly, this includes the left half plane. To see this, note that when $\lambda\Delta t \geq 2$ then $|1/(1-\lambda\Delta t)| \leq 1$. However, we are mainly interested in the case when $Re(\lambda) < 0$ because the differential equation $y'(t) = \lambda y$ is stable.

Example 3.6. In this example we investigate the regions of absolute stability for the explicit 2-stage Heun method

$$Y^{n+1} = Y^n + \frac{\Delta t}{4}\big[f(t_n, Y^n) + 3f\big(t_n + \frac{2}{3}\Delta t, Y^n + \frac{2}{3}\Delta t f(t_n, Y^n)\big)\big]\,.$$

We have written the scheme as a single equation rather than the standard way of specifying k_i because this makes it easier to determine the amplification factor.

We apply the difference scheme to the model problem $y' = \lambda y$ where $f(t, y) = \lambda y(t)$ to get

$$Y^{n+1} = Y^n + \frac{\Delta t}{4}\big[\lambda Y^n + 3\lambda(Y^n + \frac{2}{3}\Delta t\lambda Y^n)\big] = \big[1 + \frac{1}{4}(\lambda\Delta t) + \frac{3}{4}(\lambda\Delta t) + \frac{1}{2}(\lambda\Delta t)^2\big]Y^n$$

so $\zeta(\lambda\Delta t) = 1 + \lambda\Delta t + \frac{1}{2}(\lambda\Delta t)^2$. The region of absolute stability is all points z in the complex plane where $|\zeta(z)| \leq 1$. If λ is real and non-positive we have

$$-1 \leq 1 + z + \frac{z^2}{2} \leq 1 \Rightarrow -2 \leq z(1 + \frac{z}{2}) \leq 0\,.$$

For $\lambda \leq 0$ so that $z = \lambda\Delta t \leq 0$ we must have $1 + \frac{1}{2}\lambda\Delta t \geq 0$ which says $\Delta t\lambda \geq -2$. Thus the region of stability is $[-2, 0]$ when λ is real and when it is complex we have a circle of radius one centered at -1. This is the same region as the one computed for the forward Euler method.

In Figure 3.1 numerical results are presented for the case when $\lambda = -20$. For this choice of λ the stability criteria becomes $\Delta t \leq 0.1$.

It can be shown that there is no *explicit* RK method that has an unbounded region of absolute stability such as the left half plane region of stability that we got for the backward Euler method. In general, implicit methods do not have stability restrictions so this is one reason that we need implicit methods. Implicit methods will be especially important when we study initial boundary value problems.

3.2 Multistep Methods

An m-step multistep method uses previously calculated information at the m points $t_n,\ t_{n-1}, \ldots,\ t_{n-(m-1)}$ to approximate the solution at t_{n+1} whereas a one-step

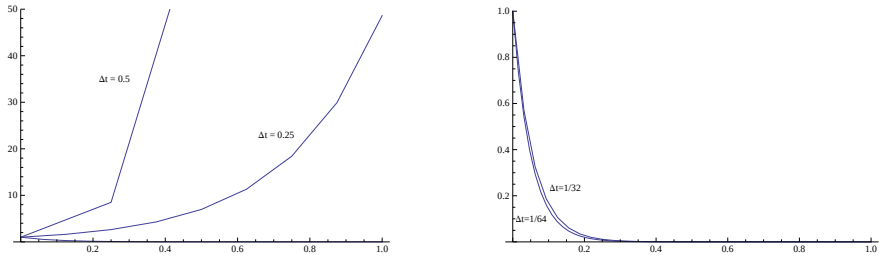

Figure 3.1: Approximations for the IVP $y'(t) = -20y$, $y(0) = 1$ using the Heun method. For the plot on the left the step size Δt is too large and numerical instability is occurring. When we reduce the step size the method converges as the graph on the right demonstrates.

method uses only the information at t_n plus additional approximations in the interval $[t_n, t_{n+1}]$ which are then discarded. This, of course, means that multistep methods require fewer function evaluations than single-step methods. However, because information from previous time steps is needed to advance the solution to t_{n+1}, these values must be stored. This is not an issue when we are solving a single IVP but when we have a very large system of IVPs the storage of information from previous steps is significant. Another issue with multistep methods is that they are not self-starting. Recall that when we implement a RK method we use the initial condition and then the scheme gives the approximation at $t_0 + \Delta t$ and subsequent points. However, if we are using say a three-step method then we need the initial condition at t_0 plus approximations at t_1 and t_2 to start using the method. So a shortcoming of m-step methods is that we have to use a one-step method to get approximations at the first $m - 1$ time steps after t_0.

An m-step method can be implicit or explicit. The solution at t_{n+1} can depend explicitly on the solution and the slope at previous points but the most common methods only use the solution at t_n and the slopes at all points. To write the general form of an m-step method which can be explicit or implicit we allow the scheme to be a linear combination of the solution at the m points t_n, t_{n-1}, $\ldots$, $t_{n-(m-1)}$ and the slopes at the $m+1$ points t_{n+1}, t_n, t_{n-1}, $\ldots$, $t_{n-(m-1)}$ points. If the method is explicit then the coefficient in front of the term $f(t_{n+1}, Y^{n+1})$ is zero.

General m-step method

$$\begin{aligned} Y^{n+1} \quad = \quad & a_{m-1}Y^n + a_{m-2}Y^{n-1} + a_{m-3}Y_{n-2} + \cdots + a_0 Y_{n+1-m} \\ & + \Delta t \Big[b_m f(t_{n+1}, Y^{n+1}) + b_{m-1} f(t_n, Y^n) + b_{m-2} f(t_{n-1}, Y^{n-1}) \\ & \quad + \cdots + b_0 f(t_{n+1-m}, Y^{n+1-m}) \Big]. \end{aligned} \tag{3.22}$$

If $b_m = 0$ then the method is explicit; otherwise it is implicit. We don't include a term $a_m Y^{n+1}$ on the right-hand side for implicit methods because if we did we

could just combine it with the Y^{n+1} term on the left-hand side of the formula and then divide by that coefficient to get the form in (3.22).

In this section we begin by looking at xxxxxxx

3.2.1 Derivation of multistep methods

We saw that a common approach to deriving one-step methods (other than Taylor series methods) is to integrate the differential equation from t_n to t_{n+1} and use a quadrature rule to approximate the integral over $f(t, y)$. However, for an m-step method we must integrate from t_{n-m+1} to t_{n+1}. For example, for a two-step method we integrate the equation from t_{n-1} to t_{n+1} to get

$$\int_{t_{n-1}}^{t_{n+1}} y'(t)\,dt = y(t_{n+1}) - y(t_{n-1}) = \int_{t_{n-1}}^{t_{n+1}} f(t, y)\,dt\,.$$

Now if we use the midpoint quadrature rule to approximate the integral we have the two-step scheme

$$Y^{n+1} = Y^{n-1} + 2\Delta t f(t_n, Y^n) \tag{3.23}$$

which is sometimes called the **modified midpoint method**. However, unlike one-step methods, our choice of quadrature rule is restricted because for the quadrature points we must use only the previously calculated times. For example, if we have a three-step method using t_n, t_{n-1}, and t_{n-2} we need to use a Newton-Cotes integration formula such as Simpson's method. Remember that Newton-Cotes quadrature rules are interpolatory and so this approach is closely related to using an interpolation polynomial.

Multistep methods are typically derived by using an interpolating polynomial in either of two ways. The first is to approximate $y(t)$ by an interpolating polynomial through t_n, $t_{n-1}, \ldots, t_{n-m+1}$ and then differentiate it, evaluate at t_{n+1} for an implicit method and set it equal to the given slope at that point to obtain the difference equation. This gives rise to a family of implicit methods called **backward difference formulas**. The second approach is to use an interpolating polynomial through t_n, $t_{n-1}, \ldots, t_{n-m+1}$ for the given slope $f(t, y)$ and then integrate the equation; the integral of the interpolating polynomial can be computed exactly. We discuss both approaches here.

Similar to one-step methods, we can also derive multistep methods by assuming the most general form of the m-step method and then determine the constraints on the coefficients which give as high an order of accuracy as possible. This approach is just the method of undetermined coefficients discussed in § 3.1.2. This approach for deriving multistep methods is explored in the exercises.

Using an interpolating polynomial to approximate the solution

Backward difference formulas (BDFs) are a family of implicit multistep methods; the backward Euler method is considered the first order BDF even though it is a single step method. We begin by demonstrating how to derive the backward Euler method by approximating $y(t)$ by a linear interpolating polynomial and then show

how this approach can be used to generate more accurate methods by simply using a higher degree interpolating polynomial.

The Lagrange form of the unique linear polynomial that passes through the points $\big(t_n, y(t_n)\big)$ and $\big(t_{n+1}, y(t_{n+1})\big)$ is

$$p_1(t) = y(t_n)\frac{t - t_{n+1}}{-\Delta t} + y(t_{n+1})\frac{t - t_n}{\Delta t}\,.$$

Differentiating with respect to t gives

$$p_1'(t) = \frac{-1}{\Delta t}y(t_n) + \frac{1}{\Delta t}y(t_{n+1})$$

which leads to the familiar approximation

$$y'(t) \approx \frac{y(t_{n+1}) - y(t_n)}{\Delta t}\,.$$

Using this expression in the differential equation $y'(t) = f(t, y)$ at t_{n+1} gives the implicit backward Euler method.

