Part I

Initial Value Problems

Chapter 2

Introduction to Initial Value Problems

In calculus and physics we encounter initial value problems although this terminology may not be used. For example, in calculus a standard problem is to determine the amount of radioactive material remaining after a fixed time if the initial mass of the material is known along with the fraction of the material which will decay at any instant. In the typical radioactive decay model the rate of change of the mass (i.e., its first derivative with respect to time) of the radioactive material is assumed proportional to the amount of material present at that instant; the proportionality constant is the given decay rate which is negative since the mass is decreasing. Thus a first order differential equation for the mass is given along with the initial mass of the object. This model is similar to one to describe population growth except in this case the proportionality constant is positive.

Recall from physics that Newton's second law of motion states that the force applied to an object equals its mass times its acceleration. If we have a function which denotes the position of the object at any time, then its first derivative with respect to time is the velocity of the object and the second derivative is the acceleration. Consequently, Newton's second law is a second order ODE for the displacement and if we specify the initial location and velocity of the object we have a second order initial value problem.

To be more precise, an initial value problem (IVP) for an unknown function $y(t)$ consists of an ordinary differential equation (ODE) for $y(t)$ and one or more auxiliary conditions specified *at the same value of* t. Here the unknown is only a function of one independent variable (t) so differentiation of $y(t)$ involves standard derivatives, not partial derivatives. The ODE specifies how the unknown changes with respect to the independent variable which we refer to as time but it could also represent an x location, etc. Recall that the order of a differential equation refers to the highest derivative occurring in the equation; e.g., a second order ODE for $y(t)$ must include a $y''(t)$ term but may or may not include a $y'(t)$ term. The number of additional

conditions corresponds to the order of the differential equation; for example, for a first order ODE we specify the value of $y(t)$ at an initial time t_0 and for a second order ODE we specify the value of both $y(t)$ and $y'(t)$ at t_0. For obvious reasons, these extra conditions are called initial conditions. The goal is to determine the value of $y(t)$ for subsequent times, $t_0 < t \leq T$ where T denotes a final time.

We begin this chapter by providing a few applications where IVPs arise. These problems provide examples where an exact solution is known so they can be used to test our numerical schemes. Instead of analyzing each IVP, we write a generic first order IVP which serves as our prototype problem for describing various approaches for approximating a solution to the IVP. We provide conditions which guarantee that the IVP has a unique solution and that the solution varies a small amount when the initial conditions are perturbed by a small amount.

Once we have specified our prototype first order IVP we introduce the idea of approximating its solution using a difference equation. In general, we have to give up the notion of finding an analytic solution which gives an expression for the solution at any time and instead find a discrete solution which is an approximation to the exact solution at a set of finite times. The basic idea is that we discretize our domain, in this case a time interval, and then derive a difference equation which approximates the differential equation in some sense. The difference equation is in terms of a discrete function and only involves differences in the function values; that is, it does not contain any derivatives. Our hope is that as the difference equation is imposed at more and more points (which much be chosen in a uniform manner) then its solution approaches the exact solution to the IVP.

One might ask why we only consider a prototype equation for a first order IVP when many IVPs include higher order equations. At the end of this chapter we briefly show how a higher order IVP can be converted into a system of first order IVPs.

In Chapter 3 we derive the Euler methods which are the simplest numerical methods for approximating the solution to a first order initial value problem. Because the methods are simple, we can easily derive them plus give graphical interpretations to gain intuition about approximations. Once we analyze the errors made in replacing the continuous differential equation by a difference equation, we see that the methods only converge linearly which is quite slow. This is the motivation for looking at higher accurate methods in the following chapter. We look at several numerical examples and verify the linear convergence of the methods and we see that in certain situations one of the methods tends to oscillate and even "blow up" while the other always provides reliable results. This motivates us to study the numerical stability of methods.

In Chapter 4 we provide a survey of numerical schemes for solving our prototype IVP. In particular, we present two classes of methods. The first class of methods consist of single step methods which use the solution at the previous time along with approximations at some intermediate time points to approximate the solution at the next time level. The second class of methods is called multistep methods which use the calculated approximations at several previous times to approximate the solution at the next time level. In particular, we look at Runge-Kutta methods (single step) and multistep methods in detail. Other topics included in this chapter

are predictor-corrector methods and extrapolation methods.

For many real-world applications a mathematical model involves several unknown functions which are inter-related. Typically there is a differential equation for each unknown which involves some or all of the other unknowns. If each differential equation is first order and the initial value of each unknown is given, then we have a *system of first order IVPs*. Systems are discussed in Chapter 5. Also in this chapter we briefly consider adaptive time stepping methods. For simplicity, in all the previous work we assume that the approximations are generated at evenly spaced points in time. Of course, in practice this is very inefficient because there may be times when the solution changes rapidly so small time increments are needed and other instances when the solution varies slowly so that a larger time increment needs to be used.

2.1 Examples of IVPs

For the simplest type of IVP we have a first order ODE with one initial condition which is the value of the unknown at the initial time. An example of such an IVP is an exponential model for population growth/decay where we assume the rate of growth or decay of the population $p(t)$ is proportional to the amount present at any time t, i.e., $p'(t) = rp(t)$ for some constant r and we know the initial population, i.e., $p(0) = p_0$. Thus, the IVP for the exponential growth model with growth rate r is given by

$$\begin{cases} p'(t) &= rp(t) \quad t > 0 \\ p(0) &= p_0 \,. \end{cases} \tag{2.1}$$

If $r < 0$ then the differential equation models exponential decay as in the example of radioactive decay. If $r > 0$ then the differential equation models exponential growth as in the case of bacteria growth in a large petri dish. Why is this growth/decay called exponential? To answer this question we solve the differential equation analytically to get the population at any time t as $p(t) = p_0 e^{rt}$ which says the population behaves exponentially. This solution can be verified by demonstrating that it satisfies the differential equation $p'(t) = rp(t)$ and the initial condition; we have

$$p(t) = p_0 e^{rt} \Rightarrow p'(t) = rp_0 e^{rt} = rp(t)$$

and $p(0) = p_0$. In § 2.2 we see how this analytic solution is obtained.

This exponential model makes sense for applications such as bacteria growth in an area not confined by space because the model assumes there is an endless supply of resources and no predators. A more realistic population model is called logistic growth where a condition is imposed which gives a carrying capacity of the system; i.e., the population is not allowed to grow larger than some prescribed value. When the population is considerably below this threshold the two models produce similar results but for larger values of time logistic growth does not allow unbounded growth as the exponential model does. The logistic model we consider restricts the growth rate in the following way

$$r = r_0 \left(1 - \frac{p}{K}\right) , \tag{2.2}$$

where K is the maximum allowable population and r_0 is a given growth rate for small values of the population. As the population p increases to near the threshold value K then p/K becomes close to one (but less than one) and so the term $(1 - p/K)$ is positive but approaches zero as p approaches K. Thus the growth rate decreases because of fewer resources; the limiting value is when $p = K$ and the growth rate is zero. However when p is small compared with K, the term $(1-p/K)$ is near one and the model behaves like exponential growth with a rate of $r \approx r_0$. Assuming the population at any time is proportional to the current population using the proportionality constant (2.2), the differential equation becomes

$$p'(t) = r_0 \left(1 - \frac{p(t)}{K}\right) p(t) = r_0 p(t) - \frac{r_0}{K} p^2(t) \tag{2.3}$$

along with $p(0) = p_0$. This equation is *nonlinear* in the unknown $p(t)$ due to the $p^2(t)$ term and is more difficult to solve than the exponential growth equation. However, it can be shown that the solution is

$$p(t) = \frac{K p_0}{(K - p_0)e^{-r_0 t} + p_0}\,. \tag{2.4}$$

This can be verified by substitution into the differential equation and verification of the initial condition $p(0) = p_0$. We expect that as we take the $\lim_{t\to\infty} p(t)$ we should get the threshold value K. Clearly this is true because

$$\lim_{t\to\infty} p(t) = K p_0 \lim_{t\to\infty} \frac{1}{(K - p_0)e^{-r_0 t} + p_0} = K p_0 \frac{1}{p_0} = K\,,$$

where we have used the fact that $\lim_{t\to\infty} e^{-r_0 t} = 0$ for $r_0 > 0$.

Example 2.1. EXPONENTIAL AND LOGISTIC GROWTH

Suppose we have bacteria growing in a petri dish with an initial count of 1000 bacteria. The number of bacteria after an hour is counted and an estimate for the initial hourly growth rate is determined to be 0.294; thus for the exponential model $r = 0.294$ and for the logistic model we set r_0 to this value. We assume that the carrying capacity, i.e., the maximum number of bacteria the petri dish can sustain, is 10,000. Write the IVPs for each model, give their analytic solution and compare the results graphically from the two models.

Let $p_e(t)$ and $p_\ell(t)$ represent the solution at any time t to the exponential growth model and the logistic growth model, respectively. Using (2.1) and (2.3), the IVPs for each model are given by

$$\text{EXPONENTIAL GROWTH MODEL:} \quad \begin{cases} p_e'(t) &= 0.294 p_e(t) \\ p_e(0) &= 10^3 \end{cases}$$

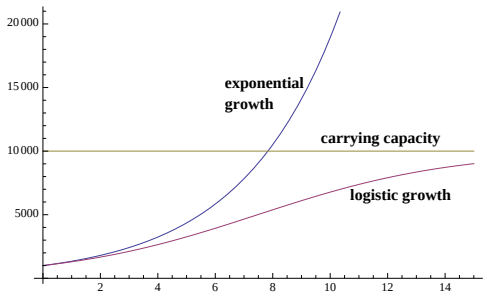
and

$$\text{LOGISTIC GROWTH MODEL:} \quad \begin{cases} p_\ell'(t) &= 0.294 p_\ell(t) - \dfrac{0.294}{10^4} p_\ell^2(t) \\ p_\ell(0) &= 10^3\,. \end{cases}$$

The analytic solutions are

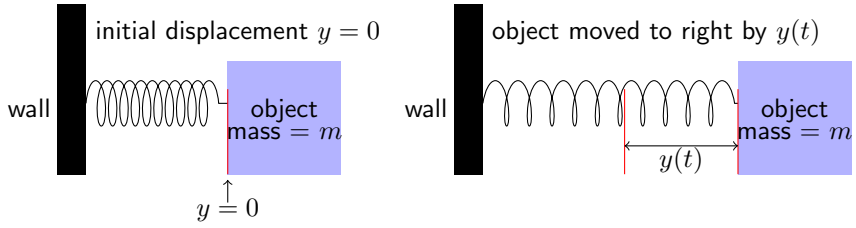
$$p_e(t) = 10^3 e^{.294t} \qquad p_\ell(t) = \frac{10^7}{(9000e^{-.294t} + 1000)},$$

where we have used (2.4) to get $p_\ell(t)$. To compare the results, we plot the exact solutions to the IVPs as well as giving a table of the bacteria population (truncated to the nearest whole number) from each model for a range of hours. As we see from the plots and the table, the predicted size of the bacteria colony is close for small values of t but as the time increases the exponential model predicts boundless growth whereas the logistic model predicts a population which never exceeds the carrying capacity of 10,000.



t	0	1	2	4	6	8	10	15	20
$p_e(t)$	1000	1341	1800	3241	5835	10506	18915	82269	357809
$p_\ell(t)$	1000	1297	2116	2647	3933	5386	6776	9013	9754

Instead of a first order ODE in the IVP we might have a higher order equation such as the *harmonic oscillator* equation. This models an object attached to a wall with a spring and the unknown function is the displacement of the object from the wall at any time. At the initial time $t = 0$ the system is in equilibrium so the displacement $y(t)$ is zero. This is illustrated pictorially in the figure on the left below. The figure on the right illustrates the situation where at a later time t the object has moved to the right an amount $y(t)$ so the spring is stretched.



Using basic laws of physics the second order ODE which models this spring-mass system is

$$my''(t) = -ky(t) - cy'(t),$$

where m is the mass of the object, k is the spring constant in Hooke's law (force = -spring constant times the displacement), and c is the coefficient of friction. Because the differential equation is second order, we must specify two initial conditions at the same instance of time; here we specify the initial displacement $y(0) = 0$ and the initial velocity $y'(t) = \nu$. If there is no friction then the differential equation models a *simple harmonic oscillator*. In this case the IVP becomes

$$\begin{array}{rcl} y''(t) & = & -\omega^2 y(t) \\ y(0) & = & 0 \\ y'(0) & = & \nu\,, \end{array} \tag{2.5}$$

where $\omega = \sqrt{k/m}$. Clearly $\sin\omega t$ and $\cos\omega t$ satisfy this differential equation so the general solution is $y(t) = C_1 \sin\omega t + C_2 \cos\omega t$ for constants C_1, C_2. Satisfying the initial condition $y(0) = 0$ implies $C_2 = 0$ and C_1 is determined by setting $y'(t) = \omega C_1 \cos\omega t$ equal to ν at $t = 0$; i.e., $C_1 = \nu/\omega$. Since the solution to the simple harmonic oscillator equation is $y(t) = (\nu/\omega)\sin\omega t$, the solution should be periodic as indicated graphically in the next example.

If the coefficient of friction is nonzero, then the equation is more difficult to solve analytically because we have to consider three cases depending on the sign of the term $c^2 - 4\omega^2$. However, the inclusion of the friction term will not cause problems when we discretize.

Example 2.2. SIMPLE HARMONIC OSCILLATOR

Assume we have an object of mass $m = 5$ attached to a wall with a spring whose Hooke's constant is $k = 0.45$. Assume that $y(t)$ denotes the displacement of the object at any time t, that the initial displacement is zero and the initial velocity is 4. Write the IVP for this problem assuming first that the coefficient of friction is zero and give the exact solution. Verify that the solution satisfies the DE and initial conditions. Plot the solution for $0 \le t \le 5$.

Note that $\omega = \sqrt{k/m} = \sqrt{0.45/5} = \sqrt{0.09} = 0.3$. The IVP is given by

$$\text{NO FRICTION:} \quad \left\{ \begin{array}{rcl} y''(t) & = & -0.09y(t) \\ y(0) & = & 0 \\ y'(0) & = & 2 \end{array} \right.$$

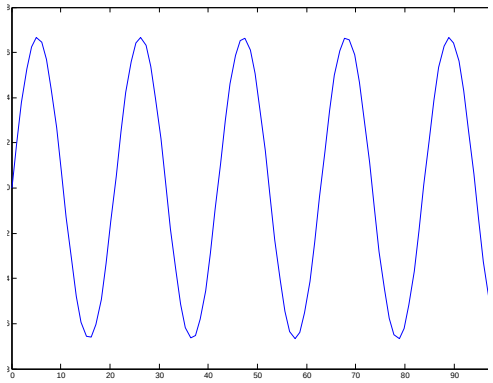
whose exact solution is

$$y(t) = \frac{2}{.3}\sin(0.3t) = 6.667\sin(0.3t)\,.$$

To verify that this is the exact solution we first see that it satisfies the initial conditions:

$$y(0) = 0 \qquad y'(t) = 0.3\frac{2}{.3}\cos(0.3t) \Rightarrow y'(0) = 2\cos(0) = 2\,.$$

To verify that it satisfies the DE we need $y''(t) = -2(0.3)\sin(0.3t)$ which can be written as $y''(t) = -(.3)^2\left[\frac{2}{.3}\sin(.3t)\right] = -0.9y(t)$.
The plot below illustrates the periodicity of the solution.



In the above examples a single unknown function is sought but in some mathematical models we have more than one unknown. An example of this scenario is a population model where there are two interacting species; this is the so-called *predator-prey model*. Here the number of prey, $\rho(t)$, is dependent on the number of predators, $p(t)$, present. In this case we have a first order ODE for the prey and for the predator;

$$\begin{aligned} \rho'(t) &= \left(1 - \frac{p(t)}{\nu}\right)\rho(t) \\ p'(t) &= -\left(1 - \frac{\rho(t)}{\mu}\right)p(t) \end{aligned} \tag{2.6}$$

Note that the equations are nonlinear. These equations must be solved simultaneously because the growth/decay of the prey is dependent upon the number of predators and vice versa. An exact solution to this system is not available but a numerical approximation to the equation is given in Chapter 5.

2.2 General First Order IVP

In § 2.1 we encountered two examples of first order IVPs. Of course there are many others examples such as

$$\mathrm{I}: \left\{ \begin{array}{rcll} y'(t) &=& \sin \pi t & 0 < t \leq 4 \\ y(0) &=& 0\,. & \end{array} \right. \qquad \mathrm{II}: \left\{ \begin{array}{rcll} y'(t) + y^2(t) &=& t & 2 < t \leq 10 \\ y(2) &=& 1\,. & \end{array} \right. \tag{2.7}$$

In the first IVP, the ODE is linear whereas in the second one the ODE is nonlinear in the unknown. Clearly, these IVPs are special cases of the following general IVP.

General first order IVP

Given scalars t_0, y_0, T and a function $f(t, y)$, find $y(t)$ satisfying

$$y'(t) = f(t, y) \quad \text{for } t_0 < t \leq T \tag{2.8a}$$

$$y(t_0) = y_0 \,. \tag{2.8b}$$

Here $f(t, y)$ is the given derivative of $y(t)$ which we refer to as the slope and y_0 is the known value at the initial time t_0. For example, for IVP I in (2.7) we have $f(t, y) = \sin \pi t$, i.e., the slope is only a function of time whereas in IVP II we have $f(t, y) = t - y^2$ so that the slope is a function of both t and y. The ODE in IVP I is linear in the unknown and in IVP II it is nonlinear due to the y^2 term so that both linear and nonlinear differential equations are included in the general equation (2.8a).

For certain choices of $f(t, y)$ we are able to find an analytic solution to (2.8). In the simple case when $f = f(t)$, i.e., f is a function of t and not both t and y, we can solve the ODE exactly if we can obtain an antiderivative of $f(t)$, i.e., if $\int f(t)\, dt$ can be evaluated. For example, for the first IVP in (2.7) we have

$$y'(t) = \sin \pi t \Rightarrow \int y'(t)\, dt = \int \sin \pi t\, dt \Rightarrow y(t) + C_1 = -\frac{1}{\pi} \cos \pi t + C_2$$

and thus the general solution is $y(t) = -\frac{1}{\pi} \cos \pi t + C$. The solution to the differential equation is not unique because C is an arbitrary constant; actually there is a family of solutions which satisfy the differential equation. To determine a unique solution we must specify $y(t)$ at some point such as its initial value. In IVP I in (2.7) $y(0) = 0$ so $y(0) = -\frac{1}{\pi} \cos 0 + C = 0$ which says that the unique solution to the IVP is $y(t) = -\frac{1}{\pi} \cos \pi x + \frac{1}{\pi}$.

If $f(t, y)$ is more complicated than simply a function of t then other techniques are available to try to find the analytic solution. These techniques include methods such as separation of variables, using an integrating factor, etc. Remember that when we write a code to approximate the solution of the IVP (2.8) we always want to test the code on a problem where the exact solution is known so it is useful to know some standard approaches. The following example illustrates how the method of separation of variables is used to solve some first order ODEs; other techniques are explored in the exercises.

Example 2.3. SEPARATION OF VARIABLES FOR FINDING THE ANALYTIC SOLUTION

Consider the differential equation $y'(t) = -ty(t)$ and find its general solution using the method of separation of variables; illustrate the family of solutions graphically. Verify

that the solution satisfies the differential equation and then impose the initial condition $y(0) = 2$ to determine a unique solution to the IVP.

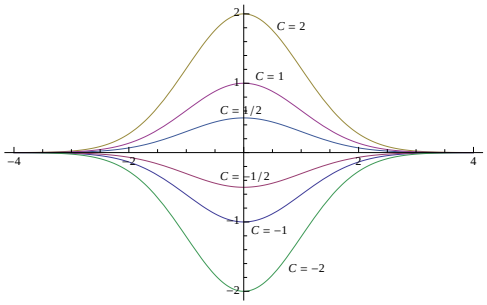
Because $f(t, y)$ is a function of both y and t we can not directly integrate the differential equation with respect to t to obtain the solution because this would require determining $\int ty(t)\ dt$ and y is unknown. For the technique of separation of variables we move all terms involving the unknown to the left-hand side of the equation and all terms involving the independent variable to the other side of the equation. Of course, this technique does not work for all equations but it is applicable for many. For this ODE we rewrite the equation as

$$\frac{dy}{y} = -tdt \quad \Rightarrow \quad \int \frac{dy}{y}\, dt = -\int t\, dt$$

so that we integrate to get the general solution

$$\ln y + C_1 = -\frac{t^2}{2} + C_2 \Rightarrow e^{\ln y + C_1} = e^{-\frac{t^2}{2} + C_2} \Rightarrow e^{C_1} y(t) = e^{-\frac{t^2}{2}} e^{C_2} \Rightarrow y(t) = Ce^{-\frac{t^2}{2}}\,.$$

Note that when we integrate an equation we have an arbitrary constant for each integral. Here we have specifically indicated this but because the sum of two arbitrary constants C_1, C_2 is another arbitrary constant C in the sequel we only give one constant. Since the general solution to this differential equation involves an arbitrary constant C there is an infinite family of solutions which satisfy the differential equation; i.e., one for each choice of C. A family of solutions is illustrated in the figure below; note that as $t \to \pm\infty$ the solution approaches zero.



