The Initial Value Problem for Ordinary Differential Equations

Nguyen Quan Ba Hong*
Doan Tran Nguyen Tung†
Nguyen An Thinh‡
Students at Faculty of Math and Computer Science
Ho Chi Minh University of Science, Vietnam
email. nguyenquanbahong@gmail.com
email. dtrngtung@live.com
email. anthinh297@gmail.com
blog. http://hongnguyenquanba.wordpress.com §

February 19, 2017

Abstract

This assignment aims to solve initial value problems for ordinary differential equations numerically. The Matlab implementation is given at the end of this context.

*Student ID. 1411103
†Student ID. 1411352
‡Student ID. 1411289

Contents

List of Figures

1 Introduction to Initial Value Problems for Ordinary Differential Equations

In this context we begin a study of time-dependent differential equations, beginning with the initial value problem (IVP) for a time-dependent ordinary differential equation (ODE).

The IVP takes the form

$$u'(t) = f(u(t), t) \text{ for } t > t_0 \tag{1.1}$$

with some initial data

$$u(t_0) = \eta \tag{1.2}$$

We will often assume $t_0 = 0$ for simplicity.

In general, (1.1) may represent a system of ODEs, i.e., u may be a vector with s components $u_1, \ldots, u_s$, and then $f(u,t)$ also represents a vector with components $f_1(u,t), \ldots, f_s(u,t)$, each of which can be a nonlinear function of all the components of u.

We will consider only the first order equation (1.1), but in fact this is more general than it appears since we can reduce higher order equations to a system of first order equations.

Indeed, consider the general IVP for the ODE of degree n,

$$v^{(n)}(t) = F\left(t, v(t), v'(t), \ldots, v^{(n-1)}(t)\right), t > 0 \tag{1.3}$$

This nth order equation requires n initial conditions, typically specified as

$$v^{(k)}(0) = \eta_{k+1}, \quad k = 0, 1, \ldots, n-1 \tag{1.4}$$

We can rewrite (1.3)-(1.4) as a system of the form (1.1)-(1.2) by introducing new variables

$$u_{k+1}(t) = v^{(k)}(t), \quad k = 0, 1, \ldots, n-1 \tag{1.5}$$

Then the equation takes the form

$$u_k{}'(t) = u_{k+1}(t), \quad k = 1, 2, \ldots, n-1 \tag{1.6}$$

$$u_n{}'(t) = F(t, u_1(t), u_2(t), \ldots, u_n(t)) \tag{1.7}$$

Put

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \tag{1.8}$$

$$f(u(t), t) = \begin{bmatrix} u_2(t) \\ \vdots \\ u_n(t) \\ F(t, u_1(t), u_2(t), \ldots, u_n(t)) \end{bmatrix} \tag{1.9}$$

The initial condition is simply (1.2), where the n components of η are obtained by combining (1.4) and (1.5), i.e.,

$$u_{k+1}(0) = \eta_{k+1}, \quad k = 0, 1, \ldots, n-1 \tag{1.10}$$

Using (1.8)-(1.10), (1.3)-(1.7) can be rewritten briefly as

$$u'(t) = f(u(t), t) \tag{1.11}$$

$$u(0) = \eta \tag{1.12}$$

Hence, any single equation of order m can be reduced to m first order equations by defining $u_j(t) = v^{(j-1)}(t)$.

More generally, an mth order system of s equations can be reduced to a system of ms first order equations.

See [1] Section D.3.1, for an example of how this procedure can be used to determine the general solution of an rth order linear differential equation.

It is also sometimes useful to note that any explicit dependence of f on t can be eliminated by introducing a new variable that is simply equal to t. In the above general establishment for nth order ODEs, we could define

$$u_{n+1}(t) = t \tag{1.13}$$

so that

$$u_{n+1}{}'(t) = 1 \tag{1.14}$$

$$u_{n+1}(t_0) = t_0 \tag{1.15}$$

The system then takes the form

$$u'(t) = f(u(t)) \tag{1.16}$$

with

$$f(u(t), t) = \begin{bmatrix} u_2(t) \\ \vdots \\ u_n(t) \\ F(t, u_1(t), u_2(t), \ldots, u_n(t)) \\ 1 \end{bmatrix} \tag{1.17}$$

$$u(t_0) = \begin{bmatrix} \eta_1 \\ \vdots \\ \eta_n \\ t_0 \end{bmatrix} \tag{1.18}$$

The equation (1.16) is said to be *autonomous* since it does not depend explicitly on time. It is often convenient to assume f is of this form since it simplifies notation.