For the second order BDF we approximate $y(t_{n+1})$ by the quadratic polynomial that passes through $\big(t_{n-1}, y(t_{n-1})\big)$,$\big(t_n, y(t_n)\big)$ and $\big(t_{n+1}, y(t_{n+1})\big)$; the Lagrange form of the polynomial is

$$\begin{aligned} p_2(t) \;=\; & y(t_{n-1})\frac{(t - t_n)(t - t_{n+1})}{(t_{n-1} - t_n)(t_{n-1} - t_{n+1})} + y(t_n)\frac{(t - t_{n-1})(t - t_{n+1})}{(t_n - t_{n-1})(t_n - t_{n+1})} \\ & + y(t_{n+1})\frac{(t - t_{n-1})(t - t_n)}{(t_{n+1} - t_{n-1})(t_{n+1} - t_n)} \end{aligned}$$

and differentiating with respect to t and assuming a constant Δt gives

$$p_2'(t) = \frac{y(t_{n-1})}{2(\Delta t)^2}\big[2t - t_n - t_{n+1}\big] - \frac{y(t_n)}{(\Delta t)^2}\big[2t - t_{n-1} - t_{n+1}\big] + \frac{y(t_{n+1})}{2(\Delta t)^2}\big[2t - t_{n-1} - t_n\big]\,.$$

We use $p_2'(t_{n+1})$ as an approximation to $y'(t_{n+1})$ in the equation $y'(t_{n+1}) = f\big(t_{n+1}, y(t_{n+1})\big)$ to get

$$\frac{y(t_{n-1})}{2(\Delta t)^2}\Delta t - \frac{y(t_n)}{(\Delta t)^2}2\Delta t + \frac{y(t_{n+1})}{2(\Delta t)^2}3\Delta t \approx f\big(t_{n+1}, y(t_{n+1})\big)\,.$$

This suggest the BDF

$$\frac{3}{2}Y^{n+1} - 2Y^n + \frac{1}{2}Y^{n-1} = \Delta t f(t_{n+1}, Y^{n+1})\,;$$

often these formulas are normalized so that the coefficient of Y^{n+1} is one. It can be shown that this method is second order.

Second order BDF $$Y^{n+1} = \frac{4}{3}Y^n - \frac{1}{3}Y^{n-1} + \frac{2}{3}\Delta t f(t_{n+1}, Y^{n+1}) \quad (3.24)$$

In general, BDF formulas using approximations at t_{n+1}, $t_n, \cdots, t_{n+1-m}$ have the general normalized form

$$Y^{n+1} = \sum_{j=1}^{m} a_{mj} Y^{(n+1)-j} + \beta \Delta t f(t_{n+1}, Y^{n+1}) \,. \quad (3.25)$$

For the two-step scheme (3.24) we have $m = 2$, $a_{21} = 4/3$, $a_{22} = -1/3$ and $\beta = 2/3$. Table 3.1 gives coefficients for other uniform BDF formulas using the terminology of (3.25). It can be proved that the accuracy of the m-step methods included in the table is m.

m-step	a_{m1}	a_{m2}	a_{m3}	a_{m4}	a_{m5}	β
1	1					1
2	4/3	-1/3				2/3
3	18/11	-9/11	2/11			6/11
4	48/25	-36/25	16/25	-3/25		12/25
5	300/137	-300/137	200/137	-75/137	12/137	60/137

Table 3.1: Coefficients for implicit BDF formulas of the form (3.25) where the coefficient of Y^{n+1} is normalized to one.

It is also possible to derive BDFs for nonuniform time steps. The formulas are derived in an analogous manner but are a bit more complicated because for the interpolating polynomial we must keep track of each Δt_i; in the case of a uniform Δt there are some cancellations which simplify the resulting formulas.

Using an interpolating polynomial to approximate the slope

The second choice for deriving schemes using an interpolating polynomial is to approximate $f(t, y)$ by an interpolating polynomial and then integrate. For example, suppose we approximate $f(t, y)$ by a polynomial of degree zero, i.e., a constant in the interval $[t_n, t_{n+1}]$. If we use the approximation $f(t, y) \approx f\big(t_n, y(t_n)\big)$ in $[t_n, t_{n+1}]$ then integrating the differential equation yields

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y)\, dt \approx \int_{t_n}^{t_{n+1}} f\big(t_n, y(t_n)\big)\, dt = f\big(t_n, y(t_n)\big)\Delta t$$

which leads to the forward Euler method. If we choose to approximate $f(t, y)$ in $[t_n, t_{n+1}]$ by $f\big(t_{n+1}, y(t_{n+1})\big)$ then we get the backward Euler method. In general,

if the interpolation polynomial approximating $f(t,y)$ includes the point t_{n+1} then the resulting scheme will be implicit because it will involve $f(t_{n+1},Y^{n+1})$; otherwise it will be explicit.

To see how to derive a two-step explicit scheme, we use the previous information at t_n and t_{n-1} and write the linear interpolating polynomial for $f(t,y)$ through the two points and integrate the equation from t_n to t_{n+1}. As before we use the Fundamental Theorem of Calculus to integrate $\int_{t_n}^{t_{n+1}} y'(t)\,dt$. Using uniform step sizes, we have

$$\begin{aligned} y(t_{n+1})-y(t_n) &\approx \int_{t_n}^{t_{n+1}} \left[f\big(t_{n-1},y(t_{n-1})\big)\frac{t-t_n}{-\Delta t} + f\big(t_n,y(t_n)\big)\frac{t-t_{n-1}}{\Delta t}\right] dt \\ &= -\frac{1}{\Delta t} f\big(t_{n-1},y(t_{n-1})\big)\frac{(t-t_n)^2}{2}\Big|_{t_n}^{t_{n+1}} \\ &\qquad +\frac{1}{\Delta t} f\big(t_n,y(t_n)\big)\frac{(t-t_{n-1})^2}{2}\Big|_{t_n}^{t_{n+1}} \\ &= -\frac{1}{\Delta t} f\big(t_{n-1},y(t_{n-1})\big)\left(\frac{(\Delta t)^2}{2}\right) + \frac{1}{\Delta t} f\big(t_n,y(t_n)\big)\frac{3\Delta t^2}{2} \end{aligned}$$

which suggests the scheme

$$Y^{n+1} = Y^n + \frac{3}{2}\Delta t f\big(t_n,Y^n\big) - \frac{\Delta t}{2} f\big(t_{n-1},Y^{n-1}\big)\,. \tag{3.26}$$

This is an example of an Adams-Bashforth multistep method; these methods will be discussed in more detail in § 3.2.2.

3.2.2 Adams-Bashforth and Adams-Moulton families

A commonly used family of explicit multistep methods are called **Adams-Bashforth methods** which use the derivative f evaluated at m prior points (including t_n) but only use the approximation to $y(t)$ at t_n; i.e., $a_0 = \cdots = a_{m-2} = 0$. The one step Adams-Bashforth method is the forward Euler method. In § 3.2.1 we used an interpolation polynomial for $f(t,y)$ to derive the 2-step scheme

$$Y^{n+1} = Y^n + \frac{3}{2}\Delta t f\big(t_n,Y^n\big) - \frac{\Delta t}{2} f\big(t_{n-1},Y^{n-1}\big)$$

which belongs to the Adams-Bashforth family with $b_2 = 0$, $b_1 = 3/2$ and $b_0 = -1/2$ in the general formula (3.22). In the exercises, you are asked to rigorously demonstrate that the local truncation error for (3.26) is third order and thus the scheme is second order accurate. The methods up to five steps are listed here for completeness.

Adams-Bashforth 2-step, 3-step, 4-step and 5-step methods

$$
\begin{aligned}
Y^{n+1} &= Y^n + \Delta t\left(\frac{3}{2}f(t_n,Y^n) - \frac{1}{2}f(t_{n-1},Y^{n-1})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{23}{12}f(t_n,Y^n) - \frac{4}{3}f(t_{n-1},Y^{n-1}) + \frac{5}{12}f(t_{i-2},Y_{i-2})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{55}{24}f(t_n,Y^n) - \frac{59}{24}f(t_{n-1},Y^{n-1}) + \frac{37}{24}f(t_{i-2},Y_{i-2})\right.\\
&\qquad\left. -\frac{3}{8}f(t_{i-3},Y_{i-3})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{1901}{720}f(t_n,Y^n) - \frac{1387}{360}f(t_{n-1},Y^{n-1}) + \frac{109}{30}f(t_{i-2},Y_{i-2})\right.\\
&\qquad\left. -\frac{637}{360}f(t_{i-3},Y_{i-3}) + \frac{251}{720}f(t_{i-4},Y_{i-4})\right)
\end{aligned}
\tag{3.27}
$$

Schemes in the *Adams-Moulton* family are implicit multistep methods which use the derivative f evaluated at t_{n+1} plus m prior points but only use the solution Y^n. The one-step Adams-Moulton method is the backward Euler scheme and the 2-step method is the trapezoidal rule; several methods are listed here for completeness.