We can always verify that we haven't made an error in determining the solution by demonstrating that it satisfies the differential equation. Here we have

$$y(t) = Ce^{-\frac{t^2}{2}} \Rightarrow y'(t) = C\left(\frac{-2t}{2}\right)e^{-\frac{t^2}{2}} = -t\left(Ce^{-\frac{t^2}{2}}\right) = -ty(t)$$

so the equation is satisfied.

To determine a unique solution we impose the value of $y(t)$ at some point; here we set $y(0) = 2$ to get the particular solution $y(t) = 2e^{-t^2/2}$ because

$$y(0) = 2, \quad y(t) = Ce^{-\frac{t^2}{2}} \quad \Rightarrow \quad 2 = Ce^0 \quad \Rightarrow \quad C = 2\,.$$

Even if we are unable to determine the analytic solution to (2.8), we can still gain some qualitative understanding of the behavior of the solution. This is done by

the visualization technique of plotting the tangent line to the solution at numerous points (t, y) and is called plotting the direction fields. Recall that the slope of the tangent line to the solution curve is given and is just $f(t, y)$. Mathematical software with graphical capabilities often provide commands for automatically drawing a direction field with arrows which are scaled to indicate the magnitude of the slope; typically they also offer the option of drawing some solutions or streamlines. Using direction fields to determine the behavior of the solution is illustrated in the following example.

Example 2.4. DIRECTION FIELDS

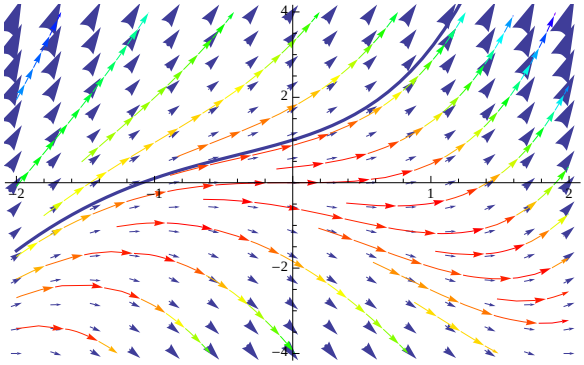
Draw the direction fields for the ODE

$$y'(t) = t^2 + y(t) \quad 0 < t < 4$$

and indicate the specific solution which satisfies $y(0) = 1$.

At each point (t, y) we draw the line with slope $t^2 + y$; this is illustrated in the figure below where numerous streamlines have been sketched. To thread a solution through the direction field start at a point and follow the solution, remembering that solutions don't cross and that nearby tangent lines should be nearly the same.

To see which streamline corresponds to the solution with $y(0) = 1$ we locate the point $(0, 1)$ and follow the tangents; this solution is indicated by a thick black line in the direction field plot below. If a different initial condition is imposed, then we get a different streamline.



Before we discuss methods for approximating the solution of the IVP (2.8) we first need to ask ourselves if our general IVP actually has an analytic solution, even if we are unable to find it. We are only interested in approximating the solution to IVPs which have a unique solution. However, even if we know that a unique solution exists, we may still have unreliable numerical results if the solution of the IVP does not depend continuously on the data. If this is the case, then small changes in the data can cause large changes in the solution and thus roundoff errors in our calculations can produce meaningless results. In this situation we say the IVP is ill-posed or ill-conditioned, a situation we would like to avoid. Luckily, most differential equations that arise from modeling real-world phenomena are well-posed.

The conditions that guarantee well-posedness of a solution to (2.8) are well known and are presented in Theorem 1. Basically the theorem requires that the derivative of $y(t)$ (given by $f(t, y)$) be continuous and, moreover, this derivative is not allowed to change too quickly as y changes. A basic problem in calculus is to determine how much a continuous function changes as the independent variables change; clearly we would like a function to change a small amount as an independent variable changes but this is not always the case. The concept of Lipschitz continuity[1] gives a precise measure of this "degree of continuity". To understand this concept first think of a linear function $g(x) = ax + b$ and consider the effect changing x has on the dependent variable $g(x)$. We have

$$|g(x_1) - g(x_2)| = |ax_1 + b - (ax_2 + b)| = |a|\,|x_1 - x_2|\,.$$

This says that as the independent variable x varies from x_1 to x_2 the change in the dependent variable g is governed by the slope of the line, i.e., $a = g'(x)$. For a general function $g(x)$ Lipschitz continuity on an interval I requires that the magnitude of the slope of the line joining any two points x_1 and x_2 in I must be bounded by a real number. Formally, a function $g(x)$ defined on a domain $D \subset \mathbb{R}^1$ is Lipschitz continuous on D if for any $x_1 \neq x_2 \in D$ there is a constant L such that

$$|g(x_1) - g(x_2)| \leq L|x_1 - x_2|\,,$$

or equivalently

$$\frac{|g(x_1) - g(x_2)|}{|x_1 - x_2|} \leq L\,.$$

Here L is called the Lipschitz constant. This condition says that we must find one constant L which works for all points in the domain. Clearly the Lipschitz constant is not unique; for example, if $L = 5$, then $L = 5.1, 6, 10, 100$, etc. also satisfy the condition. If $g(x)$ is differentiable then an easy way to determine the Lipschitz constant is to find a constant such that $|g'(x)| \leq L$ for all $x \in D$. The linear function $g(x) = ax + b$ is Lipschitz continuous with $L = |a| = |g'(x)|$. Lipschitz continuity is a stronger condition than merely saying the function is continuous so a Lipschitz continuous function is always continuous but the converse is not true. For example, the function $g(x) = \sqrt{x}$ is continuous on $D = [0, 1]$ but is not Lipschitz continuous on D because $g'(x) = 1/(2\sqrt{x})$ is not bounded near the origin.

There are functions which are Lipschitz continuous but not differentiable. For example, consider the continuous function $g(x) = |x|$ on $D = [-1, 1]$. Clearly it is not differentiable on D because it is not differentiable at $x = 0$. However, it is Lipschitz continuous with $L = 1$ because the magnitude of the slope of the secant line between any two points is always less than or equal to one. Consequently, *Lipschitz continuity is a stronger requirement than continuity but a weaker one than differentiability.*

For the existence and uniqueness result for (2.8), we need $f(t, y)$ to be Lipschitz continuous in y so we need to extend the above definition by just holding t fixed. Formally, for fixed t we have that a function $g(t, y)$ defined for y in a prescribed

[1] Named after the German mathematician Rudolf Lipschitz (1832-1903).

domain is Lipschitz continuous in the variable y if for any $(t, y_1), (t, y_2)$ there is a constant L such that

$$|g(t, y_1) - g(t, y_2)| \leq L|y_1 - y_2|\,. \tag{2.9}$$

We are now ready to state the theorem which guarantees existence and uniqueness of a solution to (2.8) as well as guaranteeing that the solution depends continuously on the data; i.e., the problem is well-posed. Note that $y(t)$ is defined on $[t_0, T]$ whereas $f(t, y)$ must be defined on a domain in $\mathbb{R}^2$. Specifically the first argument t is in $[t_0, T]$ but y can be any real number so that $D = \{(t, y) \mid t \in [t_0, T], y \in \mathbb{R}^1\}$; a shorter notation for expressing D is $D = [t_0, T] \times \mathbb{R}^1$ which we employ.

Theorem 2.1 : Existence and uniqueness for IVP (2.8)

Let $D = [t_0, T] \times \mathbb{R}^1$ and assume that $f(t, y)$ is continuous on D and is Lipschitz continuous in y on D; i.e., it satisfies (2.9). Then the IVP (2.8) has a unique solution in D and moreover, the problem is well-posed.

In the sequel we only consider IVPs which are well-posed, that is, which have a unique solution that depends continuously on the data.

2.3 Discretization

Even if we know that a solution to (2.8) exists for some choice of $f(t, y)$, we may not be able to find the *closed form solution* to the IVP; that is, a representation of the solution in terms of a finite number of simple functions. Even for the simplified case of $f(t, y) = f(t)$ this is not always possible. For example, consider $f(t) = \sin t^2$ which has no explicit formula for its antiderivative. In fact, a symbolic algebra software package like Mathematica gives the antiderivative of $\sin t^2$ in terms of the Fresnel Integral which is represented by an infinite power series near the origin; consequently there is no closed form solution to the problem. Although there are numerous techniques for finding the analytic solution of first order differential equations, we are unable to easily obtain closed form analytic solutions for many equations. When this is the case, we must turn to a *numerical approximation* to the solution where we give up finding a formula for the solution at all times and instead find an approximation at a set of distinct times. Discretization is the name given to the process of converting a continuous problem into a form which can be used to obtain numerical approximations.

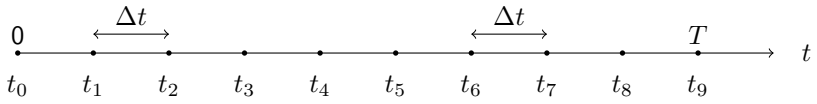
Probably the most obvious approach to discretizing a differential equation is to approximate the derivatives in the equation by difference quotients to obtain a difference equation which involves only differences in function values. The solution to the difference equation will not be a continuous function but rather a discrete function which is defined over a finite set of points. When plotting the discrete solution one often draws a line through the points to get a continuous curve but

remember that interpolation must be used to determine the solution at points other than at the given grid points.

Because the difference equation is defined at a finite set of points we first discretize the time domain $[t_0, T]$; alternately, if our solution depended on the spatial domain x instead of t we would discretize the given spatial interval. For now we use $N+1$ evenly spaced points t_n, $n = 0, 1, 2, \ldots, N$

$$t_1 = t_0 + \Delta t, \quad t_2 = t_1 + \Delta t, \quad \cdots, \quad t_N = t_{N-1} + \Delta t = T\,,$$

where $\Delta t = (T - t_0)/N$ is called the step size or time step. This is illustrated below where the time domain is $[0, T]$ and $N = 9$.



The strategy to approximate the solution to an IVP at each point $t_n \in (t_0, T]$ is to use the given initial information at t_0, i.e., y_0 and the slope evaluated at t_0 (i.e., $f(t_0, y_0)$), to get an approximate at t_1. Then the information at t_1 and possibly t_0 is used to get an approximation at t_2. This process is continued until we have an approximation at the final time $t = T$. Each method that we consider has a different way to approximate the solution at the next time step. In the next chapter we consider the simplest methods, the forward and backward Euler methods, and in the following chapter we survey classes of methods for approximating the solution of the IVP (2.8).

Because the approximation is a *discrete function* that is only known at each t_n, $n = 0, 1, \ldots, N$, we need different notation from the one used for the exact solution to the IVP. In the sequel we use small case letters for the solution to the IVP (such as $y(t)$, $p(t)$, etc.) and capital letters for the approximation. Because the approximate solution is only defined at each t_n, $n = 0, 1, \ldots, N$ we use a superscript to denote the particular value of t_n. For example, $Y^2 \approx y(t_2)$, $Y^n \approx y(t_n)$. We know the given solution y_0 at t_0 so we set $Y^0 = y_0$.

Once we have an approximation for a fixed value of Δt, how do we know that our numerical results are accurate? For example, in the left plot in Figure 2.1 we graph an exact solution (the continuous curve) to a specific IVP and a discrete approximation for $\Delta t = 0.5$. The approximate solution is plotted only at the points where it is determined. Although the approximate solution has the same general shape as the exact solution, from this plot we are unable to say if our discrete solution is correct. If we obtain additional approximations as the uniform time step is reduced, then the plot on the right in Figure 2.1 suggests that the approximate solution approaches the exact solution in some sense as $\Delta t \to 0$. However, this does not confirm that the numerical results are correct. The reason for this is that different methods produce approximations which converge to the exact solution more rapidly than other methods. The only way to confirm that the results are correct is to compare the numerical rate of convergence with the theoretical rate of convergence for a problem where the exact solution is known. Consequently when we

learn new methods we prove or merely state the theoretical rate of convergence. In § 1.2.3 the approach for determining the numerical rate of convergence is explained.

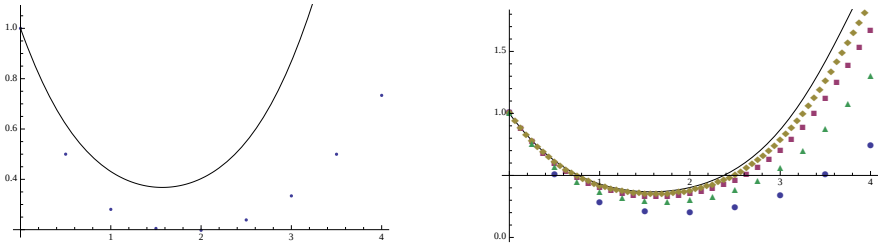


Figure 2.1: The exact solution to an IVP is shown as a solid curve. In the figure on the left a discrete solution using $\Delta t = 0.5$ is plotted. From this plot, it is not possible to say that the discrete solution is approaching the exact solution. However, in the figure on the right the discrete solutions for $\Delta t = 0.5, 0.25, 0.125$, and 0.625 are plotted. From this figure, the discrete approximations appear to be approaching the exact solution as Δt decreases.

When we implement a numerical method on a computer the error we make is due to both roundoff and discretization error. Rounding error is due to using a computer which has finite precision. First of all, we may not be able to represent a number exactly; this is part of roundoff error and is usually called *representation error*. Even if we use numbers which can be represented exactly on the computer, we encounter rounding errors when these numbers are manipulated such as when we divide two integers like 3 and 7. In some problems, roundoff error can accumulate in such a way as to make our results meaningless. Discretization error is caused by replacing the continuous problem with a discrete problem. For example, a discretization error results when we replace $y'(t_{n+1})$ by the difference quotient $\big(y(t_{n+1}) - y(t_n)\big)/\Delta t$. We have control over discretization error by choosing a method which approximates quantities like the derivative more accurately. Our main control over roundoff error is the computer precision we use in the implementation.

2.4 Higher Order IVPs

The general IVP (2.8) contains a first order ODE and in the next two chapters we look at methods for approximating its solution. What do we do if we have a higher order IVP such as the harmonic oscillator equation given in Example 2.2? The answer is we write the higher order IVP as a system of first order IVPs. Then the methods we learn to approximate the solution of the IVP (2.8) can be applied directly; this is explored in Chapter 5.

Suppose we have the second order IVP

$$\begin{aligned} y''(t) &= 2y'(t) - \sin(\pi y) + 4t \quad 0 < t \leq 2 \\ y(0) &= 1 \\ y'(0) &= 0\,, \end{aligned}$$

where now the right-hand side is a function of t, y and y'. The methods we learn in Chapter 3 and Chapter 4 only apply to first order IVPs. However, we can easily convert this second order IVP into two coupled first order IVPs. To do this, we let $w_1(t) = y(t)$, $w_2(t) = y'(t)$ and substitute into the equations and initial conditions to get a first order system for w_1, w_2

$$\begin{array}{rcll} w_1'(t) & = & w_2(t) & 0 < t \le 2 \\ w_2'(t) & = & 2w_2(t) - \sin(\pi w_1) + 4t & 0 < t \le 2 \\ w_1(0) & = & 1 \qquad w_2(0) = 0\,. & \end{array}$$

Note that these two differential equations are *coupled*, that is, the differential equation for w_1 depends on w_2 and the equation for w_2 depends on w_1.

In general, if we have the pth order IVP for $y(t)$

$$\begin{array}{rcl} y^{[p]}(t) & = & f(t, y, y', y'', \cdots, y^{[p-1]}) \quad t_0 < t \le T \\ y(t_0) & = & \alpha_1, \quad y'(t_0) = \alpha_2, \quad y''(t_0) = \alpha_3, \quad \cdots \quad y^{[p-1]}(t_0) = \alpha_p \end{array}$$

then we convert it to a system of p first-order IVPs by letting $w_1(t) = y(t)$, $w_2(t) = y'(t), \cdots, w_p(t) = y^{[p-1]}(t)$ which yields the first order coupled system

$$\begin{array}{rcl} w_1'(t) & = & w_2(t) \\ w_2'(t) & = & w_3(t) \\ & \vdots & \\ w_{p-1}'(t) & = & w_p(t) \\ w_p'(t) & = & f(t, w_1, w_2, \ldots, w_p) \end{array} \tag{2.10}$$

along with the initial conditions $w_k = \alpha_k$, $k = 1, 2, \ldots, p$. Thus any higher order IVP that we encounter can be transformed into a coupled system of first order IVPs.

Example 2.5. CONVERTING A HIGH ORDER IVP INTO A SYSTEM

Write the fourth order IVP

$$y^{[4]}(t) + 2y''(t) + 4y(t) = 5 \quad y(0) = 1, y'(0) = -3, y''(0) = 0, y'''(0) = 2$$

as a system of first order equations.

We want four first order differential equations for $w_i(t)$, $i = 1, 2, 3, 4$; to this end let $w_1 = y$, $w_2 = y'$, $w_3 = y''$, and $w_4 = y'''$. Using the first two expressions we have $w_1' = w_2$, and the second and third gives $w_2' = w_3$, the third and fourth gives $w_3' = w_4$ and the original differential equation provides the last first order equation $w_4' + 2w_3 + 4w_1 = 5$. The system of equations is thus

$$\begin{array}{rcl} w_1'(t) - w_2(t) & = & 0 \\ w_2'(t) - w_3(t) & = & 0 \\ w_3'(t) - w_4(t) & = & 0 \\ w_4' + 2w_3 + 4w_1 & = & 5 \end{array}$$

along with the initial conditions

$$w_1(0) = 1, \quad w_2(0) = -3, \quad w_3(0) = 0, \quad \text{and } w_4(0) = 2.$$

Chapter 3

The Euler Methods

There are many approaches to deriving discrete methods for the general first order IVP (2.8) but the simplest methods use the slope of a secant line to approximate the derivative in (2.8a). In this chapter we consider two methods for solving the prototype IVP (2.8) which are obtained by using this approximation to $y'(t)$ at two different values of t. We see that the two methods require very different implementations and have different stability properties.

We begin this chapter with the forward Euler method which is described by a simple formula but also has a graphical interpretation. Numerical examples which demonstrate how the method is applied by hand are provided before a computer implementation is discussed. The discretization error for forward Euler is derived in detail by first obtaining a local truncation error which is caused by approximating $y'(t)$ and then obtaining the global error which is due to the local truncation error and an accumulated error over the time steps. We see that the global error is one order of magnitude less than the local truncation error which is the typical relationship we see for the methods described here. A computer implementation of the forward Euler method is given and several examples demonstrate that the numerical results agree with the theoretical rate of convergence. However, for one example and certain choices of step size, the forward Euler produces results which oscillate.

The second method considered here is the backward Euler method which has strikingly different properties than the forward Euler method. The implementation for the prototype IVP (2.8) typically requires solving a nonlinear equation at each time step compared with a linear equation for the forward Euler method. However, it does not produce unreliable results for some problems and some choices of the time step as the forward Euler method does. Lastly we briefly discuss the concepts of consistency, stability and convergence of numerical methods to begin to understand why numerical methods may produce oscillating or unbounded results.

3.1 Forward Euler Method

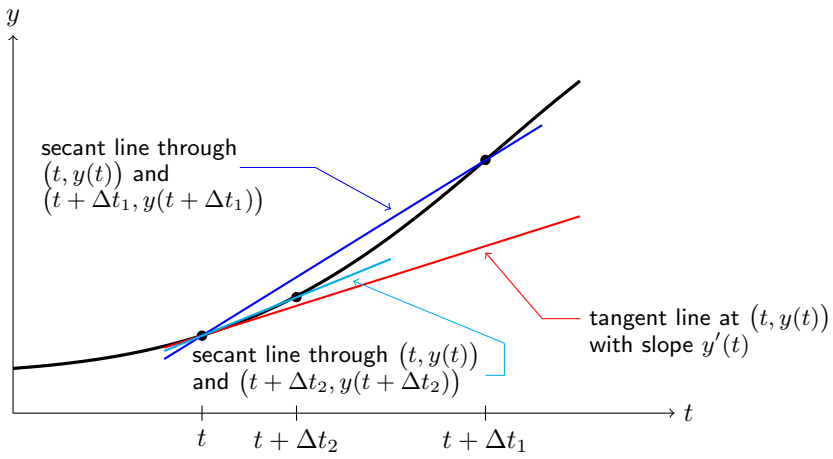
The forward Euler method is probably the best known and simplest method for approximating the solution to the IVP (2.8). This method was named after Leonhard Euler[1] and is often just referred to as "the Euler method." There are many ways to derive the method but in this chapter we use the slope of the secant line between $(t_n, y(t_n)$ and $(t_{n+1}, y(t_{n+1})$ to approximate $y'(t_n)$. Because we use a secant line approximation, this gives us a graphical interpretation of the method. In this section we explore an implementation of the algorithm, demonstrate that the method converges linearly and provide numerical results.