2 Linear Ordinary Differential Equations

Definition 2.1. The system of ODEs (1.1) is *linear* if

$$f(u, t) = A(t)\, u + g(t) \tag{2.1}$$

where $A(t) \in \mathbb{R}^{s \times s} g(t) \in \mathbb{R}^s$.

An important special case is the *constant coefficient linear system*

$$u'(t) = Au(t) + g(t) \tag{2.2}$$

where $A \in \mathbb{R}^{s \times s}$ is a constant matrix. If $g(t) \equiv 0$, then the equation is *homogeneous*. The solution the homogeneous system $u' = Au$ with data (1.2) is

$$u(t) = e^{A(t-t_0)}\eta \tag{2.3}$$

where the matrix exponential is defined as in [1], Appendix D. In the scalar case we often use λ in place of A.

2.1 Duhamel's Principle

If $g(t)$ is not identically zero, then the solution to the constant coefficient system (2.2) can be written as

$$u(t) = e^{A(t-t_0)}\eta + \int_{t_0}^{t} e^{A(t-\tau)} g(\tau)\, d\tau \tag{2.4}$$

This is known as *Duhamel's principle*. The matrix $e^{A(t-\tau)}$ is the solution operator for the homogeneous problem; it maps data at time η to the solution at time t when solving the homogeneous equation. Duhamel's priciple states that the inhomogeneous term $g(\eta)$ at any instant η has an effect on the solution at time t given by $e^{A(t-\tau)}g(\tau)$. Note that this is very similar to the idea of a Green's function for the boundary value problem (BVP).

As a special case, if $A = 0$, then the ODE is simply

$$u'(t) = g(t) \tag{2.5}$$

and of course the solution (2.4) reduces to the integral of g

$$u(t) = \eta + \int_{t_0}^{t} g(\tau)\, d\tau \tag{2.6}$$

As another special case, suppose A is constant and so is $g(t) \equiv g \in \mathbb{R}^s$. Then (2.4) reduces to

$$u(t) = e^{A(t-t_0)}\eta + \left(\int_{t_0}^{t} e^{A(t-\tau)} d\tau \right) g \tag{2.7}$$

This integral can be computed, e.g., by expressing $e^{A(t-\tau)}$ as a Taylor series and then integrating term by term. This gives

$$\int_{t_0}^{t} e^{A(t-\tau)} d\tau = A^{-1}\left(e^{A(t-t_0)} - I \right) \tag{2.8}$$

and so

$$u(t) = e^{A(t-t_0)}\eta + A^{-1}\left(e^{A(t-t_0)} - I \right) g \tag{2.9}$$

This may be familiar in the scalar case and holds also for constant coefficient systems (provided A is nonsingular). This form of the solution is used explicitly in exponential time differencing methods; see [1], Section 11.6.

3 Lipschitz Continuity

In the last section we considered linear ODEs, for which there is always a unique solution. In most applications, however, we are concerned with nonlinear problems for which there is usually no explicit formula for the solution. The standard theory for the existence of a solution to the initial value problem

$$u'(t) = f(u,t) \tag{3.1}$$
$$u(0) = \eta \tag{3.2}$$

is discussed in may texts. To guarantee that there is a unique solution it is necessary to require a certain amount of smoothness in the function $f(u,t)$ of (3.1)-(3.2). We say that the function $f(u,t)$ is *Lipschitz continuous* in u over some domain

$$\mathcal{D} = \{(u,t) : |u-\eta| \le a, t_0 \le t \le t_1\} \tag{3.3}$$

if there exists some constant $L \ge 0$ so that

$$|f(u,t) - f(u^*,t)| \le L|u-u^*| \tag{3.4}$$

for all (u,t) and (u^*,t) in $\mathcal{D}$. This is slightly stronger than mere continuity, which only requires that