Adams-Moulton 2-step, 3-step, 4-step and 5-step methods

$$
\begin{aligned}
Y^{n+1} &= Y^n + \frac{\Delta t}{2}\big(f(t_{n+1},Y^{n+1}) + f(t_n,Y^n)\big)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{5}{12}f(t_{n+1},Y^{n+1}) + \frac{2}{3}f(t_n,Y^n) - \frac{1}{12}f(t_{n-1},Y^{n-1})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{3}{8}f(t_{n+1},Y^{n+1}) + \frac{19}{24}f(t_n,Y^n) - \frac{5}{24}f(t_{n-1},Y^{n-1})\right.\\
&\qquad\left. +\frac{1}{24}f(t_{i-2},Y_{i-2})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{251}{720}f(t_{n+1},Y^{n+1}) + \frac{646}{720}f(t_n,Y^n) - \frac{264}{720}f(t_{n-1},Y^{n-1})\right.\\
&\qquad\left. +\frac{106}{720}f(t_{i-2},Y_{i-2}) - \frac{19}{720}f(t_{i-3},Y_{i-3})\right)
\end{aligned}
\tag{3.28}
$$

One drawback of an m-step method is that we need m starting values Y^0, $Y^1,\ldots,Y^{m-1}$ and we only have Y^0 from the initial condition. Typically one uses a one-step method to start the scheme. How do we decide what method to use? A "safe" approach is to use a method which has the same accuracy as the multistep method but we will see in the following examples that you can actually use a method

which has one power of Δt less because we are only taking a small number of steps with the method. For example, if we use the 2-step second order Adams-Bashforth method we need Y^1 in addition to Y^0. If we take one step with the forward Euler method it is actually second order accurate at the first step because the error there is only due to the local truncation error. However, if we use a 3-step third order Adams-Bashforth method then using the forward Euler method to get the two starting values results in a loss of accuracy. This issue is illustrated in the following examples.

Example 3.7. STARTING VALUES FOR MULTISTEP METHODS

In this example we implement the 3-step third order accurate Adams-Bashforth method given in (3.27) to solve the IVP

$$y'(t) = t^2 + y(t) \quad 2 < t < 5 \qquad y(2) = 1\,,$$

which has the exact solution

$$y(t) = 11e^{t-2} - (t^2 + 2t + 2)\,.$$

We compare the numerical rates of convergence when we use different methods to generate the starting values. Specifically we use RK methods of order one through four to generate the starting values which for a 4-step method are Y^1, Y^2, and Y^3 because we have $Y^0 = 1$. The results are tabulated below for the errors at $t = 3$. As you can see from the table, if a second, third or fourth order scheme is used to compute the starting values then the method is third order. There is nothing gained by using a higher order scheme (the fourth order) for the starting values. However, if a first order scheme (forward Euler) is used then the rate is degraded to second order even though we only used it to calculate two values, Y^1 and Y_2. Consequently to compute starting values we should use a scheme that has the same overall accuracy or one degree less than the method we are using.

	accuracy of starting method							
	first		second		third		fourth	
Δt	error	rate	error	rate	error	rate	error	rate
$1/10$	$2.425\ 10^{-1}$		$1.618\ 10^{-2}$		$8.231\ 10^{-3}$		$8.042\ 10^{-3}$	
$1/20$	$6.106\ 10^{-2}$	1.99	$2.241\ 10^{-3}$	2.87	$1.208\ 10^{-3}$	2.77	$1.195\ 10^{-3}$	2.75
$1/40$	$1.529\ 10^{-2}$	2.00	$2.946\ 10^{-4}$	2.92	$1.628\ 10^{-4}$	2.89	$1.620\ 10^{-4}$	2.88
$1/80$	$3.823\ 10^{-3}$	2.00	$3.777\ 10^{-5}$	2.96	$2.112\ 10^{-5}$	2.95	$2.107\ 10^{-5}$	2.94

Example 3.8. COMPARISON OF ADAMS BASHFORTH METHODS

In this example we solve the IVP from the previous example a by 2-step through 5-step Adams Bashforth method. In each case we use a scheme that is one degree less accurate to calculate the starting values. As can be seen from the table, all methods have the expected numerical rate of convergence.

	2-step method		3-step method		4-step method		5-step method	
Δt	error	rate	error	rate	error	rate	error	rate
$1/10$	$2.240\ 10^{-1}$		$1.618\ 10^{-2}$		$9.146\ 10^{-4}$		$5.567\ 10^{-5}$	
$1/20$	$5.896\ 10^{-2}$	1.93	$2.241\ 10^{-3}$	2.87	$6.986\ 10^{-5}$	3.71	$2.463\ 10^{-6}$	4.50
$1/40$	$1.509\ 10^{-2}$	1.97	$2.946\ 10^{-4}$	2.92	$4.802\ 10^{-6}$	3.86	$8.983\ 10^{-8}$	4.78
$1/80$	$3.816\ 10^{-3}$	1.98	$3.777\ 10^{-5}$	2.96	$3.144\ 10^{-7}$	3.93	$3.022\ 10^{-9}$	4.89

3.2.3 Stability of multistep methods

The numerical stability of a one-step method depends on the initial condition y_0 but in a m-step multistep method there are $m-1$ other starting values $Y^1, Y^2, \ldots, Y^{m-1}$ which are obtained by another method such as a RK method. In 1956 Dahlquist[3] published a seminal work formulating criteria for the stability of linear multistep methods. We will give an overview of the results here.

We first rewrite the m-step multistep method (3.22) by shifting the indices to get

$$\begin{aligned} Y^{i+m} &= a_{m-1}Y^{i+m-1} + a_{m-2}Y^{i+m-2} + a_{m-3}Y^{i+m-3} + \cdots + a_0 Y^i \\ &\quad + \Delta t\Big[b_m f(t_{i+m}, Y^{i+m}) + b_{m-1} f(t_{i+m-1}, Y^{i+m-1}) \\ &\quad + b_{m-2} f(t_{i+m-2}, Y^{i+m-2}) + \cdots + b_0 f(t_i, Y^i)\Big] \end{aligned}$$

or equivalently

$$Y^{i+m} - \sum_{j=0}^{m-1} a_j Y^{i+j} = \Delta t \sum_{j=0}^{m} b_j f(t_{i+j}, Y^{i+j})\,.$$

As before, we apply it to the model IVP $y' = \lambda y$, $y(0) = y_0$ for $Re(\lambda) < 0$ which guarantees the IVP itself is stable. Substituting $f = \lambda y$ into the difference equation gives

$$Y^{i+m} - \sum_{j=0}^{m-1} a_j Y^{i+j} = \Delta t \sum_{j=0}^{m} b_j \lambda Y^{i+j}\,.$$

Recall that a technique for solving a linear homogeneous ODE such as $y''(t) + 2y'(t) - y(t) = 0$ is to look for solutions of the form $y = e^{rt}$ and get a polynomial equation for r such as $e^{rt}(r^2 + 2r - 1) = 0$ and then determine the roots of the equation. We take the analogous approach for the difference equation and seek a solution of the form $Y^n = z^i$. Substitution into the difference equation yields

$$z^{i+m} - \sum_{j=0}^{m-1} a_j z^{i+j} = \Delta t \sum_{j=0}^{m} b_j \lambda z^{i+j}\,.$$

[3]Germund Dahlquist (1925-2005) was a Swedish mathematician.

Canceling the lowest order term z^i gives a polynomial equation in z which is a function of λ and Δt resulting in the stability equation

$$Q(\lambda\Delta t) = z^m - \sum_{j=0}^{m-1} a_j z^j - \Delta t \sum_{j=0}^{m} b_j \lambda z^j = \rho(z) - \Delta t \lambda \sigma(z)\,,$$

where

$$\rho(z) = z^m - \sum_{j=0}^{m-1} a_j z^j \quad \text{and} \quad \sigma(z) = \sum_{j=0}^{m} b_j z^j\,. \tag{3.29}$$

For stability, we need the roots of $\rho(z)$ to have magnitude ≤ 1 and if a root is identically one then it must be a simple root. If this root condition is violated, then the method is unstable so a simple check is to first see if the root condition is satisfied; if the root condition is satisfied then we need to find the region of stability. To do this, we find the roots of $\mathcal{Q}(\lambda\Delta t)$ and require that each root has magnitude less than or equal to one. To simplify the calculations we rewrite $\mathcal{Q}(\lambda\Delta t)$ as

$$\begin{aligned}\mathcal{Q}(\lambda\Delta t) &= z^m(1-\lambda\Delta t b_m) - z^{m-1}(a_{m-1} + b_{m-1}\lambda\Delta t) \\ &\quad -z^{m-2}(a_{m-2} + b_{m-2}\lambda\Delta t) - \cdots - (a_0 + b_0\lambda\Delta t)\,.\end{aligned}$$

The following two examples determine the region of stability using this approach.

Example 3.9. In this example we investigate the stability of the forward and backward Euler methods by first demonstrating that the root condition for $\rho(z)$ is satisfied and then finding the region of absolute stability; we confirm that we get the same results as before.