3.1.1 Derivation and Graphical Interpretation

The forward Euler method can be derived from several different viewpoints, some of which we explore in this and the next chapter. The simplest approach is just to use a secant line to approximate the derivative $y'(t)$. Recall that the definition of $y'(t)$ is

$$y'(t) = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t},$$

so if Δt is small so that the point $\big(t + \Delta t, y(t + \Delta t)\big)$ is close to $\big(t, y(t)\big)$, then the slope of the secant line is a good approximation to the slope $f(t, y)$ of the tangent line at $\big(t, y(t)\big)$. In the figure below we compare the secant line approximation to the actual slope $y'(t)$ for two different choices of Δt; here $\Delta t_2 = \Delta t_1/3$ so the slope of the secant line joining $\big(t, y(t)\big)$ and $\big(t + \Delta t_2, y(\Delta t_2)\big)$ (represented by the cyan line) is much closer to $y'(t)$ (tangent line represented in red) then the secant line joining $\big(t, y(t)\big)$ and $\big(t + \Delta t_1, y(\Delta t_1)\big)$ (represented by blue line). Of course, how small we require Δt also depends on how rapidly $y'(t)$ is changing.



[1] Euler (1707-1783) was a Swiss mathematician and physicist.

We know that the secant line joining the points $\big(t, y(t)\big)$ and $\big(t+\Delta t, y(t+\Delta t)\big)$ is an approximation to $y'(t)$, i.e.,

$$y'(t) \approx \frac{y(t+\Delta t) - y(t)}{\Delta t} \tag{3.1}$$

and if Δt is small then we expect this difference quotient to be a good approximation. If we set $t = t_n$ in (3.1) and $y(t_{n+1}) = y(t_n + \Delta t)$ then

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

where we have used the differential equation $y'(t_n) = f\big(t_n, y(t_n)\big)$. This suggests the following numerical method for the solution of (2.8) which is called the forward Euler method where we denote the approximate solution at t_n as Y^n and set $Y^0 = y(t_0)$.

Forward Euler: $$Y^{n+1} = Y^n + \Delta t f(t_n, Y^n)\,, \quad n = 0, 1, 2, \ldots, N-1 \tag{3.2}$$

The term "forward" is used in the name because we write the equation at the point t_n and difference forward in time to t_{n+1}; this implies that the given slope f is evaluated at the known point (t_n, Y^n).

To implement the method we know that $Y^0 = y_0$ is the solution at t_0 so we can evaluate the known slope there, i.e., $f(t_0, Y^0)$. Then the solution at t_1 is given by

$$Y^1 = Y^0 + \Delta t f(t_0, Y^0)\,.$$

For the next step, we know $Y^1 \approx y(t_1)$ and so we must evaluate the slope at (t_1, Y^1) to get

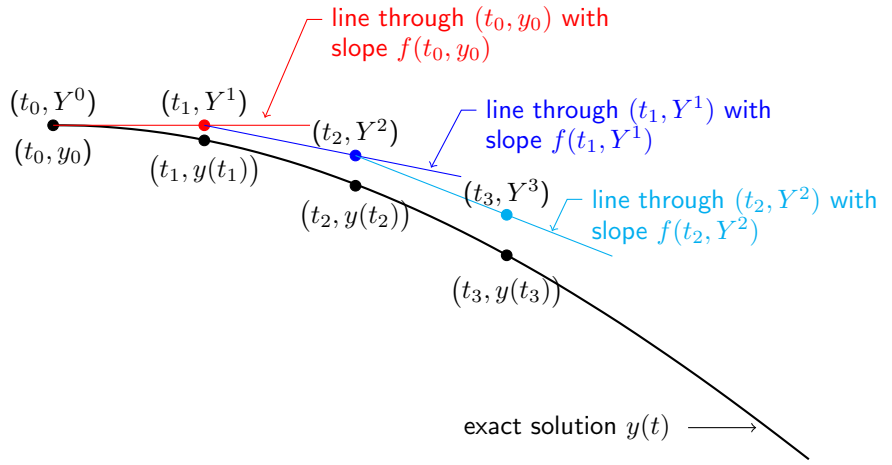
$$Y^2 = Y^1 + \Delta t f(t_0, Y^1)\,.$$

The procedure is continued until the desired time is reached.

A graphical interpretation of the forward Euler method is shown in the figure below.To start the method, we write the tangent line to the solution curve at $(t_0, y_0) = (t_0, Y^0)$ which has slope $y'(t_0) = f(t_0, Y^0)$; the equation is

$$w(t) - Y^0 = f(t_0, Y^0)(t - t_0)\,.$$

If Y^1 denotes the point on this line corresponding to $t = t_1$ then $Y^1 - Y^0 = f(t_0, Y^0)(t_1 - t_0) = \Delta t f(t_0, Y^0)$ which is just Euler's equation for the approximation Y^1 to $y(t_1)$. Now for the second step we don't have a point on the exact solution curve to compute the tangent line but if Δt is small, then $f(t_1, Y^1) \approx f(t_1, y(t_1)) = y'(t_1)$. So we write the equation passing through (t_1, Y^1) with slope $f(t_1, Y^1)$ and evaluate it at t_2 to get $Y^2 - Y^1 = \Delta t f(t_1, Y^1)$ which again is just the formula for Y^2 from (3.2). It is important to realize that at the second step we do not have the exact slope $f\big(t_1, y(t_1)\big)$ of the tangent line to the solution curve but rather the approximation $f(t_1, Y^1)$. This is true for all subsequent steps.



In the following example we implement two steps of the forward Euler method by hand and then demonstrate how the approximation relates to the tangent line of the solution curve.

Example 3.1. FORWARD EULER METHOD

Apply the forward Euler method to the IVP

$$y'(t) = -2y,\ 0 < t \leq T \qquad y(0) = 2$$

to approximate the solution at $t = 0.2$ using a step size of $\Delta t = 0.1$. Then calculate the error at $t = 0.2$ given the exact solution $y(t) = 2e^{-2t}$. Does the point (t_1, Y^1) lie on the tangent line to $y(t)$ at $t = t_0$? Does the point (t_2, Y^2) lie on the tangent line to $y(t)$ at $t = t_1$? Justify your answer.

To find the approximation at $t = 0.2$ using a time step of $\Delta t = 0.1$, we first have to apply Euler's method to determine an approximation at $t = 0.1$ and then use this to approximate the solution at $t = 0.2$. From the initial condition we set $Y^0 = 2$ and from the differential equation we have $f(t, y) = -2y$. Applying the forward Euler method gives

$$Y^1 = Y^0 + \Delta t f(t_0, Y^0) \Rightarrow Y^1 = 2 + 0.1 f(0, 2) = 2 + 0.1(-4) = 1.6$$

and thus

$$Y^2 = Y^1 + \Delta t f(t_1, Y^1) \Rightarrow Y^2 = 1.6 + 0.1 f(.1, 1.6) = 1.6 - 0.32 = 1.28\,.$$

The exact solution at $t = 0.2$ is $y(0.2) = 2e^{-0.4} \approx 1.34064$ so the error is $|1.34064 - 1.28| = 0.06064$.

The equation of the tangent line to $y(t)$ at $t = 0$ passes through the point $(0, 2)$ and has slope $y'(0) = -4$. Thus the equation of the tangent line is $w - 2 = -4(t - 0)$ and at $t = 0.1$ we have that $w = 2 - .4 = 1.6$ which is Y^1 so the point $(0.1, Y^1)$ is on the tangent line to the solution curve at $t = 0$; this is to be expected from the graphical interpretation of the forward Euler method. However, the point (t_2, Y^2) is not

on the tangent line to the solution curve $y(t)$ at $t = 0.1$ but rather on a line passing through the point (t_1, Y^1) with slope $f(t_1, Y^1)$. The actual slope of the tangent line to the solution curve at $\big(t_1, y(t_1)\big)$ is $-2y(t_1) = -2\big(2e^{-2(.1)}\big) = -4e^{-.2} = 3.2749$ whereas we approximate this by $f(t_1, Y^1) = 3.2$.

In the case when $f(t, y)$ is only a function of t or it is linear in y we can get a general formula for Y^n in terms of Y^0 and Δt so that the intermediate time steps do not have to be computed. The following example illustrates how this formula can be obtained.

Example 3.2. GENERAL SOLUTION TO FORWARD EULER DIFFERENCE EQUATION FOR A SPECIAL CASE

Consider the IVP

$$y'(t) = -\lambda y \quad 0 < t \le T, \quad y(0) = y_0$$

whose exact solution is $y(t) = y_0 e^{-\lambda t}$. Find the general solution for Y^n in terms of Y^0 and Δt for the forward Euler method.

For the forward Euler method we have

$$Y^1 = Y^0 + \Delta t(-\lambda Y^0) = (1 - \lambda \Delta t)Y^0 \,.$$

Similarly,

$$Y^2 = (1 - \lambda \Delta t)Y^1 = (1 - \lambda \Delta t)^2 Y^0, \quad Y^3 = (1 - \lambda \Delta t)Y^2 = (1 - \lambda \Delta t)^3 Y^0 \,.$$

Continuing in this manner we see that

$$Y^n = (1 - \lambda \Delta t)^n Y^0 \,.$$

In the next example we use this general formula to compare the results at a fixed time for a range of decreasing values of the uniform time step. As can be seen from the results, as $\Delta t \to 0$ the error in the solution tends to zero which implies the approximate solution is converging to the exact solution.

Example 3.3. COMPARING NUMERICAL RESULTS AS THE TIME STEP IS DECREASED

Use the general formula from Example 3.2 to approximate the solution to the IVP at $t = 1$ when $\lambda = 5$, $y_0 = 2$. Compare the results for $\Delta t = 1/20, 1/40, \ldots, 1/320$ and discuss. Then determine the time step necessary to guarantee that the relative error is less than 1%.

For this problem the general solution is $Y^n = 2(1 - 5\Delta t)^n$ and the exact solution is $y(t) = 2e^{-5t}$ so the exact solution at $t = 1$ is $y(1) = 0.013475893998$. The relative error in the table below is computed by calculating $|Y^n - y(1)|/|y(1)|$. For each value of Δt, the number of time steps, the approximate solution Y^n and the magnitude of the relative error are reported.

Δt	n	Y^n	Relative Error
1/20	20	$6.342\ 10^{-3}$	0.52935
1/40	40	$9.580\ 10^{-3}$	0.28912
1/80	80	$1.145\ 10^{-2}$	0.15048
1/160	160	$1.244\ 10^{-2}$	0.076691
1/320	320	$1.295\ 10^{-2}$	0.038705

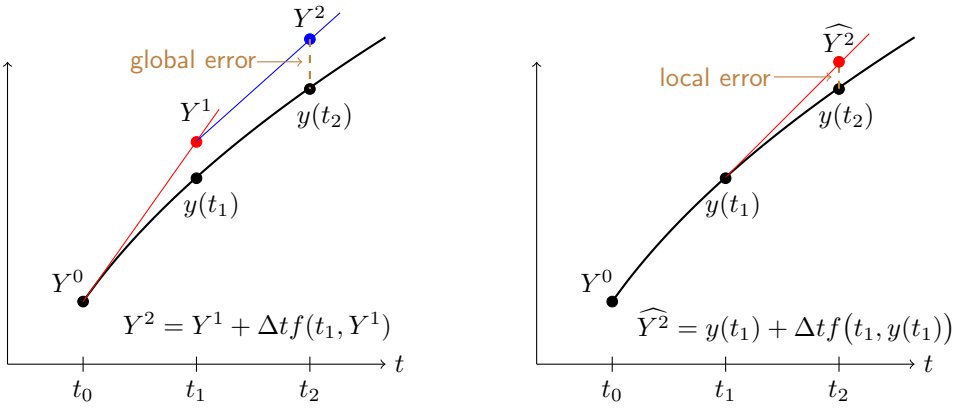
If the numerical solution is converging to the exact solution then the relative error at a fixed time should approach zero as Δt gets smaller. As can be seen from the table, the approximations tend to zero monotonically as Δt is halved and, in fact, the errors are approximately halved as we decrease Δt by half. This is indicative of linear convergence. At $\Delta t = 1/320$ the relative error is approximately 3.87% so for $\Delta t = 1/640$ we expect the relative error to be approximately 1.9% so cutting the time step again to $1/1280$ should give a relative error of $< 1\%$. To confirm this, we do the calculation $Y^n = 2(1 - 5/1280)^{1280}$ and get a relative error of approximately 0.97%.

3.1.2 Discretization errors

We know that when we implement a numerical method on a computer the error we make is due to both *roundoff* and *discretization* error. Here we are mainly concerned with discretization error and when we derive error estimates we assume that no rounding error exists. We know that at t_0 the approximate solution agrees with the exact solution. Using forward Euler to compute an approximation at t_1 incurs an error in the approximation due to the fact that we have used the difference quotient $(y(t_1) - y(t_0))/\Delta t$ to approximate $y'(t_0)$. However at t_2 and subsequent points the discretization error comes from two sources. The first source of error is the difference quotient approximation to $y'(t)$ and the second is because we have started from the incorrect point, i.e., we did not start on the exact solution curve as we did in calculating Y^1. The global discretization error at a point t_n is the magnitude of the actual error $|y(t_n) - Y^n|$ whereas the local truncation error or local discretization error is the error made because we solve the difference equation rather than the actual differential equation. In other words the local truncation error at t_n measures how well the approximate solution matches the exact solution *if the two solutions are the same at* t_{n-1}.The figure below illustrates the two errors graphically for the forward Euler method. The plot on the left demonstrates the global error at t_2 which is just $|y(t_2) - Y^2|$ where Y^2 is found by applying the forward Euler formula $Y^2 = Y^1 + \Delta t f\big(t_1, Y^1\big)$. The figure on the right demonstrates the local error at t_2; note that the starting point used is not (t_1, Y^1) but the exact solution $\big(t_1, y(t_1)\big)$. From the figure we see that the local error is $|y(t_2) - \big[y(t_1) + \Delta t f\big(t_1, y(t_1)\big)\big]$ which is just the remainder when the exact solution is plugged into the difference equation.

Calculating local truncation error

To measure the local truncation error, plug the exact solution into the difference equation and calculate the remainder.



Our strategy for analytically determining the global error for the forward Euler method is to first quantify the local truncation error in terms of Δt and then use this result to determine the global error. To determine a formula for the local truncation error for the forward Euler method we substitute the exact solution to (2.8a) into the difference equation (3.2) and calculate the remainder. If τ_{n+1} denotes the local truncation error at the $(n+1)$st time step then

$$\tau_{n+1} = y(t_{n+1}) - \big[y(t_n) + \Delta t f\big(t_n, y(t_n)\big)\big]\,. \tag{3.3}$$

In order to combine terms in (3.3) we need all terms to be evaluated at the same point $\big(t_n, y(t_n)\big)$. The only term not at this point is the exact solution $y(t_{n+1}) = y(t_n + \Delta t)$ so we use a Taylor series with remainder (see Appendix) for this term; we have

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

From the differential equation evaluated at t_n, we have $y'(t_n) = f\big(t_n, y(t_n)\big)$ so we substitute this into the Taylor series expansion for $y(t_{n+1})$. We then put the expansion into the expression (3.3) for the truncation error to yield

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

If $y''(t)$ is bounded on $[0, T]$, say $|y''(t)| \le M$, and $T = t_0 + N\Delta t$, then we have

$$\tau = \max_{1 \le n \le N} |\tau_n| \le \frac{M}{2} (\Delta t)^2\,, \tag{3.4}$$

where τ denotes the largest truncation error of all the N time steps. We say that the local truncation error for Euler's method is *order* $(\Delta t)^2$ which we write as $\mathcal{O}(\Delta t^2)$ and say that the rate is *quadratic*. This implies that the local error is proportional to the square of the step size; i.e., it is a constant times the square of the step size which in turn says that if we compute the local error for Δt then the local error using $\Delta t/2$ is reduced by approximately $(1/2)^2 = 1/4$. Remember, however, that this is not the global error but rather the error made because we have used a finite difference quotient to approximate $y'(t)$.

We now turn to estimating the global error in the forward Euler method. We should expect to only be able to find an upper bound for the error because if we can find a formula for the exact error, then we can calculate this and add it to the approximation to get the exact solution. The proof for the global error for the forward Euler method is a bit technical but it is the only global error estimate that we derive because the methods we consider follow the same relationship between the local and global error as the Euler method.

Our goal is to demonstrate that the global discretization error for the forward Euler method is $\mathcal{O}(\Delta t)$ which says that the method is first order, i.e., *linear* in Δt. At each step we make a local error of $\mathcal{O}(\Delta t)^2$ due to approximating the derivative in the differential equation; at each fixed time we have the accumulated errors of all previous steps and we want to demonstrate that this error does not exceed a constant times Δt.

Theorem 3.1 provides a formal statement and proof for the global error of the forward Euler method. Note that one hypothesis is that $f(t, y)$ must be Lipschitz continuous in y which is also assumed to guarantee existence and uniqueness of the solution to the IVP (2.8) so it is a natural assumption. We also assume that $y(t)$ possesses a bounded second derivative because we need to use the local truncation error given in (3.4); however, this condition can be relaxed but it is adequate for our needs.

Theorem 3.1 : Global error estimate for the forward Euler method

Let $D = [t_0, T] \times \mathbb{R}^1$ and assume that $f(t, y)$ is continuous on D and is Lipschitz continuous in y on D; i.e., it satisfies (2.9) with Lipschitz constant L. Also assume that there is a constant M such that

$$|y''(t)| \leq M \quad \text{for all } t \in [t_0, T]\,.$$

Then the global error at each point t_n satisfies

$$|y(t_n) - Y^n| \leq C\Delta t \quad \text{where } C = \frac{M}{2L}(e^{TL} - 1)\,;$$

thus the forward Euler method converges linearly.

Proof. Let E_n represent the global discretization error at the specific time t_n, i.e., $E_n = |y(t_n) - Y^n|$. The steps in the proof are summarized as follows.

Step I. Use the definition of the local truncation error τ_n to demonstrate that the global error satisfies

$$E_n \leq K\, E_{n-1} + |\tau_n| \quad \text{for } K = 1 + \Delta t L;$$

that is, the error at a step is bounded by a constant times the error at the previous step plus the absolute value of the local truncation error. If τ is the maximum of all $|\tau_n|$, we have

$$E_n \leq K\, E_{n-1} + \tau \quad \text{for } K = 1 + \Delta t L\,. \tag{3.5}$$

Step II. Apply (3.5) recursively and use the fact that $E_0 = 0$ to get

$$E_n \leq \tau \sum_{i=0}^{n-1} K^i\,. \tag{3.6}$$

Step III. Recognize that the sum in (3.6) is a geometric series whose sum is known to get

$$E_n \leq \frac{\tau}{\Delta t L}\left[(1 + \Delta t L)^n - 1\right]\,. \tag{3.7}$$

Step IV. Use the Taylor series expansion of $e^{\Delta t L}$ near zero to bound $(1 + \Delta t L)^n$ by $e^{n\Delta t L}$ which in turn is less than e^{TL}.

Step V. Use the bound (3.4) for τ to get the final result

$$E_n \leq \frac{M\Delta t}{2L}(e^{TL} - 1) = C\Delta t \quad \text{where } C = \frac{M}{2L}(e^{TL} - 1)\,. \tag{3.8}$$

We now give the details for each step. For the first step we use the fact that the local truncation error is the remainder when we substitute the exact solution into the difference equation; i.e.,

$$\tau_n = y(t_n) - y(t_{n-1}) - \Delta t f\big(t_{n-1}, y(t_{n-1})\big)\,.$$

To get the desired expression for E_n we solve for $y(t_n)$ in the above expression, substitute into the definition for E_n and use the triangle inequality; then we use the forward Euler scheme for Y^n and Lipschitz continuity (2.9). We have

$$\begin{aligned}
E_n &= |y(t_n) - Y^n| \\
&= \big|\big[\tau_n + y(t_{n-1}) + \Delta t f\big(t_{n-1}, y(t_{n-1})\big)\big] - Y^n\big| \\
&= \big|\tau_n + y(t_{n-1}) + \Delta t f\big(t_{n-1}, y(t_{n-1})\big) - \big[Y^{n-1} + \Delta t f(t_{n-1}, Y^{n-1})\big]\big| \\
&\le \big|\tau_n\big| + |y(t_{n-1}) - Y^{n-1}| + \Delta t |f\big(t_{n-1}, y(t_{n-1})\big) - f(t_{n-1}, Y^{n-1})\big| \\
&\le |\tau_n| + E_{n-1} + \Delta t L |y(t_{n-1}) - Y^{n-1}| = |\tau_n| + (1 + \Delta t L) E_{n-1}\,.
\end{aligned}$$

In the final step we have used the Lipschitz condition (2.9) which is a hypothesis of the theorem. Since $|\tau_n| \le \tau$, we have the desired result.

For the second step we apply (3.5) recursively

$$\begin{aligned}
E_n &\le K E_{n-1} + \tau \le K[K E_{n-2} + \tau] + \tau = K^2 E_{n-2} + (K+1)\tau \\
&\le K^3 E_{n-3} + (K^2 + K + 1)\tau \\
&\le \cdots \\
&\le K^n E_0 + \tau \sum_{i=0}^{n-1} K^i\,.
\end{aligned}$$

Because we assume for analysis that there are no roundoff errors, $E_0 = |y_0 - Y_0| = 0$ and we are left with $\tau \sum_{i=0}^{n-1} K^i$.