The forward Euler method is written as $Y^{n+1} = Y^n + \Delta t f(t_n, Y^n)$ so in the form of a multistep method with $m = 1$ we have $a_0 = 1$, $b_0 = 1$, $b_1 = 0$ and thus $\rho(z) = z - 1$ whose root is $z = 1$ so the root condition is satisfied. To find the region of absolute stability we have $\mathcal{Q}(\lambda\Delta t) = z - (1 + \lambda\Delta t)$ which has a single root $1 + \lambda\Delta t$; thus the region of absolute stability is $|1 + \lambda\Delta t| \leq 1$ which is the condition we got before by analyzing the method as a single step method.

For the backward Euler method $a_0 = 1$, $b_0 = 0$, $b_1 = 1$ and so $\rho(z) = z - 1$ which has the root $z = 1$ and so the root condition is satisfied. To find the region of absolute stability we have $\mathcal{Q}(\lambda\Delta t) = z(1 - \lambda\Delta t) - 1$ which has a single root $1/(1 - \lambda\Delta t)$ and we get the same restriction that we got before by analyzing the method as a single-step method.

Example 3.10. In this example we want to show that the 2-step Adams-Bashforth method

$$Y^{n+1} = Y^n + \frac{\Delta t}{2}\left[3f(t_n, Y^n) - f(t_{n-1}, Y^{n-1})\right]$$

is stable.

For this Adams-Bashforth method we have $m = 2$, $a_0 = 0$, $a_1 = 1$, $b_0 = -1/2$, $b_1 = 3/2$, and $b_2 = 0$. The characteristic polynomial is $\rho(z) = z^2 - z = z(z - 1)$ whose two roots are $z = 0, 1$ and the root condition is satisfied.

In summary, we have seen that some methods can be unstable if the step size Δt is too large (such as the forward Euler method) while others are stable even for a large choice of Δt (such as the backward Euler method). In general, explicit methods have stability restrictions whereas implicit methods are stable for all step sizes. Of course, one must have a small enough step size for accuracy. We have just touched on the ideas of stability of numerical methods for IVPs; the interested reader is referred to standard texts in numerical analysis for a thorough treatment of stability. The important concept is that we need a consistent and stable method to guarantee convergence of our results.

3.3 Extrapolation methods

Richardson extrapolation is a technique used throughout numerical analysis. The basic idea is that you take a sequence of approximations which are generated by a method whose error can be expanded in terms of powers of a discretization parameter and then combine these approximations to generate a more accurate solution. Recall that when we calculate the local truncation error we expand the error in terms of the step size Δt so the methods we have studied can be used with Richardson extrapolation. This approach can also be viewed as interpolating the approximations and then extrapolating to the point where the parameter is zero. A currently popular method for solving an IVP where high accuracy is required is the Burlisch-Stoer algorithm which refines this extrapolation concept to provide a robust and efficient algorithm.

In this section we demonstrate the basic idea of how extrapolation can be used with approximations generated by methods we already have. Then we briefly discuss the modifications needed to develop the Burlisch-Stoer method.

3.3.1 Richardson extrapolation

As a simple example consider the forward Euler method which we know is a first order approximation. To simplify the notation we set h to be the time step or grid spacing. From the Taylor Series expansion for $f(t+h)$ we have the forward difference approximation to $f'(t)$ which we denote by $N(h)$ where $N(h) = \big(f(t+h) - f(t)\big)/h$. We have

$$f'(t) - N(h) = \frac{h}{2} f''(t) + \frac{h^2}{3!} f'''(t) + \frac{h^3}{4!} f''''(t) + \cdots . \tag{3.30}$$

Now if we generate another approximation using step size $h/2$ we have

$$f'(t) - N(\frac{h}{2}) = \frac{h}{4} f''(t) + \frac{h^2}{4 \cdot 3!} f'''(t) + \frac{h^3}{8 \cdot 4!} f''''(t) + \cdots . \tag{3.31}$$

The goal is to combine these approximations to eliminate the $\mathcal{O}(h)$ term so that the approximation is $\mathcal{O}(h^2)$. Clearly subtracting (3.30) from twice (3.31) eliminates

the terms involving h so we get

$$\begin{aligned} f'(t) - \big[2N(\frac{h}{2}) - N(h)\big] &= [\frac{h^2}{2 \cdot 3!} - \frac{h^2}{3!}] f'''(t) + [\frac{h^3}{4 \cdot 4!} - \frac{h^3}{4!}] f''''(t) + \cdots + \\ &= -\frac{h^2}{12} f'''(t) - \frac{3h^3}{32} f''''(t) + \cdots . \end{aligned} \tag{3.32}$$

Thus the approximation $2N(h/2) - N(h)$ for $f'(x)$ is second order. This process can be repeated to eliminate the $\mathcal{O}(h^2)$ term. To see this, we use the approximation (3.32) with h halved again to get

$$f'(t) - \big[2N(\frac{h}{4}) - N(\frac{h}{2})\big] = -\frac{h^2}{4 \cdot 12} f'''(t) - \frac{3h^3}{8 \cdot 24} f''''(t) + \cdots . \tag{3.33}$$

To eliminate the h^2 terms we need to take four times (3.33) and subtract (3.32) so that $3f'(t) - [8N(\frac{h}{4}) - 4N(\frac{h}{2}) - 2N(\frac{h}{2}) + N(h)] = \mathcal{O}(h^3)$. This yields the approximation $\frac{8}{3}N(\frac{h}{4}) - 2N(\frac{h}{2}) + \frac{1}{3}N(h)$ which is a third order approximation to $f'(t)$. In theory, this procedure can be repeated to get as accurate an approximation as possible for the given computer precision. In this simple example we have used approximations at $h, h/2, h/4, \ldots$ but a general sequence $h_0, h_1, h_2, \ldots$ where $h_0 > h_1 > h_2 > \cdots$ can be used. The following example takes first order approximations generated by the forward Euler method and produces approximations of a specified order.

Example 3.11. FORWARD EULER WITH RICHARDSON EXTRAPOLATION

Return to Example 2.5 where we have tabulated the approximate solution obtained by the forward Euler method for the IVP $y'(t) = -5y(t)$ with $y(0) = 2$. The exact solution at $t = 1$ is 0.0134759. We tabulate the results from the problem below and then use Richardson extrapolation to obtain a sequence of solutions which converge quadratically. Note that no extra computations are performed except to take the linear combination of two previous results.

	Euler $\rightarrow N(\Delta t)$		$2N(\Delta t/2) - N(\Delta t)$	
Δt	Y^n	Rate	Y^n	Rate
1/10	$1.953\ 10^{-3}$			
1/20	$6.342\ 10^{-3}$	0.692	$1.073\ 10^{-2}$	
1/40	$9.580\ 10^{-3}$	0.873	$1.282\ 10^{-2}$	2.09
1/80	$1.145\ 10^{-2}$	0.942	$1.332\ 10^{-2}$	2.05
1/160	$1.244\ 10^{-2}$	0.973	$1.343\ 10^{-2}$	2.03
1/320	$1.295\ 10^{-2}$	0.986	$1.346\ 10^{-2}$	2.01

Note that to get the rates in the last column more accuracy in the solution had to be used than the recorded values in the table.

This procedure of taking linear combinations is equivalent to polynomial interpolation where we interpolate the points $\big(h_i, N(h_i)\big)$, $i = 0, 1, \ldots, s$ and then evaluate the interpolation polynomial at $h = 0$. For example, suppose we have a first order approximation as was the case for the forward difference approximation and use h_0 and $h_1 = h_0/2$. Then the linear polynomial which interpolates $\big(h_0, N(h_0)\big)$ and $\big(h_1, N(h_1)\big)$ is

$$N(h_0)\frac{h - h_1}{h_0 - h_1} + N(h_1)\frac{h - h_0}{h_1 - h_0}$$

Setting $h_1 = h_0/2$ and evaluating the polynomial at $h = 0$, i.e., extrapolating to zero, we get $-N(h_0) + 2N(h_0/2)$ which is exactly the approximation we derived in (3.32) by taking the correct linear combination of (3.30) and (3.31).

3.3.2 Burlisch-Stoer method

The Burlisch-Stoer method is an extrapolation scheme which takes advantage of the error expansion of certain methods to produce very accurate results in an efficient manner. In order for the extrapolation method to work we must know that the error in approximating $w(x)$ is of the form

$$w(x) - N(h) = K_1 h + K_2 h^2 + K_3 h^3 + K_4 h^4 + \cdots . \tag{3.34}$$

However, if all the odd terms in this error expansion are known to be zero then this greatly enhances the benefits of repeated extrapolation. For example, suppose we have a second order approximation where all the odd powers of h are zero, i.e.,

$$w(x) - N(h) = K_1 h^2 + K_2 h^4 + \cdots + K_i h^{2i} + \mathcal{O}\big((h)^{2(i+1)}\big) . \tag{3.35}$$

Then, when we obtain a numerical approximation $N(h/2)$ the linear combination which eliminates the h^2 term is

$$[4w(x) - 4N(h/2)] - [w(x) - N(h)] = 3w(x) - [4N(h/2) - N(h)] = -K_2\frac{3h^4}{4} + \mathcal{O}(h^6)$$

so that the approximation $\frac{4}{3}N(h/2) - \frac{1}{3}N(h)$ is fourth order accurate after only one extrapolation step. Although extrapolation methods can be applied to all methods with an error expansion of the form (3.34), the most efficient methods, such as the Burlisch-Stoer method, use an underlying method which has an error expansion such as (3.35).