For the third step we simplify the sum by noting that it is a geometric series of the form $\sum_{i=0}^{n-1} a r^i$ with $a = \tau$ and $r = K$. From calculus we know that the sum is given by $a(1 - r^n)/(1 - r)$ so that if we use the fact that $K = 1 + \Delta t L$ we arrive at the result (3.7)

$$E_n \le \tau \left(\frac{1 - K^n}{1 - K}\right) = \tau \left(\frac{K^n - 1}{K - 1}\right) = \frac{\tau}{\Delta t L}\Big[(1 + \Delta t L)^n - 1\Big]\,.$$

To justify the fourth step we know that for real z the Taylor series expansion $e^z = 1 + z + z^2/2! + \cdots$ near zero implies that $1 + z \le e^z$ so that $(1 + z)^n \le e^{nz}$. If we set $z = \Delta t L$ we have $(1 + \Delta t L)^n \le e^{n \Delta t L}$ so that

$$E_n \le \frac{\tau}{\Delta t L}(e^{n \Delta t L} - 1)\,.$$

For the final step we know from the hypothesis of the theorem that $|y''(t)| \le M$ so $\tau \le M \Delta t^2/2$. Also n in E_n is the number of steps taken from t_0 so $n \Delta t = t_n \le T$ where T is the final time and so $e^{n \Delta t L} \le e^{TL}$. Combining these results gives the desired result (3.8).

■

In general, the calculation of the local truncation error is straightforward (but sometimes tedious) whereas the proof for the global error estimate is much more involved. However, for the methods we consider, if the local truncation error is $\mathcal{O}(\Delta t)^r$ then we expect the global error to be one power of Δt less, i.e., $\mathcal{O}(\Delta t)^{r-1}$. When performing numerical simulations we need to demonstrate that the numerical rate of convergence agrees with the theoretical rate. This gives us confidence that the numerical scheme is implemented properly.

3.1.3 Numerical computations

In this section we provide some numerical simulations for IVPs of the form (2.8) using the forward Euler method. Before providing the simulation results we discuss the computer implementation of the forward Euler method. For the simulations presented we choose problems with known analytic solutions so that we can compute numerical rates of convergence and compare with the theoretical result given in Theorem 3.1. To do this we compute approximate solutions for a sequence of step sizes where $\Delta t \to 0$ and then compute a numerical rate of convergence using § 1.6. As expected, the numerical rate of convergence is linear; however, we see that the forward Euler method does not provide reliable results for all choices of Δt for some problems. Reasons for this failure will be discussed in § 3.4.

Computer implementation

For the computer implementation of a method we first identify what information is problem dependent. The information which changes for each IVP (2.8) is the interval $[t_0, T]$, the initial condition y_0, the given slope $f(t, y)$ and the exact solution if an error calculation is performed. From the examples calculated by hand, we know that we approximate the solution at t_1, then at t_2, etc. so implementation requires a single loop over the number of time steps, say N. However, the solution should not be stored for all times; if it is needed for plotting, etc., then the time and solution should be written to a file to be used later. For a single equation it does not take much storage to keep the solution at each time step but when we encounter systems and problems in higher dimensions the storage to save the entire solution can become prohibitively large so it is not good to get in the practice of storing the solution at each time.

The following pseudocode gives an outline of one approach to implement the forward Euler method using a uniform time step.

Algorithm 3.1 : Forward Euler Method

Define: external function for the given slope $f(t, y)$ and for the exact solution for error calculation

Input: the initial time, t_0, the final time, T, the initial condition, y_0, and the uniform time step Δt

Set:

$t = t_0$

$y = y_0$

Time step loop:

do while $t \leq T$

$m = f(t, y)$

$y = y + \Delta t\, m$

$t = t + dt$

output t, y

Determine error at final time t:

error $= \mid$ exact$(t) - y \mid$

3.1.4 Numerical examples for the forward Euler method

We now consider examples involving the exponential and logistic growth/decay models discussed in § 2.1. We apply the forward Euler method for each problem and compute the global error at a fixed time; we expect that as Δt decreases the global error at that fixed time decreases linearly. To confirm that the numerical approximations are valid, for each pair of successive errors we use (1.6) to calculate the numerical rate of convergence and verify that it approaches one. For one example we see that the forward Euler method gives numerical approximations which oscillate and grow unexpectedly. It turns out that this is a result of numerical instability due to too large a time step. This is investigated in the next chapter. In the last example, we compare the local truncation error with the global error for the forward Euler method.

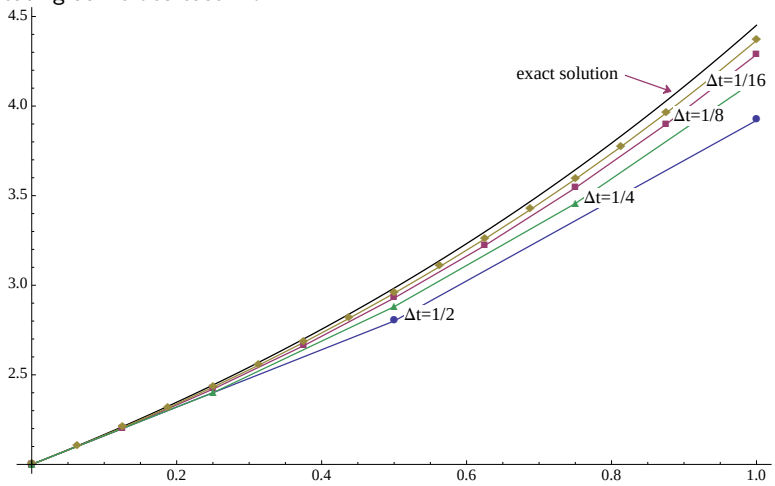
Example 3.4. Using the forward Euler method for exponential growth

Consider the exponential growth problem

$$p'(t) = 0.8p(t) \quad 0 < t \leq 1, \quad p(0) = 2$$

whose exact solution is $p(t) = 2e^{.8t}$. Compute approximations at $t = 1$ using a sequence of decreasing time steps for the forward Euler method and plot the exact solution the the approximations. Demonstrate that the numerical rate of convergence is linear by using the formula (1.6) and by using a graphical representation of the error. Does the time at which we compute the results affect the errors or the rates?

We compute approximations using the forward Euler method with $\Delta t = 1/2, 1/4, 1/8, 1/16$ and plot the exact solution along with the approximations. Remember that the approximate solution is a discrete function but we have connected the points for illustration purposes. These results are plotted in the figure below and we can easily see that the error is decreasing as we decrease Δt.

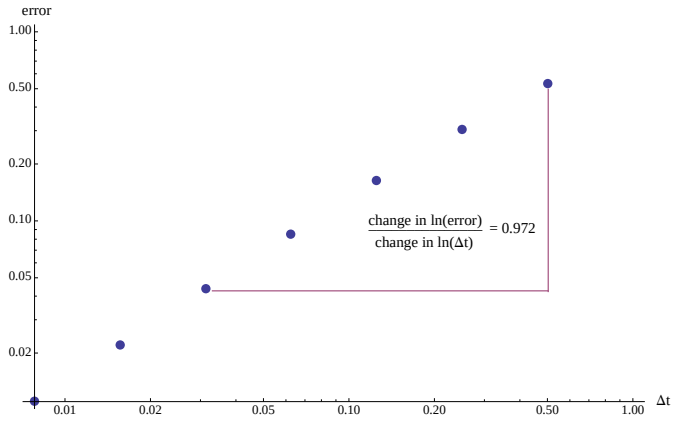


To verify that the global error is $\mathcal{O}(\Delta t)$ we compare the discrete solution to the exact solution at the point $t = 1$ where we know that the exact solution is $2e^{.8}$=4.45108; we tabulate our approximations P^n to $p(t)$ at $t = 1$ and the global error in the table below for $\Delta t = 1/4, 1/8, \ldots, 1/128$. By looking at the errors we see that as Δt is halved the error is approximately halved so this suggests linear convergence; the calculation of the numerical rate of convergence makes this result precise because we see that the sequence $\{.891, .942, .970, .985, .992\}$ tends to one. In the table the approximations and errors are given to five digits of accuracy.

Δt	$1/4$	$1/8$	$1/16$	$1/32$	$1/64$	$1/128$
P^n	4.1472	4.28718	4.36575	4.40751	4.42906	4.4400
$\lvert p(1) - P^n \rvert$	0.30388	0.16390	0.085333	0.043568	0.022017	0.011068
num. rate		0.891	0.942	0.970	0.985	0.992

We demonstrate graphically that the convergence rate is linear by using a log-log plot. Recall that if we plot a polynomial $y = ax^r$ on a log-log scale then the slope is r.[2] Since the error is $E = C(\Delta t)^r$, if we plot the error on a log-log plot we expect the slope to be r and in our case $r = 1$. This is illustrated in the log-log plot below where we compute the slope of the line for two points.

[2]Using the properties of logarithms we have $\log y = \log ax^r = \log a + r \log x$ which implies $Y = mX + b$ where $Y = \log y$, $X = \log x$, $m = r$ and $b = \log a$.



If we tabulate the errors at a different time then we get different errors but the numerical rate should still converge to one. In the table below we demonstrate this by computing the errors and rates at $t = 0.5$; note that the error is smaller at $t = 0.5$ than $t = 1$ for a given step size because we have not taken as many steps and we have less accumulated error.

Δt	$1/4$	$1/8$	$1/16$	$1/32$	$1/64$	$1/128$
P^n	2.8800	2.9282	2.9549	2.9690	2.9763	2.9799
$\lvert p(0.5) - P^n \rvert$	0.10365	0.055449	0.028739	0.014638	0.0073884	0.0037118
num. rate		0.902	0.948	0.973	0.986	0.993

The next example applies the forward Euler method to an IVP modeling logistic growth. The DE is nonlinear in this case but this does not affect the implementation of the algorithm. The results are compared to those from the previous example modeling exponential growth.

Example 3.5. Using the forward Euler method for logistic growth.

Consider the logistic model

$$p'(t) = 0.8\left(1 - \frac{p(t)}{100}\right)p(t) \quad 0 < t \leq 10 \quad p(0) = 2\,.$$

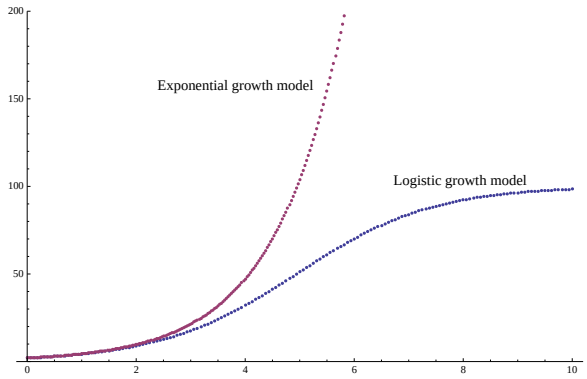
Implement the forward Euler scheme and demonstrate that we get linear convergence. Compare the results from this example with those from Example 3.4 of exponential growth.

The exact solution to this problem is given by (2.4) with $K = 100$, $r_0 = 0.8$, and $p_0 = 2$. Before generating any simulations we should think about what we expect the behavior of this solution to be compared with the exponential growth solution in the previous example. Initially the population should grow at the same rate because $r_0 = 0.8$ which is the same growth rate as in the previous example. However, the solution should not grow unbounded

but rather always stay below the carrying capacity $p = 100$. The approximations at $t = 1$ for a sequence of decreasing values of Δt are presented below along with the calculated numerical rates. The exact value at $t = 1$ is rounded to 4.3445923. Again we see that the numerical rate approaches one.

Δt	$1/4$	$1/8$	$1/16$	$1/32$	$1/64$	$1/128$
P^n	4.0740	4.1996	4.2694	4.3063	4.3253	4.3349
$\|p(1) - P^n\|$	0.27063	0.14497	0.075179	0.038302	0.019334	0.0097136
num. rate		0.901	0.947	0.973	0.993	0.993

Below we plot the approximate solution for $\Delta t = 1/16$ on $[0, 10]$ for this logistic growth problem and the approximate solution for the previous exponential growth problem. Note that the exponential growth solution increases without bound whereas the logistic growth solution never exceeds the carrying capacity of $K = 100$. Also for small time both models give similar results.



In the next example the IVP models exponential decay with a large decay constant. The example illustrates the fact that the forward Euler method can sometimes give erroneous results.

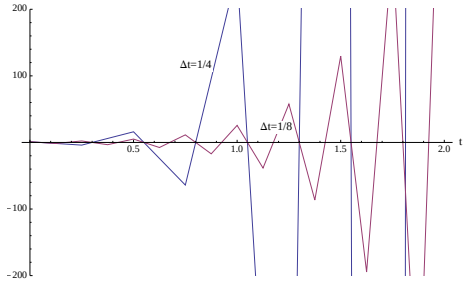
Example 3.6. **Numerically unstable computations for the forward Euler method.**

In this example we consider exponential decay where the decay rate is large. Specifically, we seek $y(t)$ such that

$$y'(t) = -20y(t) \quad 0 < t \leq 2, \quad y(0) = 1$$

which has an exact solution of $y(t) = e^{-20t}$. Plot the numerical results using $\Delta t = 1/4, 1/8$ and discuss results.

The implementation is the same as in Example 3.4 so we graph the approximate solutions on $[0, 2]$ with $\Delta t = \frac{1}{4}$ and $\frac{1}{8}$. Note that for this problem the approximate solution is oscillating and becoming unbounded.



Why are the results for the forward Euler method not reliable for this problem whereas they were for previous examples? The reason for this is a stability issue which we address in § 3.4. When we determined the theoretical rate of convergence we tacitly assumed that the method converged; however, from this example we see that it does not for these choices of the time step.

We know that the local truncation error for the forward Euler method is $\mathcal{O}(\Delta t)^2$. In the previous examples we demonstrate that the global error is $\mathcal{O}(\Delta t)$ so in the next example we demonstrate numerically that the local truncation error is second order.

Example 3.7. Comparing the local and global errors for the forward Euler method.

We consider the IVP

$$y'(t) = \cos(t)e^{\sin t} \quad 0 < t \leq \pi \qquad y(0) = 1$$

whose exact solution is $e^{\sin t}$. Demonstrate that the local truncation error for the forward Euler method is second order, i.e., $\mathcal{O}(\Delta t)^2$ and compare the local and global errors at $t = \pi$.

The local truncation error at t_n is computed from the formula

$$|y(t_n) - \widetilde{Y}^n| \quad \text{where } \widetilde{Y}^n = y(t_{n-1}) + \Delta t f\big(t_{n-1}, y(t_{n-1})\big)$$

that is, we use the exact value $y(t_{n-1})$ instead of Y^{n-1} and evaluate the slope at the point $(t_{n-1}, y(t_{n-1}))$ which is on the solution curve. In the table below we tabulate the local and global errors at $t = \pi$ using decreasing values of Δt. From the numerical rates of convergence you can clearly see that the local truncation error is $\mathcal{O}(\Delta t)^2$, as we demonstrated analytically. As expected, the global error converges linearly. Except at the first step (where the local and global errors are identical) the global error is always larger than the truncation error because it includes the accumulated errors as well as the error made by approximating the derivative by a difference quotient.

Δt	$1/8$	$1/16$	$1/32$	$1/64$	$1/128$
local error	$7.713\ 10^{-3}$	$1.947\ 10^{-3}$	$4.879\ 10^{-4}$	$1.220\ 10^{-4}$	$3.052\ 10^{-5}$
num. rate		1.99	2.00	2.00	2.00
global error	$2.835\ 10^{-2}$	$1.391\ 10^{-2}$	$6.854\ 10^{-3}$	$3.366\ 10^{-3}$	$1.681\ 10^{-3}$
num. rate		1.03	1.02	1.02	1.00

3.2 Backward Euler Method

In the previous section we derive the forward Euler method using the quotient $\big(y(t_{n+1}) - y(t_n)\big)/\Delta t$ to approximate the derivative at t_n. However, we can also use this difference quotient to approximate the derivative at t_{n+1}. This not only gives a different method but a completely new type of method. Using this quotient to approximate $y'(t_{n+1})$ gives

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

which implies

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

This equation suggests the following numerical scheme for approximating the solution to $y'(t) = f(t, y)$, $y(t_0) = y_0$ which we call the backward Euler method where $Y^0 = y_0$. It is called a "backward" scheme because we are at the point t_{n+1} and difference backwards to the point t_n to approximate the derivative $y'(t_{n+1})$.

Backward Euler: $$Y^{n+1} = Y^n + \Delta t f(t_{n+1}, Y^{n+1})\,, \quad n = 0, 1, 2, \ldots, N-1 \tag{3.9}$$

What makes this method so different from the forward Euler scheme? The answer is the point where the slope $f(t, y)$ is evaluated. To see the difficulty here consider the IVP $y'(t) = t + y^2$, $y(0) = 1$. Here $f(t, y) = t + y^2$. We set $Y^0 = 1$, $\Delta t = 0.1$ and to compute Y^1 we have the equation

$$Y^1 = Y^0 + \Delta t f(t_1, Y^1) = 1 + 0.1\Big(0.1 + \big(Y^1\big)^2\Big)$$

as opposed to the equation for Y^1 using forward Euler

$$Y^1 = Y^0 + \Delta t f(t_0, Y^0) = 1 + 0.1\Big(0 + (1)^2\Big)\,.$$

For forward Euler we evaluate the slope $f(t, y)$ at *known* values whereas for backward Euler we don't know the y-value where the slope is evaluated. In order to solve for Y^1 using the backward Euler scheme we must solve a nonlinear equation except when $f(t, y)$ is linear in y or only a function of t.

The difference between the forward and backward Euler schemes is so important that we use this characteristic to broadly classify methods. The forward Euler scheme given in (3.2) is called an explicit scheme because we write the unknown explicitly in terms of known values. The backward Euler method given in (3.9) is called an implicit scheme because the unknown is written implicitly in terms of known values and itself. The terms explicit/implicit are used in the same manner as

explicit/implicit differentiation. In explicit differentiation a function to be differentiated is given explicitly in terms of the independent variable such as $y(t) = t^3 + \cos t$; in implicit differentiation the function is given implicitly such as $y^2 + \sin y - t^2 = 4$ and we want to compute $y'(t)$. In the exercises you get practice in identifying schemes as explicit or implicit.

The local truncation error for the backward Euler method is $\mathcal{O}(\Delta t)^2$, the same as we obtained for the forward Euler method (see exercises). Thus we expect the global error to be one power less, i.e., $\mathcal{O}(\Delta t)$. The numerical examples in the next section confirm this estimate.

3.2.1 Numerical examples for the backward Euler method

In general, the backward Euler method for a single IVP requires solution of a nonlinear equation; for a system of IVPs it requires solution of a nonlinear system of equations. For this reason, the backward Euler method is not used in a straightforward manner for solving IVPs. However, when we study IBVPs we see that it is a very popular scheme for discretizing a first order time derivative. Also in § 4.4 we see how it is implemented in an efficient manner for IVPs using an approach called predictor-corrector schemes. We discuss the implementation of the backward Euler method in the following examples. In the simple case when $f(t, y)$ is only a function of t or it is linear in y we do not need to solve a nonlinear equation.

Example 3.8. Using the backward Euler method for exponential growth

Consider the exponential growth problem

$$p'(t) = 0.8p(t) \quad 0 < t \leq 1, \quad p(0) = 2$$

whose exact solution is $p(t) = 2e^{.8t}$. In Example 3.4 we apply the forward Euler method to obtain approximations at $t = 1$ using a sequence of decreasing time steps. Repeat the calculations for the backward Euler method. Discuss implementation.

To solve this IVP using the backward Euler method we see that for $f = 0.8p$ the difference equation is linear

$$P^{n+1} = P^n + 0.8\Delta t P^{n+1},$$

where $P^n \approx p(t_n)$. Thus we do not need to use Newton's method for this particular problem but rather just solve the equation

$$P^{n+1} - 0.8\Delta t P^{n+1} = P^n \quad \Rightarrow \quad P^{n+1} = \frac{1}{1 - 0.8\Delta t} P^n.$$

If we have a code that uses Newton's method it should get the same answer in one step because it is solving a linear problem rather than a nonlinear one. The results are tabulated below. Note that the numerical rate of convergence is also approaching one but for this method it is approaching one from above whereas using the forward Euler scheme for this problem the convergence was from below, i.e., through values smaller than one. The amount of work required for the backward Euler method is essentially the same as the forward Euler for this problem because the derivative $f(t, p)$ is linear in the unknown p.

Δt	$1/2$	$1/4$	$1/8$	$1/16$	$1/32$	$1/64$	$1/128$
P^n	5.5556	4.8828	4.6461	4.5441	4.4966	4.4736	4.4623
$\lvert p(1) - P^n \rvert$	1.1045	0.43173	0.19503	0.093065	0.045498	0.022499	0.011188
num. rate		1.355	1.146	1.067	1.032	1.015	1.008

In the previous example, the DE was linear in the unknown so it was straightforward to implement the backward Euler scheme and it took the same amount of work as implementing the forward Euler scheme. However, the DE modeling logistic growth is nonlinear and so implementation of the backward Euler scheme involves solving a nonlinear equation. This is addressed in the next example.

Example 3.9. Using the backward Euler method for logistic growth.

In Example 3.5 we consider the logistic model

$$p'(t) = 0.8\left(1 - \frac{p(t)}{100}\right)p(t) \quad 0 < t \leq 10 \quad p(0) = 2\,.$$

Implement the backward Euler scheme and demonstrate that we get linear convergence. Discuss implementation of the implicit method.