A common choice for the low order method is the two-step midpoint method

$$Y^{n+2} = Y^n + 2\Delta t f(t_{n+1}, Y^{n+1}) .$$

The error expansion for this method does not contain odd terms; see the exercises. The other modification that the Burlisch-Stoer method incorporates to improve the extrapolation process is to use rational interpolation rather than polynomial interpolation. A link to an implementation of this method can be found on its Wikipedia page.

3.4 Predictor-Corrector Methods

We have considered several implicit schemes for approximating the solution of an IVP. However, when we implement these schemes the solution of a nonlinear equation is necessary unless $f(t, y)$ is linear in y. This requires extra work and moreover, we know that methods such as the Newton-Raphson method for nonlinear equations are not guaranteed to converge globally. Additionally, we ultimately want to develop variable time step methods so we need methods which provide an easy way to estimate errors. For these reasons, we look at predictor-corrector schemes.

In predictor-corrector methods an implicit scheme is implemented explicitly because it is used to improve (or correct) the solution that is first obtained (or predicted) by an explicit scheme. The implicit scheme is implemented as an explicit scheme because instead of computing $f(t_{n+1}, Y^{n+1})$ we use the known predicted value at t_{n+1}. One can also take the approach of correcting more than once. You can view this approach as being similar to applying the Newton-Raphson method where we take the predictor step as the initial guess and each corrector is a Newton iteration; however, the predictor step gives a systematic approach to finding an initial guess.

We first consider the Euler-trapezoidal predictor-corrector pair where the explicit scheme is forward Euler and the implicit scheme is the trapezoidal method (3.11). Recall that the forward Euler scheme is first order and the trapezoidal is second order. Letting the result of the predicted solution at t_{n+1} be Y_p^{n+1}, we have the following predictor-corrector pair.

Euler-Trapezoidal Predictor-Corrector Method

$$\begin{aligned} Y_p^{n+1} &= Y^n + \Delta t f(t_n, Y^n) \\ Y^{n+1} &= Y^n + \tfrac{\Delta t}{2}\Big[f(t_{n+1}, Y_p^{n+1}) + f(t_n, Y^n)\Big] \end{aligned} \tag{3.36}$$

As can be seen from the description of the scheme, the implicit trapezoidal method is now implemented as an explicit method because we evaluate f at the known point (t_{n+1}, Y_p^{n+1}) instead of at the unknown point (t_{n+1}, Y^{n+1}). The method requires two function evaluations so the work is equivalent to a two-stage RK method. The scheme is often denoted by PECE because we first predict Y_p^{n+1}, then evaluate $f(t_{n+1}, Y_p^{n+1})$, then correct to get Y^{n+1} and finally evaluate $f(t_{n+1}, Y^{n+1})$ to get ready for the next step.

The predicted solution Y_p^{n+1} from the forward Euler method is first order but we add a correction to it using the trapezoidal method and improve the error. We can view the predictor-corrector pair as implementing the difference scheme

$$Y^{n+1} = Y^n + \frac{\Delta t}{2}\Big[f\big(t_{n+1}, Y^n + \Delta t f(t_n, Y^n)\big) + f(t_n, Y^n)\Big]$$

which uses an average of the slope at (t_n, Y^n) along with t_{n+1} and the Euler approximation there. To analytically demonstrate the accuracy of a predictor-corrector

method it is helpful to write the scheme in this manner. In the exercises you are asked to show that this predictor-corrector pair is second order. Example 3.12 demonstrates that numerically we get second order.

One might believe that if one correction step improves the accuracy, then two or more correction steps are better. This leads to methods which are commonly denoted as PE(CE)^r schemes where the last two steps in the correction process are repeated r times. Of course it is not known a priori how many correction steps should be done but since the predictor step provides a good starting guess, only a small number of corrections are typically required. The effectiveness of the correction step can be dynamically monitored to determine r. The next example applies the Euler-trapezoid rule to an IVP using more than one correction step.

Example 3.12. EULER-TRAPEZOIDAL PREDICTOR-CORRECTOR PAIR

In this example we first perform a step of the Euler-trapezoidal predictor-corrector pair by hand to demonstrate how it is implemented and then compare the numerical results when a different number of corrections are used. Specifically we consider the IVP

$$y'(t) = \frac{ty^2}{\sqrt{9-t^2}-2}, \quad 0 < t \leq 2 \qquad y(0) = 1$$

which has an exact solution $y(t) = 1/\sqrt{9-t^2}$ found by separating variables and integrating.

Using $Y^0 = 1$ and $\Delta t = 0.1$ we first predict the value at 0.1 using the forward Euler method with $f(t,y) = ty^2/(\sqrt{9-t^2}-2)$ to get

$$\mathrm{P}: \quad Y_p^1 = Y^0 + .1f(t_0, Y^0) = 1 + 0.1(0) = 1.0\,;$$

then evaluate the slope at this point

$$\mathrm{E}: \quad f(0.1, 1.0) = \frac{(.1)(1^2)}{(\sqrt{9-.1^2}-2)} = 0.03351867$$

and finally correct to obtain the approximation at $t_1 = 0.1$

$$\mathrm{C}: Y^1 = Y^0 + \frac{0.1}{2}\big[f(0.1,1) + f(0,1)\big] == 1.03351867$$

with

$$\mathrm{E}: f(0.1, 1.8944272) = 0.03356667\,.$$

To perform a second correction we have

$$\mathrm{C}: Y^1 = Y^0 + \frac{0.1}{2}\big[f(0.1,1.03351867) + f(0,1)\big] = 1.00167316$$

where

$$\mathrm{E}: f(.1, 1.) = 0.03463567\,.$$

The results for the approximate solutions at $t = 2$ are given in the table below using decreasing values of Δt; the corresponding results from just using the forward Euler method

are also given. As can be seen from the table, the predictor-corrector pair is second order. Note that it requires one additional function evaluation, $f(t_{n+1}, Y_i^P)$, than the Euler method. The Midpoint rule requires the same number of function evaluations and has the same accuracy as this predictor-corrector pair. However, the predictor-corrector pair provides an easy way to estimate the error at each step.

	PECE		PE(CE)^2	
Δt	Error	rate	Error	rate
$1/10$	$2.62432\ 10^{-2}$		$3.93083\ 10^{-2}$	
$1/20$	$7.66663\ 10^{-2}$	1.75	$1.16639\ 10^{-2}$	1.75
$1/40$	$3.18110\ 10^{-3}$	1.87	$3.18110\ 10^{-3}$	1.87
$1/80$	$8.31517\ 10^{-4}$	1.94	$8.31517\ 10^{-4}$	1.94
$1/160$	$2.12653\ 10^{-4}$	1.97	$2.12653\ 10^{-4}$	1.97

In the previous example we saw that the predictor was first order, the corrector second order and the overall method was second order. It can be proved that *if the corrector is $\mathcal{O}(\Delta t^n)$ and the predictor is at least $\mathcal{O}(\Delta t^{n-1})$ then the overall method is $\mathcal{O}(\Delta t^n)$*. Consequently the PC pairs should be chosen so that the corrector is one degree higher accuracy than the predictor.

Higher order predictor-corrector pairs often consist of an explicit multistep method such as an Adams-Bashforth method and a corresponding implicit Adams-Moulton multistep method. The pair should be chosen so that the only additional function evaluation in the corrector equation is at the predicted point. To achieve this one often chooses the predictor and corrector to have the same accuracy. For example, one such pair is an explicit third order Adams-Bashforth predictor coupled with an implicit third order Adams-Moulton. Notice that the corrector only requires one additional function evaluation at (t_{n+1}, Y_p^{n+1}).

Third order Adams-Moulton predictor-corrector pair

$$\begin{aligned} Y_p^{n+1} &= Y^n + \tfrac{\Delta t}{12}\left[23f(t_n, Y^n) - 16f(t_{n-1}, Y^{n-1}) + 5f(t_{i-2}, Y^{n-2})\right] \\ Y^{n+1} &= Y^n + \tfrac{\Delta t}{12}\left[5f(t_{n+1}, Y_p^{n+1}) + 8f(t_n, Y^n) - f(t_{n-1}, Y^{n-1})\right] \end{aligned} \tag{3.37}$$

Example 3.13. THIRD ORDER ADAMS-MOULTON PREDICTOR CORRECTOR PAIR

In the table below we compare the errors and rates of convergence for the PC pair (3.37) and the third order Adams-Bashforth method defined in (3.27). Note that both numerical rates are approaching three but the error in the PC pair is almost an order of magnitude smaller at a fixed Δt.

	Predictor only		PC pair	
Δt	Error	rate	Error	rate
$1/10$	$2.0100\ 10^{-2}$		$1.5300\ 10^{-3}$	
$1/20$	$3.6475\ 10^{-3}$	2.47	$3.3482\ 10^{-4}$	2.19
$1/40$	$5.4518\ 10^{-4}$	2.74	$5.5105\ 10^{-5}$	2.60
$1/80$	$7.4570\ 10^{-5}$	2.87	$7.9035\ 10^{-6}$	2.80
$1/160$	$9.7513\ 10^{-6}$	2.93	$1.0583\ 10^{-6}$	2.90

3.5 Comparison of single-step and multistep methods

We have seen that single-step schemes are methods which essentially have no "memory". That is, once $y(t_n)$ is obtained they perform approximations to $y(t)$ in the interval $(t_n, t_{n+1}]$ as a means to approximate $y(t_{n+1})$; these approximations are discarded once $y(t_{n+1})$ is computed. On the other hand, multistep methods "remember" the previously calculated solutions and slopes because they combine information that was previously calculated at points such as $t_n, t_{n-1}, t_{n-2} \ldots$ to approximate the solution at t_{n+1}.

There are advantages and disadvantages to both single step and multistep methods. Because multistep methods use previously calculated information, we must store these values; this is not an issue when we are solving a single IVP but if we have a system then our solution and the slope are vectors and so this requires more storage. However multistep methods have the advantage that $f(t, y)$ has already been evaluated at prior points so this information can be stored and no new function evaluations are needed for explicit multistep methods. Consequently multistep methods require fewer function evaluations per step than single-step methods and should be used where it is costly to evaluate $f(t, y)$.

If we look at methods such as the Adams-Bashforth schemes given in (3.27) then we realize another shortcoming of multistep methods. Initially we set $Y_0 = y(t_0)$ and then use this to start a single-step method. However, if we are using a two-step method we need both Y_0 and $Yone$ to implement the scheme. How can we get an approximation to $y(t_1)$? The obvious approach is to use a single step method. So if we use m previous values (including t_n) then we must take $m - 1$ steps of a single-step method to start the simulations; it is $m - 1$ steps because we have the value Y_0. Of course care must be taken in the choice of which single step method to use and this was discussed in Example **??**. We saw that if our multistep method is $\mathcal{O}(\Delta t^r)$ then we should choose a single step method of the same accuracy or one power less; a scheme which converges at a rate of $\mathcal{O}(\Delta t^{r-2})$ or less would contaminate the accuracy of the method.

In the next chapter we investigate variable time step and variable order methods. It is typically easier to do this with single-step rather than multistep methods. However, *multivariable methods* have been formulated which are equivalent to multistep methods on paper but are implemented in a different way which allows easier use of a variable time step.

Older texts often recommend multistep methods for problems that require high accuracy and whose slope is expensive to evaluate and Runge-Kutta methods for the rest of the problems. However, with the advent of faster computers and more efficient algorithms, the advantage of one method over the other is less apparent. It is worthwhile to understand and implement both single-step and multistep methods.

EXERCISES

1. Each of the following Runge-Kutta schemes is written in the Butcher tableau format. Identify each scheme as explicit or implicit and then write the scheme as

$$Y^{n+1} = Y^n + \sum_{i=1}^{s} b_i f(t_n + c_i, Y^n + k_i)$$

where the appropriate values are substituted for b_i, c_i, and k_i.

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

b.
$$\begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

2. Modify the derivation of the explicit second order Taylor series method in § 3.1.1 to derive an implicit second order Taylor series method.

3. Use a Taylor series to derive a third order accurate explicit difference equation for the IVP (2.2).

4. Gauss quadrature rules are popular for numerical integration because one gets the highest accuracy possible for a fixed number of quadrature points; however one gives up the "niceness" of the quadrature points. In addition, these rules are defined over the interval $[-1, 1]$. For example, the one-point Gauss quadrature rule is

$$\int_{-1}^{1} g(x)\, dx = \frac{1}{2} g(0)$$

and the two-point Gauss quadrature rule is

$$\int_{-1}^{1} g(x)\, dx = \frac{1}{2}\left[g(\frac{-1}{\sqrt{3}}) + g(\frac{1}{\sqrt{3}})\right.$$

Use the one-point Gauss rule to derive a Gauss-Runge-Kutta method. Is the method explicit or implicit? Does it coincide with any method we have derived?

5. Simpson's numerical integration rule is given by

$$\int_a^b g(x)\, dx = \frac{b-a}{6}\left[g(a) + 4g\left(\frac{a+b}{2}\right) + g(b)\right]$$

If $g(x) \geq 0$ on $[a, b]$ then it approximates the area under the curve $g(x)$ by the area under a parabola passing through the points $(a, g(a))$, $(b, g(b))$ and $((a+b)/2, g((a+b)/2))$. Use this quadrature rule to approximate $\int_{t_n}^{t_{n+1}} f(t, y)\, dt$ to obtain an explicit 3-stage RK method. When you need to evaluate terms such as f at $t_n + \Delta t/2$ use an appropriate Euler step to obtain an approximation to the corresponding y value as we did in the Midpoint method. Write your method in the format of (3.15) and in a Butcher tableau.

6. In § **??** we derived a second order BDF formula for uniform grids. In an analogous manner, derive the corresponding method for a nonuniform grid.

7. Use an appropriate interpolating polynomial to derive the multistep method

$$Y^{n+1} = Y^{n-1} + 2\Delta t f(t_n, Y^n)\,.$$

   Determine the accuracy of this method.

8. Determine the local truncation error for the 2-step Adams-Bashforth method (3.26).

APPENDIX

.1 Norms

If we have an approximate solution at a given point and we want to calculate the absolute error, then we simply take the magnitude of the difference between the exact solution and the approximation. However, many times the approximate solution is represented by a vector. This occurs if we have a boundary value problem, an initial boundary value problem or a system of initial value problems. If we want to assign a number to the error then we have to use the concept of a norm. The Euclidean length of a vector which you learned in algebra is actually a norm.

We want to generalize this concept to include other measures of a norm. We can view the Euclidean length as a map (or function) whose domain is $\mathbb{R}^n$ and whose range is all scalars i.e., $f : \mathbb{R}^n \to \mathbb{R}^1$. What properties does this Euclidean length have? We know that the length is always ≥ 0 and only equal to zero if the vector is identically zero. We know what multiplication of a vector by a scalar k does to the length; i.e., it changes the length by $|k|$. Also, from the triangle inequality we know that the length of the sum of two vectors is always $\leq$ the sum of the two lengths. We combine these properties into a formal definition of a vector norm.

A **vector norm**, denoted $\|\mathbf{x}\|$, is a map from $\mathbb{R}^n$ to $\mathbb{R}^1$ which has the properties

1. $\|\mathbf{x}\| \geq 0$ and $= 0$ only if $\mathbf{x} = \mathbf{0}$
2. $\|k\mathbf{x}\| = |k|\|\mathbf{x}\|$
3. $\|\mathbf{x}\| + \|\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangle inequality)

There are other ways to measure the length of vectors. All we have to do is find a map which satisfies the above three conditions; however, practically it should be useful. Three of the most useful vector norms are defined below.

1. **Euclidean norm**, denoted $\|\mathbf{x}\|_2$ and defined by $\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \left(\vec{x}^T\vec{x}\right)^{1/2}$
2. **maximum or infinity norm**, denoted $\|\mathbf{x}\|_\infty$ and defined by $\|\mathbf{x}\|_\infty = \max_{1\leq i\leq n} |x_i|$
3. **one-norm**, denoted $\|\mathbf{x}\|_1$ and defined by $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$

Although these are the standard definitions for the vector norms, when we output the error norm we need to normalize it. For example, suppose you compute an error vector, all of whose components are 0.1. Clearly we expect the Euclidean norm to

be 0.1 for a vector of any length but if we compute the norm using the definition above for a vector of length 10 then the result is 0.316 and for a vector of length 100 it is 1. So what we need to do is either normalize by the vector of the exact solution evaluated at the same grid points to give a relative error or use an alternate definition; for example for the Euclidean norm we use

$$\|\mathbf{x}\|_2 = \Big[\frac{1}{n}\sum_{i=1}^{n} x_i^2\Big]^{1/2}$$

which gives the answer of 0.1 for a vector all of whose components are 0.1 no matter what its length.

Example .14. CALCULATING NORMS OF A GIVEN VECTOR
Let $\mathbf{x} = (-3, 2, 4, -7)^T$. Determine $\|\mathbf{x}\|_1$, $\|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_\infty$.

$$\|\mathbf{x}\|_1 = |-3| + 2 + 4 + |-7| = 16$$
$$\|\mathbf{x}\|_2 = \sqrt{9 + 4 + 16 + 49} = \sqrt{78}$$
$$\|\mathbf{x}\|_\infty = \max\{|-3|, 2, 4, |-7|\} = 7$$

Many times we use a norm to measure the length of an error vector. In the previous example we saw that different norms give us different numbers for the same vector. Each norm actually measures a different attribute of a norm. We might ask ourselves how different these norms can be for the same vector. This is important to us because if we can show a particular norm of the error goes to zero, then we would like to know that the error measured in another norm goes to zero too. For vector norms (and all norms defined on finite dimensional spaces) one can demonstrate that all norms are **norm-equivalent**. This means that if we have two vector norms $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ then there exists constants C_1, C_2 greater than zero such that

$$C_1\|\mathbf{x}\|_\beta \le \|\mathbf{x}\|_\alpha \le C_2\|\mathbf{x}\|_\beta \quad \text{for all } \vec{x}\,. \tag{38}$$

Note that if this inequality holds, we also have the equivalent statement

$$\frac{1}{C_2}\|\mathbf{x}\|_\alpha \le \|\mathbf{x}\|_\beta \le \frac{1}{C_1}\|\mathbf{x}\|_\alpha \quad \text{for all } \vec{x}$$

If two norms are norm-equivalent and we have that $\|\vec{x}\|_\beta \to 0$ then clearly $\|\vec{x}\|_\alpha \to 0$. In the following example we determine the particular constants to demonstrate that $\|\mathbf{x}\|_\infty$ and $\|\mathbf{x}\|_2$ are norm-equivalent.