To implement the backward Euler scheme for this problem we see that at each step we have the nonlinear equation

$$P^{n+1} = P^n + \Delta t f(t_{n+1}, P^{n+1}) = P^n + .8\Delta t\left(P^{n+1} - \frac{\left(P^{n+1}\right)^2}{100}\right)$$

for P^{n+1}. Thus to determine each P^{n+1} we have to employ a method such as Newton's method. Recall that to find the root z of the nonlinear equation $g(z) = 0$ (a function of one independent variable) each iteration of Newton's method is given by

$$z^k = z^{k-1} - \frac{g(z^{k-1})}{g'(z^{k-1})}$$

for the iteration counter $k = 1, 2, \ldots$ and where an initial guess z^0 is prescribed. For our problem, to compute the solution at t_{n+1} we have the nonlinear equation

$$g(z) = z - P^n - .8\Delta t\left(z - \frac{z^2}{100}\right) = 0$$

where $z = P^{n+1}$. Our goal is to approximate the value of z which makes $g(z) = 0$ and this is our approximation P^{n+1}. For an initial guess z^0 we simply take P^n because if Δt is small enough and the solution is smooth then the approximation at t_{n+1} is close to the solution at t_n. To implement Newton's method we also need the derivative g' which for us is just

$$g'(z) = 1 - .8\Delta t\left(1 - \frac{z}{50}\right)\,.$$

The results using backward Euler are tabulated below; note that the numerical rates of convergence approach one as $\Delta t \to 0$. We have imposed the convergence criteria

for Newton's method that the normalized difference in successive iterates is less than a prescribed tolerance, i.e.,

$$\frac{|z^k - z^{k-1}|}{|z^k|} \leq 10^{-8}\,.$$

Δt	1/4	1/8	1/16	1/32	1/64	1/128
P_n	4.714	4.514	4.426	4.384	4.364	4.354
$\lvert p(1) - P_n \rvert$	0.3699	0.1693	0.08123	0.03981	0.01971	0.009808
num. rate		1.127	1.060	1.029	1.014	1.007

In these computations, two to four Newton iterations are necessary to satisfy this convergence criteria. It is well known that Newton's method typically converges quadratically (when it converges) so we should demonstrate this. To this end, we look at the normalized difference in successive iterates. For example, for $\Delta t = 1/4$ at $t = 1$ we have the sequence 0.381966, $4.8198\ 10^{-3}$, $7.60327\ 10^{-7}$, $1.9105\ 10^{-15}$ so that the difference at one iteration is approximately the square of the difference at the previous iteration indicating quadratic convergence.

Lastly we repeat the calculations for the exponential decay IVP in Example 3.6 using the backward Euler method. Recall that the forward Euler method produced oscillatory results for $\Delta t = 1/4$ and $\Delta t = 1/8$. This example illustrates that the backward Euler method does not produce unstable results for this problem.

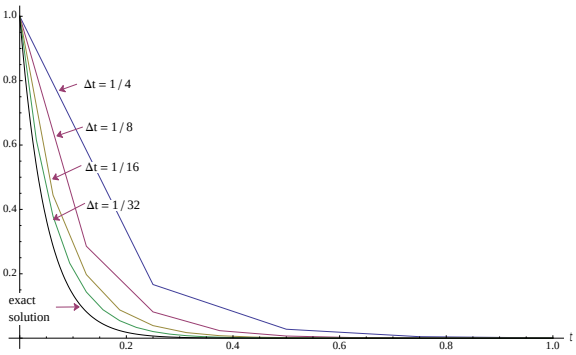
Example 3.10. Backward Euler stable where forward Euler method oscillates.

We consider the problem in Example 3.6 for exponential decay where the forward Euler method gives unreliable results. Specifically, we seek $y(t)$ such that

$$y'(t) = -20y(t) \quad 0 < t \leq 2, \quad y(0) = 1$$

which has an exact solution of $y(t) = e^{-20t}$. Plot the results using the backward Euler method for $\Delta t = 1/2, 1/4, \ldots, 1/16$. Compare results with those from the forward Euler method.

Because this is just an exponential decay problem, the implementation is analogous to that in Example 3.8. We graph approximations using the backward Euler method along with the exact solution. As can be seen from the plot, it appears that the discrete solution is approaching the exact solution as $\Delta t \to 0$ whereas the forward Euler method gave unreliable results for $\Delta t = 1/2, 1/4$. Recall that the backward Euler method is an implicit scheme whereas the forward Euler method is an explicit scheme.



3.3 Improving on the Euler Methods

Although the forward Euler method is easy to understand and to implement, its rate of convergence is only linear so it converges very slowly. The backward Euler is more difficult to implement and still converges only linearly. In the next chapter we look at many methods which improve on the accuracy of the Euler methods, but we introduce two methods here to give a preview.

When we derive the forward Euler method we use the slope of the secant line joining $(t_n, y(t_n))$ and $(t_{n+1}, y(t_{n+1}))$ to approximate the derivative at t_n, i.e., $y'(t_n)$ and for the backward Euler we use this slope to approximate $y'(t_{n+1})$. In the same manner, we can use this slope to approximate the derivative at any point in $[t_n, t_{n+1}]$ so an obvious choice would be the midpoint, i.e.,

$$y'\left(\frac{t_n + t_{n+1}}{2}\right) \approx \frac{y(t_n) + y(t_{n+1})}{\Delta t}\,.$$

As before, we use the differential equation $y'(t) = f(t, y)$ and now evaluate at the midpoint to get

$$\frac{y(t_n) + y(t_{n+1})}{\Delta t} \approx f\left(\frac{t_n + t_{n+1}}{2}, y\Big(\frac{t_n + t_{n+1}}{2}\Big)\right)$$

which implies

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

Now the problem with using this expression to generate a scheme is that we do not know y at the midpoint of the interval $[t_n, t_{n+1}]$. So what can we do? The only option is to approximate it there so an obvious approach is to take a step of length $\Delta t/2$ using forward Euler, i.e.,

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

Using this approximation leads to the difference scheme

$$Y^{n+1} = Y^n + \Delta t f\Big(t_n + \frac{\Delta t}{2}, Y^n + \frac{\Delta t}{2} f(t_n, Y^n)\Big)\,. \tag{3.10}$$

This scheme is often called the improved Euler method or the midpoint method and is clearly explicit because we are evaluating the slope at known points. When we look at the scheme we see that we actually have to perform two function evaluations whereas for the forward Euler we only had to evaluate $f(t_n, Y^n)$. Since we are doing more work we should expect better results and, in fact, this method is second order accurate which we demonstrate in the next chapter. The midpoint rule belongs to a family of methods called Runge-Kutta methods which we explore in Chapter 4.

An implicit scheme which converges quadratically, as opposed to the linear convergence of the backward Euler method, is the so-called modified Euler method or the trapezoidal method. This method is found by approximating f at the midpoint of (t_n, t_{n+1}) by averaging its value at (t_n, Y^n) and (t_{n+1}, Y^{n+1}); it is defined by

$$Y^{n+1} = Y^n + \frac{\Delta t}{2}\left[f(t_n, Y^n) + f(t_{n+1}, Y^{n+1})\right]\,. \tag{3.11}$$

This method is clearly implicit because it requires evaluating the slope at the unknown point (t_{n+1}, Y^{n+1}). In the next chapter we look at approaches to systematically derive new methods rather than heuristic approaches.

3.4 Consistency, stability and convergence

From Example 3.6 we see that the forward Euler method fails to provide reliable results for some values of Δt in an exponential decay problem but in Theorem 3.1 we prove that the global error satisfies $|y(t_n) - Y^n| \leq C\Delta t$ at each point t_n. At first glance, the numerical results seem to be at odds with our theorem. However, with closer inspection we realize that to prove the theorem we tacitly assumed that Y^n is the *exact* solution of the difference equation. This is not the case because when we implement methods small errors are introduced due to round off. We want to make sure that the solution of the difference equation that we compute remains close to the one we would get if exact arithmetic is used. The forward Euler method in Example 3.6 exhibits numerical instability because roundoff error accumulates so that the computed solution of the difference equation is very different from its exact solution; this results in the solution oscillating and becoming unbounded. However, recall that the backward Euler method gives reasonable results for this same problem so the roundoff errors here do not unduly contaminate the computed solution to the difference equation.

Why does one numerical method produce reasonable results and the other meaningless results even though their global error is theoretically the same? The answer lies in the stability properties of the numerical scheme. In this section we formally define convergence and its connection with the concepts of consistency and stability. When we consider families of methods in the next chapter we learn techniques to determine their stability properties.

Any numerical scheme we use must be consistent with the differential equation we are approximating. The discrete solution satisfies the difference equation but the exact solution $y(t_n)$ yields a residual when substituted into the difference equation which we call the local truncation error. As before we define $\tau(\Delta t)$ to be the largest (in absolute value) local truncation error made at each of the N time steps with increment Δt , i.e., $\tau(\Delta t) = \max_{1\leq n\leq N} |\tau_n(\Delta t)|$. For the scheme to be consistent, this error should go to zero as $\Delta t \to 0$, i.e.,

$$\lim_{\Delta t\to 0} \tau(\Delta t) = 0\,. \tag{3.12}$$

Both the forward and backward Euler methods are consistent with (2.8) because we know that the maximum local truncation error is $\mathcal{O}(\Delta t)^2$ for all t_n. If the local truncation error is constant then the method is not consistent. Clearly we only want to use difference schemes which are consistent with our IVP (2.8). However, the consistency requirement is a local one and does not guarantee that the method is convergent as we saw in Example 3.6.

We now want to determine how to make a consistent scheme convergent. Intuitively we know that for a scheme to be convergent the discrete solution at each point must get closer to the exact solution as the step size reduces. As with consistency, we can write this formally and say that a method is convergent if

$$\lim_{\Delta t\to 0} \max_{1\leq n\leq N} |y(t_n) - Y^n| = 0\,.$$

To interpret this mathematical expression consider performing calculations at $\Delta t = 1/4$ first. Then we take the maximum global error at each time step. Now we reduce Δt to $1/8$ and repeat the process. Hopefully the maximum global error using $\Delta t = 1/8$ is smaller than the one when $\Delta t = 1/4$. We keep reducing Δt and this sequence of maximum global errors should approach zero for the method to converge.

The reason the consistency requirement is not sufficient for convergence is that it requires the *exact* solution to the difference equation to be close to the exact solution of the differential equation. It does not take into account the fact that we are not computing the exact solution of the difference equation due to roundoff errors. It turns out that the additional condition that is needed is stability which requires the difference in the computed solution to the difference equation and its exact solution to be small. This requirement combined with consistency gives convergence.

Numerical Convergence Requirements

Consistency + Stability = Convergence

To investigate the effects of the (incorrect) computed solution to the difference equation, we let $\widetilde{Y^n}$ represent the computed solution to the difference equation

which has an actual solution of Y^n at time t_n. We want the difference between $y(t_n)$ and $\widetilde{Y^n}$ to be small. At a specific t_n we have

$$|y(t_n) - \widetilde{Y^n}| = |y(t_n) - Y^n + Y^n - \widetilde{Y^n}| \leq |y(t_n) - Y^n| + |Y^n - \widetilde{Y^n}|,$$

where we have used the triangle inequality. Now the first term $|y(t_n) - Y^n|$ is governed by making the local truncation error sufficiently small (i.e., making the equation consistent) and the second term is controlled by the stability requirement. So if each of these two terms can be made sufficiently small then when we take the maximum over all points t_n and take the limit as Δt approaches zero we get convergence.

In the next chapter we investigate the stability of methods and demonstrate that the forward Euler method is only stable for Δt sufficiently small whereas the backward Euler method is numerically stable for all values of Δt. Recall that the forward Euler method is explicit whereas the backward Euler is implicit. This pattern follows for other methods; that is, explicit methods have a stability requirement whereas implicit methods do not. Of course this doesn't mean we can take a very large time step when using implicit methods because we still have to balance the accuracy of our results.

EXERCISES

3.1. Classify each difference equation as explicit or implicit. Justify your answer.

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

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

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

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

3.2. Assume that the following set of errors isobtained from three different methods for approximating the solution of an IVP of the form (2.8) at a specific time. First look at the errors and try to decide the accuracy of the method. Then use the result (1.6) to determine a sequence of approximate numerical rates for each method using successive pairs of errors. Use these results to state whether the accuracy of the method is linear, quadratic, cubic or quartic.

Δt	Errors Method I	Errors Method II	Errors Method III
$1/4$	0.23426×10^{-2}	0.27688	0.71889×10^{-5}
$1/8$	0.64406×10^{-3}	0.15249	0.49840×10^{-6}
$1/16$	0.16883×10^{-3}	0.80353×10^{-1}	0.32812×10^{-7}
$1/32$	0.43215×10^{-4}	0.41292×10^{-1}	0.21048×10^{-8}

3.3. Suppose the solution to the DE $y'(t) = f(t, y)$ is a concave up function on $[0, T]$. Will the forward Euler method give an underestimate or an overestimate to the solution? Why?

3.4. Show that if we integrate the IVP (2.8a) from t_n to t_{n+1} and use a right Riemann sum to approximate the integral of $f(t, y)$ then we obtain the backward Euler method.

3.5. Consider the IVP $y'(t) = -\lambda y(t)$ with $y(0) = 1$. Apply the backward Euler method to this problem and show that we have a closed form formula for Y^n, i.e.,

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

3.6. Derive the backward Euler method by using the Taylor series expansion for $y(t_n - \Delta t)$.

3.7. Consider approximating the solution to the IVP

$$y'(t) = 1 - y \quad y(0) = 0$$

using the backward Euler method. In this case the given slope $y'(t)$ is linear in y so the resulting difference equation is linear. Use backward Euler to approximate the solution at $t = 0.5$ with $\Delta t = 1/2, 1/4 \ldots, 1/32$. Compute the error in each case and the numerical rate of convergence.

3.8. What IVP has a solution which exhibits the logarithmic growth $y(t) = 2+3\ln t$ where the initial time is prescribed at $t = 1$?

3.9. Determine an approximation to the solution at $t = 0.5$ to the IVP

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

using the forward Euler method with $\Delta t = 1/4$. Compute the local and global errors at $t = 1/4$ and $t = 1/2$. The exact solution is $y(t) = (e^{2t} - 1)/(e^{2x} + 1)$.

Computer Exercises

3.10. Consider the exponential growth problem

$$p'(t) = 4p(t) \quad 0 < t \leq 1, \quad p(0) = 2$$

whose exact solution is $p(t) = 2e^{4t}$.

This problem has a growth rate which is five times that of the rate in Example 3.4 so we expect it to growth much faster. Tabulate the errors for $\Delta t = 1/2, 1/4, \ldots, 1/128$ and compare with those from the previous example. Compare the numerical errors as well as the actual magnitude of the errors. Why do you think that this problem has larger errors.

3.11. Consider the IVP

$$y'(t) = \cos^2(t)\cos^2(2y) \quad \frac{\pi}{2} < t \leq \pi, \quad y(\frac{\pi}{2}) = \pi\,.$$

This nonlinear differential equation is separable and the IVP has the exact solution $y(t) = \frac{1}{2}\left[\arctan\left(t + \frac{1}{2}\sin(2t) - \frac{\pi}{2}\right) + 2\pi\right]$. Compute the solution using forward and backward Euler methods and demonstrate that the convergence is linear. For the backward Euler method incorporate Newton's method and verify that it is converging quadratically. For each method compute and tabulate the numerical rates using successive values of $N = 10, 20, 40, \ldots, 320$. Discuss your results and compare with theory.

3.12. Write a code which implements the forward Euler method to solve an IVP of the form (2.8). Use your code to approximate the solution of the IVP

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

which has an exact solution $y(t) = (e^{2t} - 1)/(e^{2x} + 1)$. Compute the errors at $t = 1$ using $\Delta t = 1/4, 1/8, 1/16, 1/32, 1/64$.

a. Tabulate the *global error* at $t = 1$ for each value of Δt and demonstrate that your method converges with accuracy $\mathcal{O}(\Delta t)$; justify your answer by calculating the numerical rate of convergence for successive pairs of errors.

b. Tabulate the *local error* at $t = 1$ for each value of Δt and determine the rate of convergence of the local error; justify your answer by calculating the numerical rate of convergence for successive pairs of errors. Compare your results with those obtained in (a).

3.13. Suppose you are interested in modeling the growth of the Bread Mold Fungus, *Rhizopus stolonifer* and comparing your numerical results to experimental data that is taken by measuring the number of square inches of mold on a slice of bread over a period of several days. Assume that the slice of bread is a square of side 5 inches.

a. To obtain a model describing the growth of the mold you first make the hypothesis that the growth rate of the fungus is proportional to the amount of mold present at any time with a proportionality constant of k. Assume that the initial amount of mold present is 0.25 square inches. Let $p(t)$ denote the number of square inches of mold present on day t. Write an initial value problem for the growth of the mold.

b. Assume that the following data is collected over a period of ten days. Assuming that k is a constant, use the data at day one to determine k. Then using the forward Euler method with Δt a fourth and an eight of a day, obtain numerical estimates for each day of the ten day period; tabulate your results and compare with the experimental data. When do the results become physically unreasonable?

$t = 0$	p =0.25	$t = 1$	p =0.55
$t = 2$	p =1.1	$t = 3$	p =2.25
$t = 5$	p =7.5	$t = 7$	p =16.25
$t = 8$	p =19.5	$t = 10$	p =22.75

c. The difficulty with the exponential growth model is that the bread model grows in an unbounded way as you saw in (b). To improve the model for the growth of bread mold, we want to incorporate the fact that the number of square inches of mold can't exceed the number of square inches in a slice of bread. Write a logistic differential equation which models this growth using the same initial condition and growth rate as before.

d. Use the forward Euler method with Δt a fourth and an eighth of a day to obtain numerical estimates for the amount of mold present on each of the ten days using your logistic model. Tabulate your results as in (b) and compare your results to those from the exponential growth model.

Chapter 4

A Survey of Methods

In the last chapter we developed the forward and backward Euler method for approximating the solution to the first order IVP (2.8). 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 3.6 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 (2.8) 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 (i.e., 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. 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 are 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 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 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 § 4.1.3 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 are 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 § 4.2.1 and the widely used Adams-Bashforth and Adams-Moulton families of multistep methods are presented in § 4.2.2. Because multistep methods rely on previously computed values the stability analysis is more complicated than for single-step methods so we 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 § 4.3 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 discuss how the currently popular Burlisch-Stoer extrapolation method exploits certain properties of methods to provide a robust algorithm with high accuracy.

In general, the backward Euler method, which is implicit, is more costly to implement than the forward Euler method, which is explicit, due to the fact that it typically requires the solution of a nonlinear equation at each time step. In § 4.4 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.

4.1 Single-Step Methods

In this section we look at the important class of numerical methods for the IVP (2.8) 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. Implicit single-step methods use the additional point t_{n+1}.

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 are 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 § 4.1.3.

4.1.1 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

$$y(t_n + \Delta t) = y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2!} y''(t_n) + \cdots + \frac{(\Delta t)^k}{k!} y^{[k]}(t_n) + \cdots . \quad (4.1)$$

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 after 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., $y'(t_n) = f\big(t_n, y(t_n)\big)$.

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 (4.1) 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 generates the second order explicit Taylor series method.

Second order explicit Taylor series method

$$Y^{n+1} = Y^n + \Delta t f(t_n, Y^n) + \frac{(\Delta t)^2}{2}\Big[f_t(t_n, Y^n) + f(t_n, Y^n) f_y(t_n, Y^n)\Big] \quad (4.2)$$

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 is 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 4.1. SECOND ORDER TAYLOR METHOD

Approximate the solution to

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

using (4.2) by hand using $\Delta t = 0.1$ and $T = 0.2$. Then implement the method and obtain approximations for $\Delta t = 1/4, 1/8, \cdots, 1/128$. 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 (4.2) gives the expression

$$Y^{n+1} = Y^n + 3\Delta t Y^n (t_n)^2 + \frac{(\Delta t)^2}{2}\left(6t_n Y^n + 9(t_n)^4 Y^n\right)\,. \quad (4.3)$$

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 (4.2). 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 (4.2), not the equation (4.3) 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 are 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

$$y(t_n) = y(t_{n+1} - \Delta t) = y(t_{n+1}) - \Delta t y'(t_{n+1}) + \frac{(\Delta t)^2}{2!}y''(t_{n+1}) + \cdots\,. \quad (4.4)$$

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.

4.1.2 Derivation of methods which don't require repeated differentiation of the slope

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 described in this section.