Example .15. NORM EQUIVALENCE

The norms $\|\mathbf{x}\|_\infty$ and $\|\mathbf{x}\|_2$ are equivalent.

We have

$$\|\mathbf{x}\|_2^2 = \sum_{i=1}^{n} x_i^2 \geq \max |x_i|^2 = \|\mathbf{x}\|_\infty^2$$

so $C_1 = 1$.

Also

$$\|\mathbf{x}\|_2^2 = \sum_{i=1}^{n} x_i^2 \leq n \max |x_i|^2 = n\|\mathbf{x}\|_\infty^2$$

so $C_2 = \sqrt{n}$. Summarizing

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{x}\|_\infty \quad \text{for all } \mathbf{x}$$

.2 Polynomial interpolation

Polynomial interpolation can be used to approximate a discrete set of data points or a function. In polynomial interpolation we find a polynomial which passes through all data points; this is in contrast to an approximation such as linear least squares which minimizes the ℓ_2-norm of the error. If we only interpolate function values then the interpolation is called **Lagrange interpolation** and if we interpolate derivative values then it is called **Hermite interpolation**. We will describe Lagrange interpolation here and we will assume we are given data points which could be from experimental data or from evaluating a function at a point. Specifically, if we are given $n+1$ distinct points

$$(x_1, y_1), (x_2, y_2), \cdots (x_{n+1}, y_{n+1})$$

we seek a polynomial of degree less than or equal to n, say p_n, such that

$$p_n(x_i) = y_i, \quad \text{for } i = 1, 2, \ldots, n+1$$

The reason the polynomial can be of degree less than n is, for example, if all the points happen to lie on a line then a first degree polynomial works even if we have more than two points. Interpolation means that the graph of the polynomial passes through our $n+1$ distinct data points. The general polynomial of degree n looks like

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

It can be proved that there is a unique polynomial which satisfies this interpolation problem.

An obvious way to find p_n is to solve $n+1$ equations. For example, if we want the quadratic that passes through the 3 points $(1, 10)$, (2, 19), (-1,-14) then clearly $p_2 = a_0 + a_1 x + a_2 x^2$ must satisfy

$$p_2(1) = 10 \qquad p_2(2) = 19 \qquad p_2(-1) = -14$$

or equivalently

$$a_0+a_1(1)+a_2(1)^2 = 10 \quad a_0+2\cdot a_1+(2)^2a_2 = 19 \quad a_0+a_1(-1)+a_2(-1)^2 = -14\,.$$

Solving these three equations we get the polynomial $p_2 = -x^2 + 12x - 1$. Although this approach is straightforward, there are more efficient ways to obtain an interpolating polynomial.

There are basically two main efficient approaches for determining the interpolating polynomial. Each has advantages in certain circumstances. Since we are guaranteed that we will get the same polynomial using different approaches (since the polynomial is unique) it is just a question of which is a more efficient implementation.

The Lagrange form of the interpolating polynomial is used in most situations. However, its drawback is that if you have $n+1$ points and decide to add a point, then the computations must all be redone. The Newton form of the interpolating polynomial has the advantage that adding a point reuses previous computations so it is much more efficient. For our needs, the Lagrange form is adequate.

To motivate this form of the interpolating polynomial, let's revisit our example where we found that the quadratic interpolating polynomial which interpolates the points $(1, 10)$, $(2, 19)$, $(-1, -14)$ is $p_2 = -x^2 + 12x - 1$. Instead of writing this polynomial in terms of the monomials $1, x, x^2$ lets rewrite it as

$$p_2 = 10 * L_1(x) + 19L_2(x) - 14L_3(x)$$

where $L_i(x)$, $i = 1, 3$ are quadratic polynomials which have the properties

$$L_1(2) = L_1(-1) = 0, \qquad L_1(1) = 1$$

$$L_2(1) = L_2(-1) = 0, \qquad L_2(2) = 1$$

$$L_3(1) = L_3(2) = 0, \qquad L_3(-1) = 1\,.$$

If we can do this, then clearly $p_2(1) = 10$, $p_2(2) = 19$ and $p_2(-1) = -14$. Once we have the $L_i(x)$ we have found our interpolating polynomial, just not reduced to its simplest form. At first it may seem that we have traded finding one quadratic polynomial for finding three others. However, let's look at what $L_i(x)$, $i = 1, 3$ are in our example.

$$L_1(x) = \frac{(x-2)(x+1)}{(1-2)(1+1)} = \frac{(x-2)(x+1)}{-2}$$

Clearly because we have the factor $(x-2)$ in the numerator, $L_1(2) = 0$. Similarly the factor $(x+1)$ in the numerator makes $L_1(-1) = 0$. When we evaluate the numerator at $x = 1$ we get the denominator which is just -2 so it satisfies $L_1(1) = 1$. Similarly

$$L_2(x) = \frac{(x-1)(x+1)}{(2-1)(2+1)} = \frac{(x-1)(x+1)}{3}$$

and

$$L_3(x) = \frac{(x-1)(x-2)}{(-1-1)(-1-2)} = \frac{(x-1)(x-2)}{6}$$

The numerator is easily determined because for L_i we simply use factors of $(x-x_j)$ for $j \neq i$. How do we get the denominator? Because we want $L_i(x_i) = 1$, we simply choose the denominator to equal the numerator evaluated at x_i.

Given $n+1$ distinct points

$$(x_1, y_1), (x_2, y_2), \cdots (x_{n+1}, y_{n+1})$$

The Lagrange form of the interpolating polynomial is

$$p_n = \sum_{i=1}^{n+1} y_i L_i(x)\,, \tag{39}$$

where

$$L_i(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_{n+1})}{(x_i-x_1)(x_i-x_2)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_{n+1})} \tag{40}$$

The important properties of $L_i(x)$ are

1. $L_i(x_j) = 0$ for $j \neq i$
2. $L_i(x_i) = 1$

.3 Partial derivatives

When a function depends on more than one variable, we express its rate of change with respect to a particular independent variable by using **partial derivatives**. Let $g = g(x_1, x_2, \ldots, x_n)$ then the definition and standard notation for the first partial of g with respect to x_i is

$$\frac{\partial g}{\partial x_i} = g_{x_i} = \lim_{h\to 0} \frac{g(x_1, x_2, \ldots, x_i + h, x_{i+1}, \ldots, x_n) - g(x_1, x_2, \ldots, x_i, \ldots, x_n)}{h}\,.$$

This gives the change in g with respect to x_i where all other independent variables are held fixed. For example, if $g = g(x_1, x_2)$ then g_{x_1} gives the change in g along lines parallel to the x_1-axis since x_2 is held fixed.

To calculate higher derivatives we simply apply this formula repeatedly. As long as the function is continuously differentiable then the order of differentiation does not matter. The notation used is, for example if $g = g(x_1, x_2)$,

$$g_{x_1x_1} = \frac{\partial^2 g}{\partial x_1^2} \qquad g_{x_1x_2} = \frac{\partial^2 g}{\partial x_1 \partial x_2} = \frac{\partial^2 g}{\partial x_2 \partial x_1}$$

Calculating partial derivatives is straightforward because we simply hold all other variables fixed; hence we can use the rules learned for differentiating functions of a single variable. The following example illustrates how to compute partial derivatives.

Example .16. CALCULATING PARTIAL DERIVATIVES

Let $g(x,y,z) = e^{2x}y^4\sin(3z)$. Compute g_x, g_y, g_z. Then compute g_{xz} and g_{zx} and show that they are the same.

To calculate g_x we treat y^4 and $\sin(3z)$ as if they were constants because they are held fixed. We have $g_x = 2e^{2x}y^4\sin(3z)$. To calculate g_y we hold the terms involving x and z fixed to get $g_y = 4y^3e^{2x}\sin(3z)$. Finally, to get g_z we hold the terms involving x and y fixed to get $g_z = 3e^{2x}y^4\cos(3z)$.

To calculate g_{xz} we take the partial of g_x with respect to z. Since $g_x = 2e^{2x}y^4\sin(3z)$ we have $g_{xz} = 6e^{2x}y^4\cos(3z)$. To calculate g_{zx} we take the partial of $g_z = 3e^{2x}y^4\cos(3z)$ with respect to x to get $g_{zx} = 6e^{2x}y^4\cos(3z)$. Because the given g is continuously differentiable the order of differentiation does not matter so that $g_{xz} = g_{zx}$.