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 (2.8a) from t_n to t_{n+1} we have

$$\int_{t_n}^{t_{n+1}} y'(t)\,dt = \int_{t_n}^{t_{n+1}} f(t,y)\,dt\,. \tag{4.5}$$

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 we 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

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

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

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

Using the midpoint quadrature rule to approximate the integral of $f(t, y)$ in (4.5) gives

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

We encountered this scheme in § 3.3. Recall that we don't know y evaluated at the midpoint so we must 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

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + \tfrac{\Delta t}{2}, Y^n + \tfrac{1}{2}k_1) \\ Y^{n+1} &= Y^n + k_2 \end{aligned} \tag{4.6}$$

The midpoint rule has a simple geometrical interpretation. Recall that for the forward Euler method we used a tangent line at t_n to extrapolate the solution at t_{n+1}. In the midpoint rule we use a 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 4.2. 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

$$\tau_{n+1} = y(t_{n+1}) - \left[y(t_n) + \Delta t f\Big(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f\big(t_n, y(t_n)\big)\Big)\right]. \tag{4.7}$$

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

$$y(t_{n+1}) = y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2} y''(t_n) + \frac{(\Delta t)^3}{3!} y'''(t_n) + \mathcal{O}(\Delta t)^4. \tag{4.8}$$

Substituting this into (4.7) yields

$$\begin{aligned} \tau_{n+1} \quad = \quad & \left[y(t_n) + \Delta t y'(t_n) + \frac{(\Delta t)^2}{2} y''(t_n) + \frac{(\Delta t)^3}{3!} y'''(t_n) + \mathcal{O}(\Delta t)^4 \right] \\ & - \left[y(t_n) + \Delta t f\Big(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f\big(t_n, y(t_n)\big)\Big)\right]. \end{aligned} \tag{4.9}$$

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 (see Appendix)

$$\begin{aligned} g(x+h, y+k) = {} & g(x,y) + h g_x(x,y) + k g_y(x,y) \\ & + \frac{1}{2!}\left[h^2 g_{xx}(x,y) + k^2 g_{yy}(x,y) + 2kh g_{xy}(x,y)\right] \\ & + \frac{1}{3!}\left[h^3 g_{xxx}(x,y) + k^3 g_{yyy}(x,y) + 3k^2 h g_{xyy}(x,y) + 3h^2 k g_{xxy}(x,y)\right] \\ & + \cdots. \end{aligned} \tag{4.10}$$

To use this formula to expand $f(t_n + \frac{\Delta t}{2}, y(t_n) + \frac{\Delta t}{2} f(t_n, y(t_n)))$, 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 $\big(t_n, y(t_n)\big)$ and we have omitted this explicit dependence for brevity. Substituting this expansion into the expression (4.9) for τ_{n+1} and collecting terms involving each power of Δt yields

$$\begin{aligned}\tau_{n+1} &= \Delta t\Big[y' - f\Big] + \Delta t^2\Big[\frac{1}{2}y'' - \frac{1}{2}(f_t + ff_y)\Big] \\ &\quad + \Delta t^3\big(\frac{1}{3!}y''' - \frac{1}{8}\Big[f_{tt} + f^2 f_{yy} + 2ff_{ty}\Big] + \mathcal{O}(\Delta t)^4 .\end{aligned} \tag{4.11}$$

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,

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

Thus the terms in (4.11) 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 $\int_a^b g(x)\ dx$ where $g(x) \geq 0$ 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

$$\int_a^b g(x)\ dx \approx \frac{(b-a)}{2}\Big(g(a) + g(b)\Big) .$$

Integrating the differential equation (2.8a) from t_n to t_{n+1} and using this quadrature rule gives

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

We encountered this implicit scheme in § 3.3.

Trapezoidal Rule $$Y^{n+1} = Y^n + \frac{\Delta t}{2}\big[f(t_n, Y^n) + f(t_{n+1}, Y^{n+1})\big] \tag{4.12}$$

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 see in § 4.4.

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 § 4.1.3.

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 4.3. 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; we denote the general point in $(t_n, t_{n+1}]$ as $t_n + c_2\Delta t$ and the general approximation to y at this point ias $Y^n + a_{21}\Delta t f(t_n, Y^n)$. The general difference equation using two function evaluations is

$$Y^{n+1} = Y^n + b_1\Delta t f(t_n, Y^n) + b_2\Delta t f\big(t_n + c_2\Delta t, Y^n + a_{21}\Delta t f(t_n, Y^n)\big)\,. \qquad (4.13)$$

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 (4.10) for a Taylor series expansion in two variables to get

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

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

$$\begin{aligned} \Delta t\,[f(1-b_1-b_2)] &= 0 \\ \Delta t^2\left[f_t(\frac{1}{2}-b_2c_2)+ff_y(\frac{1}{2}-b_2a_{21})\right] &= 0\,, \end{aligned}$$

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

$$b_1+b_2=1, \quad b_2c_2=\frac{1}{2} \quad \text{and} \quad b_2a_{21}=\frac{1}{2}\,. \tag{4.14}$$

Note that the midpoint method given in (4.6) 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_tf_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 remain. Consequently there is no third order explicit one-step method which only performs two function evaluations per time step.

4.1.3 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 backward 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 § 4.1.2 because it gives families of methods with a prescribed local truncation error. This is illustrated in Example 4.3 where we assumed the most general form of an explicit two-stage single step method in (4.13). To illustrate the fact that it is a two-stage method we can also write (4.13) as

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n+c_2\Delta t, Y^n+a_{21}k_1) \\ Y^{n+1} &= Y^n+b_1k_1+b_2k_2\,. \end{aligned}$$

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

We obtain 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 a two-stage RK method. Another commonly used two-stage RK method is the Heun method. Here the intermediate time is $t_n + \frac{2}{3}\Delta t$, so $c_2 = 2/3$. The equation $b_2 c_2 = 1/2$ requires $b_2 = 3/4$ and the condition $b_1 + b_2 = 1$ requires $b_1 = 1/4$ and $b_2 a_{21} = 1/2$ implies $a_{21} = 2/3$. Thus 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 $\frac{2}{3}\Delta t$.

Heun Method

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + \tfrac{2}{3}\Delta t, Y^n + \tfrac{2}{3}k_1) \\ Y^{n+1} &= Y^n + \tfrac{1}{4}k_1 + \tfrac{3}{4}k_2 \end{aligned} \tag{4.15}$$

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

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n) \\ k_2 &= \Delta t f(t_n + c_2\Delta t, Y^n + a_{21}k_1) \\ k_3 &= \Delta t f(t_n + c_3\Delta t, Y^n + a_{31}k_1 + a_{32}k_2) \\ &\vdots \\ k_s &= \Delta t f\big(t_n + c_s\Delta t, Y^n + a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{ss-1}k_{s-1}\big) \\ Y^{n+1} &= Y^n + \textstyle\sum_{j=1}^{s} b_j k_j \end{aligned} \tag{4.16}$$

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

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{4.17}$$

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

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

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{4.18}$$

which uses an approximation at the point t_n, two approximations at the midpoint $t_n + \Delta t/2$, 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.

NUMERICAL IMPLEMENTATION OF RUNGE-KUTTA METHODS

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 to write a routine where the user chooses the method based upon the stage of the desired RK method. 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. A driver routine is used to call both 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 a 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. Note that storing the coefficients as arrays allows us to use a dot product instead of loops.

Algorithm 4.1 : Advancing explicit RK method one time step

Assume: Coefficients in RK method are stored as follows: c_i and b_i, $i = 1, \ldots, s$ are stored in 1D array and coefficients a_{ij}, $i, j = 1, \ldots, s$, in a 2D array

Input: the solution y at the time t, the uniform time step Δt, s the number of stages, coefficients a, b, c.

Loop over number of stages:

$k = 0$

for $i = 1, \ldots, s$

$t_{\text{eval}} = t + c(i)\Delta t$

$y_{\text{eval}} = y + \text{dot\_product}\big(a(i,\cdot), k\big)$

$k(i) = \Delta t f(t_{\text{eval}}, y_{\text{eval}})$

$y = y + \text{dot\_product}(k, b)$

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 4.4. 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 are 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 4.4 we saw that for $s \le 4$ 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.

Maximum accuracy of s-stage explicit RK methods

Stage	1	2	3	4	5	6	7	8	9	10
Order	1	2	3	4	4	5	6	6	7	7

Analogous to the general explicit s-stage RK scheme (4.16) 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

$$\begin{aligned} k_1 &= \Delta t f(t_n, Y^n + a_{11}k_1 + a_{12}k_2 + \cdots a_{1s}k_s) \\ k_2 &= \Delta t f(t_n + c_2\Delta t, Y^n + a_{21}k_1 + +a_{22}k_2 + \cdots a_{2s}k_s) \\ &\vdots \\ k_s &= \Delta t f(t_n + c_s\Delta t, Y^n + a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{ss}k_s) \\ Y^{n+1} &= Y^n + \sum_{j=1}^{s} b_j k_j \end{aligned} \tag{4.19}$$

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

$$\begin{array}{c|cccc} 0 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{4.20}$$

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

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

4.1.4 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

$$y'(t) = \lambda y \quad 0 < t \leq T, \lambda \in \mathbb{C}, \quad y(0) = y_0 \tag{4.21}$$

with the solution $y(t) = y_0 e^{\lambda t}$; here $\mathbb{C}$ represents all complex numbers of the form $\alpha + i\beta$. 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 (4.21) 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	$\delta = \lvert\beta - \alpha\rvert$	$\lvert y_1(0.5) - y_2(0.5)\rvert$	$\lvert y_1(1) - y_2(1)\rvert$	$\lvert y_1(10) - y_2(10)\rvert$
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 $e^{\alpha t}$ grows 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 (4.21) we want to know that small changes, such as those due to roundoff, do not cause large changes in the solution. Here we are going to look at stability of a difference equation of the form

$$Y^{n+1} = \zeta(\lambda \Delta t) Y^n \tag{4.22}$$

applied to the model problem (4.21). We apply the difference equation (4.22) 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. Our single step methods fit into this framework as the following example illustrates.

Example 4.5. AMPLIFICATION FACTORS

Determine the amplification factors for the forward and backward Euler methods.

The forward Euler method applied to the differential equation $y' = \lambda y$ gives

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

so $\zeta(\lambda \Delta t) = 1 + \Delta t \lambda$.

For the backward Euler method we have

$$Y^{n+1} = Y^n + \Delta t \lambda Y^{n+1} \Rightarrow (1 - \Delta t \lambda) Y^{n+1} = Y^n$$

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 know that the magnitude of ζ must be less than or equal to one or else Y^n becomes 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.

Absolute Stability

The region of absolute stability for the difference equation (4.22) 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.

In the next example we determine the region of absolute stability of the forward Euler method and compare to the results in Example 3.6.

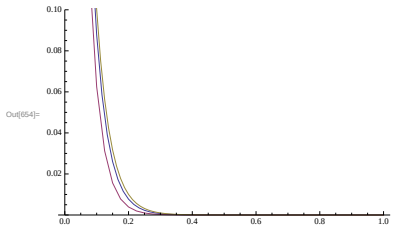
Example 4.6. Determine if the forward Euler method and the backward Euler method are A-stable; if not, determine the region of absolute stability. Then discuss the previous numerical results for $y'(t) = -20y(t)$ in light of these results.

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 3.6 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 $\lambda < 0$. This agrees with the results from Example **??**.

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.

Next we show that the explicit Heun method has the same region of stability as the forward Euler method so we expect the same behavior. Recall that the Heun method is second order accurate whereas the Euler method is first order so accuracy has nothing to do with stability.

Example 4.7. Investigate the region of absolute stability for the explicit 2-stage Heun method given in (4.15). Plot results for this method applied to the IVP from Example **??**. Choose values of the time step where the stability criteria is not met and then some values where it is satisfied.

We first write 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.

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

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

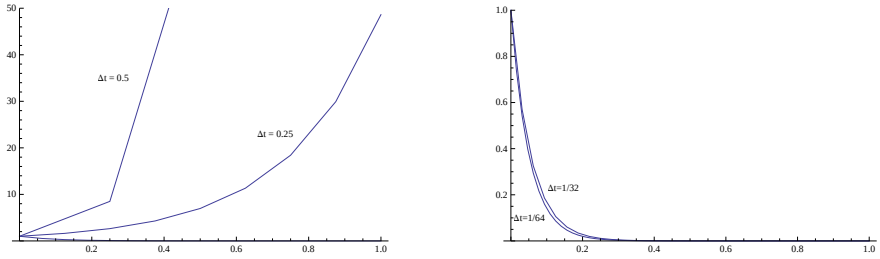
$$Y^{n+1} = Y^n + \frac{\Delta t}{4}\left[\lambda Y^n + 3\lambda(Y^n + \frac{2}{3}\Delta t\lambda Y^n)\right] = \left[1 + \frac{1}{4}(\lambda\Delta t) + \frac{3}{4}(\lambda\Delta t) + \frac{1}{2}(\lambda\Delta t)^2\right]Y^n$$

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.

The numerical results are shown below for the case when $\lambda = -20$. For this choice of λ the stability criteria becomes $\Delta t \leq 0.1$ so for the choices of the time step $\Delta t = 0.5, 0.25$ shown on the left, we expect the results to be unreliable but for the ones on the plot on the right, the stability criteria is satisfied so the results are reliable.



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 are especially important when we study initial boundary value problems.

4.2 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 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

$$\begin{aligned} Y^{n+1} &= a_{m-1}Y^n + a_{m-2}Y^{n-1} + a_{m-3}Y_{n-2} + \cdots + a_0 Y_{n+1-m} \\ &\quad + \Delta t\Big[b_m f(t_{n+1}, Y^{n+1}) + b_{m-1} f(t_n, Y^n) + b_{m-2} f(t_{n-1}, Y^{n-1}) \\ &\quad + \cdots + b_0 f(t_{n+1-m}, Y^{n+1-m})\Big]. \end{aligned} \tag{4.23}$$

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 (4.23).

In this section we begin by looking at two different approaches to derive multistep methods which involve either approximating the solution or the slope by an interpolating polynomial through the points used in the m-step method. For commonly used explicit methods we consider the family of Adams-Bashforth methods and for implicit multistep method we investigate the so-called backward difference methods and the Adams-Moulton families. Implementation issues for multistep methods are discussed. Finally we state conditions which guarantee stability of multistep methods.

4.2.1 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 on the right we have the two-step scheme

$$Y^{n+1} = Y^{n-1} + 2\Delta t f(t_n, Y^n) \tag{4.24}$$

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 to get an approximation to $y'(t)$ and substitute this approximation into the DE. If we evaluate the approximation at t_{n+1} fthen we obtain an implicit method. 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 § 4.1.2. 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 is 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

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

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
$$Y^{n+1} = \frac{4}{3}Y^n - \frac{1}{3}Y^{n-1} + \frac{2}{3}\Delta t f(t_{n+1}, Y^{n+1}) \qquad (4.25)$$

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

$$Y^{n+1} = \sum_{j=1}^{m} a_{mj} Y^{(n+1)-j} + \beta \Delta t f(t_{n+1}, Y^{n+1})\,. \qquad (4.26)$$

For the two-step scheme (4.25) we have $m = 2$, $a_{21} = 4/3$, $a_{22} = -1/3$ and $\beta = 2/3$. Table 4.1 gives coefficients for other uniform BDF formulas using the terminology of (4.26). It can be proved that the accuracy of the m-step methods included in the table is m.

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.

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 4.1: Coefficients for implicit BDF formulas of the form (4.26) where the coefficient of Y^{n+1} is normalized to one.

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 is implicit because it involves $f(t_{n+1}, Y^{n+1})$; otherwise it is 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

$$Y^{n+1} = Y^n + \frac{3}{2}\Delta t f\big(t_n, Y^n\big) - \frac{\Delta t}{2} f\big(t_{n-1}, Y^{n-1}\big)\,. \tag{4.27}$$

This is an example of an Adams-Bashforth multistep method; these methods are discussed in more detail in § 4.2.2.

4.2.2 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 § 4.2.1 we use 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 (4.23). In the exercises, you are asked to rigorously demonstrate that the local truncation error for (4.27) 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

$$\begin{aligned}
Y^{n+1} &= Y^n + \Delta t\left(\frac{3}{2}f(t_n, Y^n) - \frac{1}{2}f(t_{n-1}, Y^{n-1})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{23}{12}f(t_n, Y^n) - \frac{4}{3}f(t_{n-1}, Y^{n-1}) + \frac{5}{12}f(t_{n-2}, Y^{n-2})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{55}{24}f(t_n, Y^n) - \frac{59}{24}f(t_{n-1}, Y^{n-1}) + \frac{37}{24}f(t_{n-2}, Y^{n-2})\right.\\
&\qquad\left. -\frac{3}{8}f(t_{n-3}, Y^{n-3})\right)\\
Y^{n+1} &= Y^n + \Delta t\left(\frac{1901}{720}f(t_n, Y^n) - \frac{1387}{360}f(t_{n-1}, Y^{n-1}) + \frac{109}{30}f(t_{n-2}, Y^{n-2})\right.\\
&\qquad\left. -\frac{637}{360}f(t_{n-3}, Y^{n-3}) + \frac{251}{720}f(t_{n-4}, Y^{n-4})\right)
\end{aligned} \tag{4.28}$$

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

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

$$
\begin{aligned}
Y^{n+1} &= Y^n + \Delta t \left(\frac{5}{12} f(t_{n+1}, Y^{n+1}) + \frac{2}{3} f(t_n, Y^n) - \frac{1}{12} f(t_{n-1}, Y^{n-1}) \right) \\
Y^{n+1} &= Y^n + \Delta t \left(\frac{3}{8} f(t_{n+1}, Y^{n+1}) + \frac{19}{24} f(t_n, Y^n) - \frac{5}{24} f(t_{n-1}, Y^{n-1}) \right. \\
&\qquad \left. + \frac{1}{24} f(t_{n-2}, Y^{n-2}) \right) \\
Y^{n+1} &= Y^n + \Delta t \left(\frac{251}{720} f(t_{n+1}, Y^{n+1}) + \frac{646}{720} f(t_n, Y^n) - \frac{264}{720} f(t_{n-1}, Y^{n-1}) \right. \\
&\qquad \left. + \frac{106}{720} f(t_{n-2}, Y^{n-2}) - \frac{19}{720} f(t_{n-3}, Y^{n-3}) \right)
\end{aligned}
\tag{4.29}
$$

As mentioned, multistep methods require storage of the m previously calculated values. In the Adams family of methods only Y^n is used but the slopes at all m points must be stored. So when implementing the methods we must take this into account. For a single equation the extra storage is negligible but for systems it requires m vectors.

Another 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 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 example.

Example 4.8. STARTING VALUES FOR MULTISTEP METHODS

Implement the 3-step third order accurate Adams-Bashforth method given in (4.28) to solve the IVP

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

which has the exact solution

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

Compare the numerical rates of convergence when different methods are used 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 $Y^0 = 1$. Tabulate 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

The next example provides results for 2-step through 5-step Adams-Bashforth methods. From the previous example we see that to maintain the accuracy the starting values need to be determined by methods of order $m-1$.

Example 4.9. COMPARISON OF ADAMS-BASHFORTH METHODS

Solve the IVP from the previous example by 2-step through 5-step Adams Bashforth methods. In each case 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

4.2.3 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 give an overview of the results here.

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

We first rewrite the m-step multistep method (4.23) 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

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

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

$$\rho(z) = z^m - \sum_{j=0}^{m-1} a_j z^j \quad \text{and} \quad \sigma(z) = \sum_{j=0}^{m} b_j z^j\,. \tag{4.30}$$

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

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

In the following example we determine the region of stability for both the forward and backward Euler methods using the analysis for multistep methods. The same stability conditions as we obtained by analyzing the stability of the one-step methods using the amplification factor are realized.

Example 4.10. 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. Confirm that the same results as obtained for stability by considering the methods as single-step methods are achieved.

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.

The next example analyzes the stability of a 2-step method using the Dalhquist conditions.

Example 4.11. 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 see 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.

4.3 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 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 is used with approximations generated by methods we already have. Then we briefly discuss the modifications needed to develop the Burlisch-Stoer method.

4.3.1 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

$$f'(t) - N(h) = \frac{h}{2}f''(t) + \frac{h^2}{3!}f'''(t) + \frac{h^3}{4!}f''''(t) + \cdots . \tag{4.31}$$

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

$$f'(t) - N(\frac{h}{2}) = \frac{h}{4}f''(t) + \frac{h^2}{4\cdot 3!}f'''(t) + \frac{h^3}{8\cdot 4!}f''''(t) + \cdots . \tag{4.32}$$

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 (4.31) from twice (4.32) eliminates the terms involving h so we get

$$\begin{aligned} f'(t) - \big[2N(\frac{h}{2}) - N(h)\big] &= \big[\frac{h^2}{2\cdot 3!} - \frac{h^2}{3!}\big]f'''(t) + \big[\frac{h^3}{4\cdot 4!} - \frac{h^3}{4!}\big]f''''(t) + \cdots + \\ &= -\frac{h^2}{12}f'''(t) - \frac{3h^3}{32}f''''(t) + \cdots . \end{aligned} \tag{4.33}$$

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 (4.33) with h halved again to get

$$f'(t) - \big[2N(\frac{h}{4}) - N(\frac{h}{2})\big] = -\frac{h^2}{4\cdot 12}f'''(t) - \frac{3h^3}{8\cdot 24}f''''(t) + \cdots . \tag{4.34}$$

To eliminate the h^2 terms we need to take four times (4.34) and subtract (4.33) 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 4.12. FORWARD EULER WITH RICHARDSON EXTRAPOLATION

Consider the IVP $y'(t) = -5y(t)$ with $y(0) = 2$ whose exact solution at $t = 1$ is 0.0134759. Tabulate the results usng the forward Euler method with $\Delta t = 1/10, 1/20, \ldots 1/320$ and then use Richardson extrapolation to get a sequence of approximations which are quadratically convergent. Calculate the numerical rate of convergence.

We tabulate the results for 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, i.e., $2N(\Delta t/2) - N(\Delta t)$.

	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 (4.33) by taking the correct linear combination of (4.31) and (4.32).

4.3.2 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

$$w(x) - N(h) = K_1 h + K_2 h^2 + K_3 h^3 + K_4 h^4 + \cdots . \tag{4.35}$$

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.,

$$w(x) - N(h) = K_1 h^2 + K_2 h^4 + \cdots + K_i h^{2i} + \mathcal{O}\big((h)^{2(i+1)}\big) . \tag{4.36}$$

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 (4.35), the most efficient methods, such as the Burlisch-Stoer method, use an underlying method which has an error expansion such as (4.36).

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.

4.4 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 or only a function of t. 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 (4.12). 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

$$\begin{aligned} Y_p^{n+1} &= Y^n + \Delta t f(t_n, Y^n) \\ Y^{n+1} &= Y^n + \tfrac{\Delta t}{2}\Big[f(t_{n+1}, Y_p^{n+1}) + f(t_n, Y^n)\Big] \end{aligned} \tag{4.37}$$

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) and at t_{n+1} 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 4.13 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-trapezoidal rule to an IVP using more than one correction step.

Example 4.13. EULER-TRAPEZOIDAL PREDICTOR-CORRECTOR PAIR

Perform a single 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 use 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 $y(t) = 1/\sqrt{9-t^2}$ that can be 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

$$\begin{aligned} Y_p^{n+1} &= Y^n + \tfrac{\Delta t}{12}\left[23f(t_n, Y^n) - 16f(t_{n-1}, Y^{n-1}) + 5f(t_{i-2}, Y^{n-2})\right] \\ Y^{n+1} &= Y^n + \tfrac{\Delta t}{12}\left[5f(t_{n+1}, Y_p^{n+1}) + 8f(t_n, Y^n) - f(t_{n-1}, Y^{n-1})\right] \end{aligned} \tag{4.38}$$

Example 4.14. THIRD ORDER ADAMS-MOULTON PREDICTOR-CORRECTOR PAIR

Compare the errors and rates of convergence for the PC pair (4.38) with the third order Adams-Bashforth method defined in (4.28) for the problem in Example 4.13.

We tabulate the results below. 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		Predictor-Corrector	
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

4.5 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 (4.28) 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 Y^1 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 4.8. 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.
$$\begin{array}{c|ccc} 0 & 0 & 0 & \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

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 § 4.1.1 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 (2.8).

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 (4.16) 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 (4.27).

Chapter 5

Systems and Adaptive Step Size Methods

When modeling phenomena where we know the initial state and how it changes with time, we often have either a higher order IVP or a system of IVPs rather than a single first order IVP. In this chapter we first recall how a higher order IVP can be transformed into a system of first order IVPs. Then we extend in a straightforward manner some of the methods from Chapter **??** to systems of equations. We discuss implementation issues and give examples that illustrate the use of systems of IVPs. Then we point out how to extend our stability tests for a single equation to a system.

The final concept we investigate in our study of IVPs are methods which efficiently allow a variable time step to be taken. For these methods we need a means to estimate the next time step. If we can get an estimate for the error made at time t_n then the magnitude of the error can be used to accept or reject the step and, if the step is accepted, to estimate the next time step. Consequently, our goal is to find methods which can be used to estimate the error. One strategy is to obtain two approximations at a given time and use these to measure the error. Of course obtaining the second approximation must be done efficiently or else the cost is prohibitively large. In addition to variable step, many production codes are also variable order. We do not address these here.

5.1 Higher Order IVPs

The methods we have learned only apply to first order IVPs. However, recall from § 2.4 that it is straightforward to write an IVP of order $m > 1$ into a system of m first order coupled system of IVPs. In general, if we have the pth order IVP for $y(t)$

$$\begin{aligned} y^{[p]}(t) &= f(t, y, y', y'', \cdots, y^{[p-1]}) \quad t_0 < t \leq T \\ y(t_0) &= \alpha_1, \quad y'(t_0) = \alpha_2, \quad y''(t_0) = \alpha_3, \quad \cdots \quad y^{[p-1]}(t_0) = \alpha_p \end{aligned}$$

then we convert it to a system of p first-order IVPs by letting $w_1(t) = y(t)$, $w_2(t) = y'(t), \cdots, w_p(t) = y^{[p-1]}(t)$ which yields the first order coupled system

$$\begin{array}{rcl} w_1'(t) & = & w_2(t) \\ w_2'(t) & = & w_3(t) \\ & \vdots & \\ w_{p-1}'(t) & = & w_p(t) \\ w_p'(t) & = & f(t, w_1, w_2, \ldots, w_p) \end{array} \tag{5.1}$$

along with the initial conditions $w_k = \alpha_k$, $k = 1, 2, \ldots, p$. Thus any higher order IVP that we encounter can be transformed into a coupled system of first order IVPs.

Example 5.1. CONVERTING A HIGH ORDER IVP INTO A SYSTEM

Write the fourth order IVP

$$y^{[4]}(t) + 2y''(t) + 4y(t) = 5 \quad y(0) = 1, y'(0) = -3, y''(0) = 0, y'''(0) = 2$$

as a system of first order equations.

We want four first order differential equations for $w_i(t)$, $i = 1, 2, 3, 4$; to this end let $w_1 = y$, $w_2 = y'$, $w_3 = y''$, and $w_4 = y'''$. Using the first two expressions we have $w_1' = w_2$, and the second and third gives $w_2' = w_3$, the third and fourth gives $w_3' = w_4$ and the original differential equation provides the last first order equation $w_4' + 2w_3 + 4w_1 = 5$. The system of equations is thus

$$\begin{array}{rcl} w_1'(t) - w_2(t) & = & 0 \\ w_2'(t) - w_3(t) & = & 0 \\ w_3'(t) - w_4(t) & = & 0 \\ w_4' + 2w_3 + 4w_1 & = & 5 \end{array}$$

along with the initial conditions

$$w_1(0) = 1, \quad w_2(0) = -3, \quad w_3(0) = 0, \quad \text{and } w_4(0) = 2.$$

Oftentimes a model is already in the form of a coupled system of first order IVPs such as the predator-prey model (2.6). Our goal is to apply the methods of the previous chapter to a system of first order IVPs. The notation we use for a general system of N first order IVPs is

$$\begin{array}{rcll} w_1'(t) & = & f_1(t, w_1, w_2, \ldots, w_N) & t_0 < t \le T \\ w_2'(t) & = & f_2(t, w_1, w_2, \ldots, w_N) & t_0 < t \le T \\ & \vdots & & \\ w_N'(t) & = & f_N(t, w_1, w_2, \ldots, w_N) & t_0 < t \le T \end{array} \tag{5.2}$$

along with the initial conditions $w_k(t_0) = \alpha_k$, $k = 1, 2, \ldots, N$. For example, using this notation the pth order IVP written as the system (5.1) has $f_1 = w_2$, $f_2 = w_3$, etc.

Existence, uniqueness and continuous dependence of the solution to the system (5.2) can be established. Analogous to the case of a single IVP each function f_i must satisfy a Lipschitz condition with respect to each unknown w_i. Details of this analysis is found in standards texts in ODEs. For the sequel, we assume that each system has a unique solution which depends continuously on the data.

In the next two sections we demonstrate how one-step and multistep methods from Chapter **??** are easily extended to the system of N equations (5.2).

5.2 Single-step methods for systems

We now want to extend single-step methods to the system (5.2). For simplicity we first extend the forward Euler method for a system and then with the intuition gained from applying that method we extend a general explicit Runge-Kutta method to a system. Implicit RK methods are extended in an analogous way. We use the notation W_k^n as the approximation to $w_k(t_n)$ where the subscript of W refers to the unknown number and the superscript to the point t_n.

Suppose we have the first order system (5.2) with the initial conditions $w_k(t_0) = \alpha_k$ for $k = 1, 2, \ldots, N$. The forward Euler method for each of the $k = 1, 2, \ldots, N$ equations is

$$W_k^{n+1} = W_k^n + \Delta t f_k\big(t_n, W_1^n, W_2^n, \cdots, W_N^n\big)\,.$$

We write the Euler method as a vector equation so we can solve for all N unknowns simultaneously at each t_n; this is not necessary but results in an efficient implementation of the method. To this end we set

$$\mathbf{W}^n = \begin{pmatrix} W_1^n \\ W_2^n \\ \vdots \\ W_N^n \end{pmatrix} \quad \text{and} \quad \mathbf{F}^n = \begin{pmatrix} f_1(t_n, \mathbf{W}^n) \\ f_2(t_n, \mathbf{W}^n) \\ \vdots \\ f_N(t_n, \mathbf{W}^n)) \end{pmatrix}$$

so that $\mathbf{W}_0 = (\alpha_1, \alpha_2, \ldots, \alpha_N)^T$. For $n = 0, 1, 2, \ldots$ we have the following vector equation for the forward Euler method for a system

$$\mathbf{W}^{n+1} = \mathbf{W}^n + \Delta t \mathbf{F}^n\,. \tag{5.3}$$

To implement the scheme at each point t_n we have N function evaluations to form the vector $\mathbf{F}^n$, then we perform the scalar multiplication to get $\Delta t \mathbf{F}^n$ and then a vector addition to obtain the final result $\mathbf{W}^{n+1}$. To compute the error at a specific time, we have to take into account the fact that the approximation is now a vector instead of a scalar. Also the exact solution is a vector of each w_i evaluated at the specific time the error is determined. We can easily calculate an error vector as the difference in these two vectors. To obtain a single number to use in the calculation of a numerical rate, we must use a vector norm. A common vector norm is the standard Euclidean norm which is often called ℓ_2 norm or the "little l2 norm". Another commonly used vector norm is the max or infinity norm. See the appendix for details.

Example 5.2. FORWARD EULER FOR A SYSTEM

Consider the system of three IVPs

$$\begin{aligned} w_1'(t) &= 2w_2(t) - 4t \qquad 0 < t < 10 \\ w_2'(t) &= -w_1(t) + w_3(t) - e^t + 2 \qquad 0 < t < 10 \\ w_3'(t) &= w_1(t) - 2w_2(t) + w_3(t) + 4t \qquad 0 < t < 10 \\ w_1(0) &= -1, \qquad w_2(0) = 0, \qquad w_3(0) = 2 \end{aligned}$$

for the unknown $(w_1, w_2, w_3)^T$ whose exact solution is $(-\cos(2t), \sin(2t) + 2t, \cos(2t) + e^t)^T$. Determine by hand an approximation at $t = 0.2$ using $\Delta t = 0.1$ and the forward Euler method. Calculate the Euclidean or ℓ_2-norm of the error at $t = 0.2$.

For this problem we have

$$\mathbf{W}^0 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \quad \text{and} \quad \mathbf{F}^n = \begin{pmatrix} 2W_2^n - 4t_n \\ -W_1^n + W_3^n - e^{t_n} + 2 \\ W_1^n - 2W_2^n + W_3^n + 4t_n \end{pmatrix}$$

so that with $t_n = t_0 = 0$

$$\mathbf{F}^0 = \begin{pmatrix} 2(0) - 4(0) \\ -(-1) + 2 - e^0 + 2 \\ -1 - 0 + 2 + 4(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix}.$$

With $\Delta t = 0.1$ we determine $\mathbf{W}^1$ from

$$\mathbf{W}^1 = \mathbf{W}^0 + \Delta t \mathbf{F}^0 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + 0.1 \begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} -1.0 \\ 0.4 \\ 2.1 \end{pmatrix}.$$

Now to determine $\mathbf{W}^2 = \mathbf{W}^1 + \Delta t \mathbf{F}^1$ we need $\mathbf{F}^1$ which is found by using $\mathbf{W}^1$ and $t_1 = \Delta t = 0.1$ in the definition of $\mathbf{F}^n$. We have

$$\mathbf{F}^1 = \begin{pmatrix} 2(0.4) - 4(.1) \\ 1 + 2.1 - e^{.1} + 2 \\ -1 - 2(.4) + 2.1 + 4(.1) \end{pmatrix} = \begin{pmatrix} 0.400 \\ 3.995 \\ 0.700 \end{pmatrix}$$

so that

$$\mathbf{W}^2 = \begin{pmatrix} -1.0 \\ 0.4 \\ 2.1 \end{pmatrix} + 0.1 \begin{pmatrix} 0.400 \\ 3.995 \\ 0.700 \end{pmatrix} = \begin{pmatrix} -0.9600 \\ 0.7995 \\ 2.1700 \end{pmatrix}.$$

The exact solution at $t = 0.2$ is $(-0.921061, 0.789418, 2.14246)^T$. Unlike the case of a single IVP we now have an error *vector* instead of a single number; at $t = 0.2$ the error vector in our calculation is $(0.038939, .010082, .02754)^T$. To obtain a single number from this vector to use in the calculation of a numerical rate, we must use a vector norm. For this calculation at $t = 0.2$ the Euclidean norm of the error is $1.98\ 10^{-2}$.

Suppose now that we have an s-stage RK method; recall that for a single first order equation we have s function evaluations for each t_n. If we have N first order IVPs, then we need sN function evaluations at each t_n. For example, if we use a 4-stage RK with 10,000 equations then at each time we need 40,000 function

evaluations; if we do 100 time steps then we have 4 million function evaluations. If function evaluations are expensive, multistep methods may be more efficient.

In an s-stage RK method for a single equation we must compute each k_i, $i = 1, 2, \ldots, s$ as defined in (4.16). For a system, we have a vector of slopes so each scalar k_i in (4.16) is now a vector. Thus for a system an s-stage RK method is written as

$$\begin{aligned}
\mathbf{k}_1 &= \Delta t \mathbf{F}\big(t_n, \mathbf{W}^n\big) \\
\mathbf{k}_2 &= \Delta t \mathbf{F}\big(t_n + c_2\Delta t, \mathbf{W}^n + a_{21}\mathbf{k}_1\big) \\
\mathbf{k}_3 &= \Delta t \mathbf{F}\big(t_n + c_3\Delta t, \mathbf{W}^n + a_{31}\mathbf{k}_1 + a_{32}\mathbf{k}_2\big) \\
&\vdots \\
\mathbf{k}_s &= \Delta t \mathbf{F}\big(t_n + c_s\Delta t, \mathbf{W}^n + a_{s1}\mathbf{k}_1 + a_{s2}\mathbf{k}_2 + \cdots + a_{ss-1}\mathbf{k}_{s-1}\big) \\
\mathbf{W}^{n+1} &= \mathbf{W}^n + \sum_{j=1}^{s} b_j \mathbf{k}_j \,.
\end{aligned}$$

For example, the 2-stage Heun RK method has coefficients $c_1 = 0$, $c_2 = 2/3$, $a_{21} = 2/3$, $b_1 = 1/4$ and $b_2 = 3/4$ so for a system we have

$$\begin{aligned}
\mathbf{k}_1 &= \Delta t \mathbf{F}(t_n, \mathbf{W}^n) \\
\mathbf{k}_2 &= \Delta t \mathbf{F}(t_n + \tfrac{2}{3}\Delta t, \mathbf{W}^n + \tfrac{2}{3}\mathbf{k}_1) \\
\mathbf{W}^{n+1} &= \mathbf{W}^n + \tfrac{1}{4}\mathbf{k}_1 + \tfrac{3}{4}\mathbf{k}_2 \,.
\end{aligned}$$

The following example uses the Heun method to approximate the solution to the IVP in the previous example by hand and then we see how the approximation can be computed using dot and matrix products so we can efficiently implement the method.

Example 5.3. HEUN METHOD FOR A SYSTEM

By hand approximate the solution to the system at $t = 0.2$ given in the previous example using the Heun method. Calculate the Euclidean error at $t = 0.2$ using $\Delta t = 0.1$. Compare with the results for the forward Euler method in the previous example. Then redo the calculation using dot and matrix products to illustrate how it might be implemented on a computer.

As in the previous example, $\mathbf{W}^0 = (-1, 0, 2)^T$ and $\mathbf{F}^n = \big(2W_2^n - 4t_n, -W_1^n + W_3^n - e^{t_n} + 2, W_1^n - 2W_2^n + W_3^n + 4t_n\big)^T$. For the first step of length $\Delta t = 0.1$ we have $\mathbf{k}_1 = 0.1(0, 4, 1)^T$ and to determine $\mathbf{k}_2$ we need to evaluate $\mathbf{F}$ at $(\frac{2}{3}(.1), \mathbf{W}_0 + \frac{2}{3}\mathbf{k}_1)$; performing this calculation gives $\mathbf{k}_2 = (.026667, .399773, .08)^T$ so that

$$\mathbf{W}^1 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 0.0 \\ 0.4 \\ 0.1 \end{pmatrix} + \frac{3}{4}\begin{pmatrix} .026667 \\ .399773 \\ .080000 \end{pmatrix} = \begin{pmatrix} -0.98000 \\ 0.39983 \\ 2.08500 \end{pmatrix}.$$

Similarly for the approximation at 0.2 we have

$$\mathbf{W}^2 = \begin{pmatrix} -0.98000 \\ 0.39983 \\ 2.08500 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 0.039966 \\ 0.395983 \\ -0.070534 \end{pmatrix} + \frac{3}{4}\begin{pmatrix} 0.066097 \\ 0.390402 \\ 0.051767 \end{pmatrix} = \begin{pmatrix} -0.92040 \\ 0.79163 \\ 2.14150 \end{pmatrix}.$$

The exact solution at $t = 0.2$ is $(-0.921061, 0.789418, 2.14246)^T$ giving an error vector of $(0.000661, .002215, .000096)^T$; calculating the standard Euclidean norm of the error and normalizing by the Euclidean norm of the solution gives 1.0166×10^{-3} which is considerably smaller than we obtained for the forward Euler.

Now we want redo the problem using dot and matrix products. Let

$$\mathbf{c} = \begin{pmatrix} 0 \\ \frac{2}{3} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \frac{1}{4} \\ \frac{3}{4} \end{pmatrix}, \quad a = \begin{pmatrix} 0 & 0 \\ \frac{2}{3} & 0 \end{pmatrix}, \quad K = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

As before $N = 3$, $\mathbf{W}^0 = (-1, 0, 2)^T$ and to get $\mathbf{W}^1$ we compute

$$\mathbf{W}^0 + K\mathbf{b}$$

using a matrix times vector operation. To form the two columns of K we need to determine the point where we evaluate $\mathbf{F}$. To determine the first column of K, i.e., $\mathbf{k}_1$ we have

$$\begin{aligned} t_{\text{eval}} &= t + c(1)\Delta t = 0 + 0(0.1) = 0 \\ \mathbf{W}_{\text{eval}} &= \mathbf{W}^0 + K\big(a(1,:)\big)^T = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \\ &= \\ K(\cdot, 1) &= \Delta t \mathbf{F}(t_{\text{eval}}, \mathbf{W}_{\text{eval}}) = 0.1 \begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.0 \\ 0.4 \\ 0.1 \end{pmatrix} \end{aligned}$$

and for the second column of K

$$\begin{aligned} t_{\text{eval}} &= t + c(2)\Delta t = 0 + \frac{2}{3}(0.1) = \frac{0.2}{3} \\ \mathbf{W}_{\text{eval}} &= \mathbf{W}^0 + K\big(a(2,:)\big)^T = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ .4 & 0 \\ .1 & 0 \end{pmatrix} \begin{pmatrix} 2/3 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ .8/3 \\ 2 + .2/3 \end{pmatrix} \\ &= \\ K(\cdot, 2) &= \Delta t \mathbf{F}(t_{\text{eval}}, \mathbf{W}_{\text{eval}}) = \begin{pmatrix} 0.026667 \\ 0.399773 \\ 0.08 \end{pmatrix} \end{aligned}$$

Now we want to determine how to modify our implementation of an explicit RK method for a single IVP to incorporate a system. The coefficients of the particular RK method do not change but, of course, the solution at each time t_n is now a vector as well as the slope $\mathbf{F}(t_n, W^n)$. The routine which evaluates the slope requires the scalar time and a N-vector as input and outputs an N-vector which is the slope at the given point. For a single equation using a stage s method we have s scalars k_i which we store as an s-vector so we need a two dimensional array to store the K which is dimensioned by the number of equations N by the number of stages s in the method. The following pseudocode illustrates how one could modify the routine for advancing RK for a single equation.

Algorithm 5.1 : Advancing explicit RK for a system one time step

Assume: Coefficients in RK method are stored as follows: c_i and b_i, $i = 1, \ldots, s$ are stored in 1D array and coefficients a_{ij}, $i, j = 1, \ldots, s$, in a 2D array

Input: the solution W at the time t stored as an N-vector, the uniform time step Δt, the number of stages s, coefficients a, b, c. $\mathbf{k}$ is a two dimensional $N \times s$ array.

Output: the solution W and the new time t

Loop over number of stages:

$K = 0$

for $i = 1, \ldots, s$

$t_{\text{eval}} = t + c(i)\Delta t$

$W_{\text{eval}} = W + \text{matrix\_product}\big(K, a(i,:)^T\big)$

$K(:, i) = \Delta t f(t_{\text{eval}}, W_{\text{eval}})$

$W = W + \text{matrix\_product}\big(K, b\big)$

$t = t + \Delta t$

Example 5.4. RATES OF CONVERGENCE FOR RK FOR A SYSTEM

For the problem in Example 5.2 apply the forward Euler and the Heun method with $\Delta t = 1/10, 1/20, \ldots, 1/80$ and output the ℓ_2 and ℓ_∞ norm of the normalized error at $t = 1$. Compute the numerical rates of convergence and compare with the results for a single IVP.