There are three partial differential operators that we will use extensively. Recall that in calculus we learned that we take the gradient of a scalar and get a vector field. So the magnitude of the gradient of u is the magnitude of the change in u, analogous to the magnitude of the slope in one dimension. The standard notation used is the Greek symbol nabla, ∇ or simply "grad". Remember that it is an operator and so just writing ∇ does not make sense but rather we must write, e.g., ∇u. The ∇ operator is the vector of partial derivatives so in three dimensions it is $(\partial/\partial x, \partial/\partial y, \partial/\partial z)^T$.

One use we will have for the gradient is when we want to impose a flux boundary condition. Clearly there are times when we want to know the rate of change in $g(x,y)$ in a direction other than parallel to the coordinate axis; remember that the standard partial derivative gives the change in the coordinate axis. When this is the case we define a unit vector in the direction of the desired change and we use the gradient of g. If $\hat{\mathbf{n}}$ is the unit vector giving the direction then the derivative and the notation we use is

$$\frac{\partial g}{\partial \hat{\mathbf{n}}} \equiv \nabla g \cdot \hat{\mathbf{n}}\,. \tag{41}$$

Note that if $\hat{\mathbf{n}} = (1,0)^T$, i.e., in the direction of the x-axis, then we just get g_x which is the standard partial derivative in the direction of the x-axis; similarly if $\hat{\mathbf{n}} = (0,1)^T$ then we get g_y. We will have a particular use for this notation when we specify a boundary condition such as the flux on the boundary. In one dimension the flux is just $g'(x)$ but in higher dimensions it is the change in g along the normal to the boundary. So in higher dimensions we will specify $\partial g/\partial\hat{\mathbf{n}}$ as a Neumann boundary condition.

The next differential operator that we need is the divergence. Recall that the divergence is a *vector* operator. It is also represented by ∇ or simply "div" but typically we use a dot after it to indicate that it operates on a vector; other sources will use a bold face ∇. So if $\mathbf{w} = (w_1, w_2, w_3)$ then the divergence of $\mathbf{w}$, denoted $\nabla\cdot\mathbf{w}$, is the scalar $\partial w_1/\partial x + \partial w_2/\partial y + \partial w_3/\partial z$.

The last differential operator that we need is called the Laplacian.[4] It combines the gradient and the divergence to get a second order operator but of course the order is critical. If $u(x,y,z)$ is a scalar function then we can take its gradient to get

[4]Named after the French mathematician Pierre-Simon de Laplace (1749-1827).

a vector function, then the divergence may be applied to this vector function to get a scalar function. Because this operator is used so extensively in PDEs it is given a special notation, Δ which is the Greek symbol for capital delta. In particular we have $\Delta = \nabla \cdot \nabla$ so if $g = g(x, y, z)$ then

$$\Delta g \equiv \nabla \cdot \nabla g = g_{xx} + g_{yy} + g_{zz} \tag{42}$$

because

$$\nabla \cdot \nabla g = \nabla \cdot \left[(g_x, g_y, g_z)^T\right] = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T \cdot (g_x, g_y, g_z)^T = g_{xx} + g_{yy} + g_{zz}\,.$$

In the sequel we will typically use the notation Δg instead of $\nabla \cdot \nabla g$. In some texts $\nabla^2 g$ is used for Δg but we will not use this notation.

.4 Taylor series expansions

Expanding a function in a Taylor series is a useful tool in numerical analysis. We will use Taylor series to derive methods and to obtain estimates for the error in approximating a solution to a differential equation. Expanding functions of either one or two variables should be practiced until it becomes second nature.

A Taylor series expansion of a function is an infinite power series although there is a useful result which combines terms after the nth term into a remainder term. The function being expanded must be infinitely differentiable and the expansion is valid in a neighborhood of a point. The Taylor series expansion of a differentiable function $g(\tilde{x})$ about $\tilde{x} = a$ is

$$\begin{aligned} g(\tilde{x}) &= g(a) + g'(a)(\tilde{x} - a) + \frac{g''(a)}{2!}(\tilde{x} - a)^2 + \frac{g'''(a)}{3!}(\tilde{x} - a)^3 \\ &\quad + \cdots + \frac{g^{[n]}(a)}{n!}(\tilde{x} - a)^n + \cdots . \end{aligned} \tag{43}$$

Note that we are expanding $g(x)$ in powers of $(x - a)$. Although this is the form of the Taylor series that is usually given in calculus texts, we typically use this expansion in the slightly different form

$$\begin{aligned} g(x + \Delta x) &= g(x) + \Delta x g'(x) + \frac{(\Delta x)^2}{2!} g''(x) + \frac{(\Delta x)^3}{3!} g'''(x) \\ &\quad + \frac{(\Delta x)^n}{n!} g^{[n]}(x) + \cdots . \end{aligned} \tag{44}$$

This is easily obtained from (15) by setting $x + \Delta x = \tilde{x}$ and $x = a$ so that $\tilde{x} - a = \Delta x$. Note that this expansions tells us how g behaves in the neighborhood of a point x.

Even though a Taylor series is an infinite series, there is a result called *Taylor series with remainder* which collects all terms after the nth term in a single term

called the remainder. We have

$$\begin{aligned} g(x+\Delta x) &= g(x)+\Delta x g'(x)+\frac{(\Delta x)^2}{2!}g''(x)+\frac{(\Delta t)^3}{3!}g'''(x)+\cdots \\ &\quad +\frac{(\Delta x)^n}{n!}g^{[n]}(x)+\frac{(\Delta x)^{n+1}}{(n+1)!}g^{[n+1]}(\xi) \quad \xi\in(x,x+\Delta x)\,. \end{aligned} \tag{45}$$

Note that the result says that there *exists* a point $\xi \in (x, x+\Delta x)$ where the expansion holds but it does not give a means for determining the specific ξ. This makes sense because the usual Taylor series is an infinite series and if we could find the ξ that satisfies the remainder term then we would be converting an infinite series into a finite one. If one forgets and writes the remainder term as $g^{[n+1]}(x)(\Delta x)^{n+1}/(n+1)$ then this is incorrect because we have converted the infinite Taylor series into a finite series for all differentiable functions $g(x)$.

In many cases the function we want to expand is in terms of more than one independent variable. Of course this means that we must use partial derivatives in the expansion instead of ordinary derivatives. To see how we can use (16) to derive a Taylor series expansion in two independent variables consider the points $P=(x,y)$, $Q=(x+h,y)$ and $R=(x+h,y+k)$. Then we want an expansion which tells us how $g(x,y)$ varies at R given its value at P. From P to Q the independent variable y is held fixed so we can use (16) to write

$$\begin{aligned} g(Q) &= g(P)+\Delta x g_x(P)+\frac{(\Delta x)^2}{2!}g_{xx}(P)+\frac{(\Delta t)^3}{3!}g_{xxx}(P) \\ &\quad +\frac{(\Delta x)^n}{n!}\frac{\partial g^{[n]}}{\partial x^n}(P))+\cdots\,. \end{aligned}$$

Now from Q to R the independent variable x is held fixed so we have

$$\begin{aligned} g(R) &= g(Q)+\Delta y g_y(Q)+\frac{(\Delta y)^2}{2!}g_{yy}(Q)+\frac{(\Delta y)^3}{3!}g_{yyy}(Q) \\ &\quad +\frac{(\Delta y)^n}{n!}\frac{\partial g^{[n]}}{\partial y^n}(Q)+\cdots\,. \end{aligned}$$

Now we can substitute the expression for $g(Q)$ into this and all of those terms will be evaluated at the known point P; however, we still have terms evaluated at Q. What we do is expand each of these terms about $P=(x,y)$; for example, expanding $g_y(x+h,y)$ about x gives

$$\begin{aligned} g_y(Q) &= g_y(P)+\Delta x g_{yx}(P)+\frac{(\Delta x)^2}{2!}g_{yxx}(P)+\frac{(\Delta x)^3}{3!}g_{yxxx}(P)+\cdots \\ &\quad +\frac{(\Delta x)^n}{n!}\frac{\partial g^{[n+1]}}{\partial x^n\partial y}(P)+\cdots\,. \end{aligned}$$

so we get mixed derivative terms. After expanding all the terms about P and grouping like terms we get the following Taylor series expansion for a differentiable function $g(x,y)$ of two independent variables

$$
\begin{aligned}
g(x+h, y+k) = g(x,y) &+ hg_x(x,y) + kg_y(x,y) \\
&+ \frac{1}{2!}\left[h^2 g_{xx}(x,y) + k^2 g_{yy}(x,y) + 2khg_{xy}(x,y)\right] \\
&+ \frac{1}{3!}\left[h^3 g_{xxx}(x,y) + k^3 g_{yyy}(x,y) + 3k^2 h g_{xyy}(x,y) + 3h^2 k g_{xxy}(x,y)\right] \\
&+ \cdots .
\end{aligned}
\tag{46}
$$

Here, for brevity, we have taken the change in x as h and the change in y as k. Note that this expansion assumes that $g(x,y)$ is continuously differentiable in both variables so that not only can we differentiate it but the order of differentiation doesn't matter.