In the table below we tabulate the results using the forward Euler method for this system at $t = 1$ where both the normalized ℓ_2-norm and ℓ_∞-norm (i.e., the maximum norm) of the normalized error normalized by the corresponding norm of the solution is reported. Clearly we have linear convergence as we did in the case of a single equation.

Δt	ℓ_2 Error	rate	ℓ_∞ Error	rate
$1/10$	$6.630\ 10^{-2}$		$6.019\ 10^{-2}$	
$1/20$	$3.336\ 10^{-2}$	0.99	$3.156\ 10^{-2}$	0.93
$1/40$	$1.670\ 10^{-2}$	1.00	$1.631\ 10^{-2}$	0.95
$1/80$	$8.350\ 10^{-3}$	1.00	$8.277\ 10^{-3}$	0.98

In the table below we tabulate the results using the Heun method for this system at $t = 1$ where both the normalized ℓ_2-norm and ℓ_∞-norm (i.e., the maximum norm) of the error normalized by the corresponding norm of the solution is reported. Clearly we have quadratic convergence as we did in the case of a single equation.

Δt	ℓ_2 Error	rate	ℓ_∞ Error	rate
$1/10$	$5.176\ 10^{-3}$		$5.074\ 10^{-3}$	
$1/20$	$1.285\ 10^{-3}$	2.01	$1.242\ 10^{-3}$	2.03
$1/40$	$3.198\ 10^{-4}$	2.01	$3.067\ 10^{-4}$	2.02
$1/80$	$7.975\ 10^{-5}$	2.00	$7.614\ 10^{-5}$	2.01

5.3 Multistep methods for systems

Recall that explicit multistep methods use not only the approximation at t_n but other nearby calculated times to extrapolate the solution to the new point. The m-step explicit method from § 4.2.2 for a single IVP is

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

For a system of N equations the function f is now a vector $\mathbf{F}$ so we must store its value at the previous m steps. In the Adams-Bashforth or Adams Moulton methods $a_0, a_1, \ldots, a_{m-2} = 0$ so only the solution at t_n is used. This saves additional storage because we only have to store m slope vectors and a single vector approximation to the solution. So for the system of N equations using an m-step Adams-Bashforth method we must store $(m+1)$ vectors of length N. Remember that an m-step method requires m starting values so we need to calculate $m-1$ values from a single-step method.

As a concrete example, consider the 2-step Adams-Bashforth method

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

for a single IVP. Using the notation of the previous section we extend the method for the system of N equations as

$$\mathbf{W}^{n+1} = \mathbf{W}^n + \Delta t\Big[\frac{3}{2}\mathbf{F}(t_n, \mathbf{W}^n) - \frac{1}{2}\mathbf{F}(t_{n-1}, \mathbf{W}^{n-1})\Big]. \tag{5.4}$$

At each step we must store three vectors $\mathbf{W}^n$, $\mathbf{F}(t_n, \mathbf{W}^n)$, and $\mathbf{F}(t_{n-1}, \mathbf{W}^{n-1})$. In the next example we apply this 2-step method to the system of the previous examples.

Example 5.5. ADAMS-BASHFORTH METHOD FOR A SYSTEM

Apply the 2-step Adams-Bashforth method (5.4) to the system of Example 5.2 to approximate by hand the solution at $t = 0.2$ using a time step of $\Delta t = 0.1$. Give the Euclidean norm of the error. Use appropriate starting values. Then tabulate the results at $t = 1$ for a sequence of Δt approaching zero and calculate the numerical rate of convergence.

Because this is a 2-step method we need two starting values. We have the initial condition for $\mathbf{W}^0$ but we also need $\mathbf{W}^1$. Because this method is second order we can use either a first or second order scheme to generate an approximation to $\mathbf{W}^1$. Here we use the results from the Heun method in Example 5.3 for $\mathbf{W}^1$. Consequently we have

$$\mathbf{W}^0 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \quad \text{and} \quad \mathbf{W}^1 = \begin{pmatrix} -0.98000 \\ 0.39982 \\ 2.08500 \end{pmatrix}.$$

From the previous example we have $\mathbf{F}(0, \mathbf{W}^0) = (0.0, 4.0, 1.0)^T$ and $\mathbf{F}(0.1, \mathbf{W}^1)$=$(.39966, 3.95982, -.704659)^T$. Then $\mathbf{W}^2$ is given by

$$\mathbf{W}^2 = \begin{pmatrix} -0.98000 \\ 0.39982 \\ 2.08500 \end{pmatrix} + 0.1 \left[\frac{3}{2} \begin{pmatrix} 0.39966 \\ 3.95983 \\ -0.70466 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 0.0 \\ 4.0 \\ 1.0 \end{pmatrix} \right] = \begin{pmatrix} -0.92005 \\ 0.79380 \\ 1.92930 \end{pmatrix}.$$

The table below tabulates the errors at $t = 1$. Of course we can only use the starting value $\mathbf{W}^1 = (. - 0.98000, 0.39982, 2.08500)^T$ as starting values for the computations at $\Delta t = 0.1$; for the other choices of Δt we must generate starting values because t_1 is different. From the results we see that the rate is two, as expected.

Δt	ℓ_2 Error	rate	ℓ_∞ Error	rate
$1/10$	$1.346\ 10^{-2}$		$1.340\ 10^{-2}$	
$1/20$	$3.392\ 10^{-3}$	1.99	$3.364\ 10^{-3}$	1.99
$1/40$	$8.550\ 10^{-4}$	1.99	$8.456\ 10^{-4}$	1.99
$1/80$	$2.149\ 10^{-5}$	1.99	$2.121\ 10^{-5}$	1.99

5.4 Stability of Systems

Oftentimes we have a system of first order IVPs or we have a higher order IVP which we first write as a system of first order IVPs. We want to extend our definition of absolute stability to a system but we first look at stability for the differential equations themselves. Analogous to the problem $y'(t) = \lambda y$ we consider the linear model problem

$$\mathbf{w}'(t) = A\mathbf{w}$$

for an $N \times N$ system of IVPs where A is an $N \times N$ matrix. Consider first the simple case where A is a diagonal matrix and the equations are uncoupled so basically we have the same situation as a single equation. Thus the stability criteria is that the real part of each diagonal entry must be less than or equal to zero. But the diagonal entries of a diagonal matrix are just its N eigenvalues λ_i[1] counted according to multiplicity. So an equivalent statement of stability when A is diagonal is that $\text{Re}(\lambda_i) < 0$, $i = 1, 2, \ldots, N$; it turns out that this is the stability criteria for a general matrix A. Recall that even if the entries of A are real the eigenvalues may be complex. If A is symmetric we are guaranteed that the eigenvalues are real. If we

[1]The eigenvalues of an $N \times N$ matrix A are scalars λ such that $A\mathbf{x} = \lambda\mathbf{x}$; the vector $\mathbf{x}$ is called the eigenvector corresponding to the eigenvalue λ.

have the general system (5.2) where $f_i(t, \mathbf{w})$ is not linear in $\mathbf{w}$, then the condition becomes one on the eigenvalues of the Jacobian matrix for f where the (i, j) entry of the Jacobian is $\partial f_i/\partial w_j$.

Now if we apply the forward Euler method to the system $\mathbf{w}'(t) = A\mathbf{w}$ where the entries of A are a_{ij} then we have the system

$$\mathbf{W}^{n+1} = \begin{pmatrix} 1+\Delta t a_{11} & \Delta t a_{12} & \Delta t a_{13} & \cdots & \Delta t a_{1N} \\ \Delta t a_{21} & 1+\Delta t a_{22} & \Delta t a_{23} & \cdots & \Delta t a_{2N} \\ & \ddots & \ddots & & \\ & \cdots & \cdots & \Delta t a_{N,N-1} & 1+\Delta t a_{N,N} \end{pmatrix} \mathbf{W}^n$$

The condition on Δt is determined by choosing it so that all the eigenvalues of the matrix have real parts less than zero.

5.5 Adaptive time step methods

If the solution to a differential equation varies rapidly over a portion of the time interval and slowly over the remaining time, then clearly using a fixed time step is inefficient. In this section we want to investigate methods which allow us to estimate an error and then use this error to decide if the time step can be increased or if it should be decreased. We have already encountered Predictor-Corrector methods which can easily provide an estimate for the error by comparing the results of the predictor and corrector steps. Another approach is to use two approximations such as a p-order and a $(p+1)$-order RK method and compare the local truncation error between the two which should be $\mathcal{O}(\Delta t)^1$. Of course, in order to do this efficiently the methods should be nested in the sense that the function values required for the $(p+1)$-order method include all the function evaluations from the p-order method; for this reason the methods are called embedded RK methods. The best known of these methods is the Runge-Kutta-Fehlberg method which uses a fourth and fifth order explicit RK method (RKF45). We begin this section with a simple example illustrating how to approximations at a given point help us to determine whether to accept the answer or not and, if accepted, to predict a new step size.

5.5.1 Adaptive methods using Richardson extrapolation

We begin this section by first considering a very simple example of an algorithm which produces an automatic step size selector. In adaptive time step methods Δt is no longer constant so we for use Δt_n where for M time steps we set

$$\Delta t_0 = t_1 - t_0, \quad \Delta t_1 = t_2 - t_1, \cdots, \Delta t_n = \Delta t_{n+1} - \Delta t_n, \cdots, \Delta t_{M-1} = t_M - t_{M-1}\,.$$

Recall that Richardson extrapolation is introduced in § 4.3 as a technique to combine lower order approximations to get more accurate approximations. We explore this idea by comparing an approximation at t_{n+1} obtained from t_n using a step size of Δt_n and a second one at t_{n+1} which obtained from t_n by taking two steps with step size $\Delta t_n/2$. We want to see how these two approximations are used to estimate the

local truncation error and provide an estimate for the next time step Δt_{n+1} as well as determining if we should accept the result at t_{n+1}. To describe this approach we use the forward Euler method because its simplicity should make the technique clear.

We assume we have the solution at t_n and have an estimate for the next time step Δt_n and the goal is to determine whether this estimate the next time step Δt_{n+1}. First we take an Euler step with Δt_n to get the approximation Y_1^{n+1} where the subscript denotes the specific approximation because we have two. Next we take two Euler steps starting from t_n using a step size of $\Delta t_n/2$ to get the approximation Y_2^{n+1}.

Recall that the local truncation error for the forward Euler method using a step size of Δt is $C(\Delta t)^2 + \mathcal{O}(\Delta t)^3$. Thus the exact solution satisfies

$$y(t_{n+1}) = y(t_n) + \Delta t f(t_n, y(t_n)\big) + C(\Delta t)^2 + \mathcal{O}(\Delta t)^3\,,$$

where we have equality because we have included the local truncation error. Consequently, for our solution Y_1^{n+1} we have the local truncation error $C(\Delta t_n)^2 + \mathcal{O}(\Delta t_n)^3$, i.e.,

$$y(t_{n+1}) - Y_1^{n+1} = C(\Delta t_n)^2 + \mathcal{O}(\Delta t_n)^3\,.$$

Now for Y_2^{n+1} we take two steps each with a local truncation error of $C(\Delta t_n)^2/4 + \mathcal{O}(\Delta t_n/2)^3$ so basically at t_{n+1} we have twice this error

$$y(t_{n+1}) - Y_2^{n+1} = C(\Delta t_n)^2/2 + \mathcal{O}(\Delta t_n/2)^3\,.$$

Now using Richardson extrapolation the solution $2Y_2^{n+1} - Y_1^{n+1}$ eliminates the $(\Delta t_n)^2$ so that is has a local truncation error of $\mathcal{O}(\Delta t_n^3)$.

We want to see how to use these truncation errors to decide whether to accept or reject the improved solution $2Y_2^{n+1} - Y_1^{n+1}$ at t_{n+1} and if we accept it to choose a new time step Δt_{n+1}. Suppose that the user inputs a tolerance for the maximum rate of increase in the error. We have an estimate for this rate, r_n, from our solutions so we require it to be less than the given tolerance σ, i.e.,

$$r_n = \frac{|Y_1^{n+1} - Y_2^{n+1}|}{\Delta t_n} \leq \text{prescribed tolerance} = \sigma\,. \tag{5.5}$$

If this is satisfied then we accept the improved solution $2Y_2^{n+1} - Y_1^{n+1}$; otherwise we reduce Δt_n and repeat the procedure. We estimate the next step size by computing the ratio of the acceptable rate (i.e., the prescribed tolerance) and the actual rate r_n. Then we take this fraction of the current step size to estimate the new one. In practice, one often adds a multiplicative factor less than one for "safety" since we have made certain assumptions. For example, we could compute a step size from

$$\Delta t_{n+1} = \gamma \left(\frac{\sigma}{r_n}\right) \Delta t_n \quad \text{where} \quad \gamma < 1\,. \tag{5.6}$$

From this expression we see that if the acceptable rate σ is smaller than the actual rate r_n then Δt_n is decreased. If the criteria (5.5) is not satisfied then we must

repeat the step with a reduced time step. We can estimate this from (5.6) without the safety factor because in this case $r_n > \sigma$ so the fraction is less than one. The following example illustrates this technique.

Example 5.6. Consider the IVP

$$y'(t) = -22ty(t) \quad -1 < t \leq 1 \quad y(-1) = e^{-7}$$

whose exact solution is $y(t) = e^{4-11t^2}$; the solution is small and varies slowly in the domain $[-1, -.5] \cup [.5, 1]$ but peaks to over fifty at the origin. Below is a plot of the point wise error when $\Delta t = 0.001$. As can be seen from the plot, a variable time step needs to be used.

Use the algorithm described above with Richardson extrapolation to solve this IVP using the time step formula (5.6). Choose the tolerance $\sigma = 0.01$, $\gamma = 0.75$ and the initial time step of $\Delta t = 0.01$. Tabulate the (5.5) along with whether the step is accepted or rejected and the new step size at several different times. The starting step size is chosen to be $\Delta t_0 =$ and the tolerance $\sigma =$.

initial Δt_n	$\frac{r_n}{\sigma}$	accept/reject (if rejected)	new Δt_n (if accepted)
0.01	0.1	accept	0.0716
0.0716	.87	accept	0.0614
0.0614	2.0	reject	0.0460
0.0460	1.5	reject	0.0345
0.0345	1.1	reject	0.0259
0.0259	.86	accept	0.0225
0.0225	1.12	reject	0.0169
0.0169	.84	accept	0.0150
0.0150	.98	accept	0.0115
0.0115	.95	accept	

5.5.2 Adaptive methods using predictor-corrector schemes

Using predictor-corrector pairs also provides a way to estimate the error and thus determine if the current step size is appropriate. For example, for the third order predictor and corrector pair given in (4.38) one can specifically compute the constant in the local truncation error to get

$$|y(t_{n+1}) - Y_p^{n+1}| = \frac{9}{24} y^{[4]}(\xi)(\Delta t)^4 \qquad |y(t_{n+1}) - Y^{n+1}| = \frac{1}{24} y^{[4]}(\eta)(\Delta t)^4 \,.$$

For small Δt we assume that the fourth derivative is constant over the interval and so the coefficient in the local truncation error for Y_p^{n+1} is approximately nine times the local error for Y^{n+1}. We have

$$|y(t_{n+1}) - Y^{n+1}| \approx \frac{1}{9} |y(t_{n+1}) - Y_p^{n+1}| \,.$$

If the step size Δt is too large, then the assumption that the fourth derivative is constant from t_n to t_{n+1} may not hold and the above relationship is not true. Typically the exact solution $y(t_{n+1})$ is not known so instead we monitor the difference in the predicted and corrected solution $|Y^{n+1} - Y_p^{n+1}|$. If it is larger than our prescribed tolerance, then the step is rejected and Δt is decreased. Otherwise the step is accepted; if the difference is below the minimum prescribed tolerance then the step size is increased in the next step.

5.5.3 Embedded RK methods

To use RK methods for step size control we use two different methods to approximate the solution at t_{n+1} and compare the approximations which gives an approximation to the local truncation error. If the results are close, then we are confident that a correct step size was chosen; if they vary considerably then we assume that too large a step size was used and we reduce the time step and repeat the calculation. If they are extremely close then this suggests a larger step size can be used on the next step. Typically, an ad hoc formula is used to estimate the next time step based on these observations. Of course, to efficiently implement this approach we should choose the methods so that they have function evaluations in common to reduce the work.

A commonly used RK pair for error control is a combination of a fourth and fifth order explicit method; it is called the Runge-Kutta-Fehlberg method and was developed by the mathematician Erwin Fehlberg in the late 1960's. It is typically known by the acronym RKF45. For many years it has been considered the "work horse" of IVP solvers. Recall that to get an accuracy of $(\Delta t)^5$ at least six function evaluations are required. The six function evaluations are

$$\begin{aligned}
k_1 &= f\left(t_n, Y^n\right) \\
k_2 &= f\left(t_n + \frac{1}{4}\Delta t, Y^n + \frac{1}{4}k_1\right) \\
k_3 &= f\left(t_n + \frac{3}{8}\Delta t, Y^n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right) \\
k_4 &= f\left(t_n + \frac{12}{13}\Delta t, Y^n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right) \\
k_5 &= f\left(t_n + \Delta t, Y^n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right) \\
k_6 &= f\left(t_n + \frac{1}{2}\Delta t, Y^n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right).
\end{aligned}$$

The fourth order RK method which uses the four function evaluations k_1, k_3, k_4, k_5

$$Y^{n+1} = Y^n + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5 \tag{5.7}$$

is used to first approximate $y(t_{n+1})$ and then the fifth order RK method

$$Y^{n+1} = Y^n + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6 \qquad (5.8)$$

is used for comparison. Note that the fifth order method uses all of the coefficients of the fourth order method so it is efficient to implement because it only requires an additional function evaluation. For this reason, we call RK45 an embedded method. Also note that the fourth order method is actually a 5-stage method but remember that no 5-stage method is fifth order. Typically the Butcher tableau for the coefficients c_i and a_{ij} is written for the higher order method and then two lines are appended at the bottom for the coefficients b_i in each method. For example, for RKF45 the tableau is

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & 0 \\
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array}$$

To implement the RKF45 scheme we find two approximations at t_{n+1}; Y_4^{n+1} using the fourth order scheme (5.7) and Y_5^{n+1} using the fifth order scheme (5.8). We then determine the difference $|Y_5^{n+1} - Y_4^{n+1}|$ which should be $\mathcal{O}(\Delta t)$. This error is used to make the decision whether to accept the step or not; if we accept the step then the decision must be made whether or not to increase the step size for the next calculation or keep it the same. One must a priori choose a minimum and maximum acceptable value for the normalized difference between $|Y_5^{n+1} - Y_4^{n+1}|$ and use these values for deciding whether a step is acceptable or not. To implement the RK45 method the user needs to input a maximum and minimum time step so that we never allow the time step to get too large or too small. Typically the code has some default values that the user can either accept or modify.

5.6 Stiff systems

Some differential equations are more difficult to solve than others. We know that for problems where the solution curve varies a lot, we should take a small step size and where it changes very little a larger step size should be used for efficiency. If the change in the solution curve is relatively small everywhere then a uniform step size is the most efficient approach. This all seems very heuristic. However, there are problems which require a very small step size even when the solution curve is very smooth. There is no universally accepted definition of stiff differential equations

but typically the solution curve changes rapidly and then tends towards a slowly-varying solution. Because the stability region for implicit methods is typically much larger than explicit methods, most stiff equations are approximated using an implicit method.

To illustrate the concept of stiffness we look at a single IVP which is considered stiff. The example is from a combustion model and is due to Shampine (2003) who is one of the authors of the Matlab ODE suite. The idea is to model flame propagation as when you light a match. We know that the flame grows rapidly initially until it reaches a critical size which is dependent on the amount of oxygen. We assume that the flame is a ball and $y(t)$ represents its radius; in addition we assume that the problem is normalized so that the maximum radius is one. We have the IVP

$$y'(t) = y^2(1-y) \quad 0 < t \leq \frac{2}{\delta}; \qquad y(0) = \delta \tag{5.9}$$

where $\delta << 1$ is the small given initial radius. At ignition the solution y increases rapidly to a limiting value of one; this happens quickly on the interval $[0, 1/\delta]$ but on the interval $[1/\delta, 2/\delta]$ the solution is approximately equal to one. Knowing the behavior of the problem suggests that we should take a small step size initially and then on $[1/\delta, 2/\delta]$ where the solution is almost constant we should be able to take a large step size. However, if we use the RKF45 method with an automatic step size selector, then we can capture the solution on $[0, 1/\delta]$ but on $[1/\delta, 2/\delta]$ the step size is reduced by so much that the minimum allowable step size is surpassed and the method often fails if the minimum step size is set too large. Initially the problem is not stiff but it becomes stiff as its approaches the value one, its steady state solution. The term "stiff" was used to described this phenomena because it was felt the steady state solution is so "rigid".

When one has a system of equations like (5.2) the stiffness of the problem depends upon the eigenvalues of the Jacobian matrix. Recall that we said we need all eigenvalues to have real part less than zero for stability. If the Jacobian matrix has eigenvalues which have a very large negative real part and eigenvalues with a very small negative real part, then the system is stiff and special care must be used to solve it. You probably don't know a priori if a system is stiff but if you encounter behavior where the solution curve is not changing much but you find that your step size needs to be smaller and smaller, then your system is probably stiff. In that case, an implicit method is typically used.