$$|f(u,t) - f(u^*,t)| \to 0 \text{ as } u \to u^* \tag{3.5}$$

Lipschitz continuity requires that

$$|f(u,t) - f(u^*,t)| = O(|u-u^*|) \text{ as } u \to u^* \tag{3.6}$$

If $f(u,t)$ is differentiable with respect to u in $\mathcal{D}$ and this derivative $f_u = \dfrac{\partial f}{\partial u}$ is bounded then we can take

$$L = \max_{(u,t)\in\mathcal{D}} |f_u(u,t)|\, s \tag{3.7}$$

since

$$f(u,t) = f(u^*,t) + f_u(v,t)(u-u^*) \tag{3.8}$$

for some value v between u and u^* by applying Mean Value Theorem for f.

Example 3.1. For the linear problem

$$u'(t) = \lambda u(t) + g(t) \tag{3.9}$$
$$f'(u) \equiv \lambda \tag{3.10}$$

and we can take $L = |\lambda|$. This problem of course has a unique solution for any initial data η given by (2.4) with $A = \lambda$.

In particular, if $\lambda = 0$ then $L = 0$. In this case $f(u,t) = g(t)$ is independent of u. The solution is then obtained by simply integrating the function $g(t)$, as in (2.6).

3.1 Existence and Uniqueness of Solutions

The basic existence and uniqueness theorem states that if f is Lipschitz continuous over some region $\mathcal{D}$ then there is a unique solution to the initial value problem (3.1)-(3.2) at least up to time $T^* = \min\left(t_1, t_0 + \frac{a}{S}\right)$, where

$$S = \max_{(u,t)\in\mathcal{D}} |f(u,t)| \tag{3.11}$$

Note that this is the maximum modulus of the slope that the solution $u(t)$ can attain in this time interval, so that up to time $t_0 + \frac{a}{S}$ we know that $u(t)$ remains in the domain $\mathcal{D}$ where (3.4) holds.

Example 3.2. Consider the initial value problem

$$u'(t) = (u(t))^2 \tag{3.12}$$
$$u(0) = \eta > 0 \tag{3.13}$$

The function $f(u) = u^2$ is independent of t and is Lipschitz continuous in u over any finite interval $|u - \eta| \le a$ with $L = 2(\eta + a)$, and the maximum slope over this interval is $S = (\eta + a)^2$. Indeed,

$$|f(u) - f(u^*)| = \left|u^2 - (u^*)^2\right| \tag{3.14}$$
$$= |u + u^*|\,|u - u^*| \tag{3.15}$$
$$\le 2(\eta + a)\,|u - u^*| \tag{3.16}$$

$$S = \max_{v\in\{u:|u-\eta|\le a\}} |f(v)| \tag{3.17}$$
$$= \max_{v\in\{u:|u-\eta|\le a\}} v^2 \tag{3.18}$$
$$= (\eta + a)^2 \tag{3.19}$$

The theorem guarantees that a unique solution exists at least up to time $\frac{a}{(\eta + a)^2}$. Since a is arbitrary, we can choose a to maximize this expression, which yields $a = \eta$ and so there is a solution at least up to time $\frac{1}{4\eta}$.

In fact this problem can be solved analytically and the unique solution is

$$u(t) = \frac{1}{\frac{1}{\eta} - t} \tag{3.20}$$

Note that $u(t) \to \infty$ as $t \to \frac{1}{\eta}$. There is no solution beyond time $\frac{1}{\eta}$.

If the function f is not Lipschitz continuous in any neighborhood of some point then the initial value problem may fail to have a unique solution over any time interval if this initial value is imposed.

Example 3.3. Consider the initial value problem

$$u'(t) = \sqrt{u(t)} \tag{3.21}$$

with initial condition

$$u(0) = 0 \tag{3.22}$$

The function $f(u) = \sqrt{u}$ is not Lipschitz continuous near $u = 0$ since

$$f'(u) = \frac{1}{2\sqrt{u}} \to \infty \text{ as } u \to 0 \tag{3.23}$$

We cannot find a constant L so that the bound (3.4) holds for all u and u^* near 0.

As a result, this initial value problem does not have a unique solution. In fact it has two distinct solutions

$$u(t) \equiv 0 \tag{3.24}$$

and

$$u(t) = \frac{t^2}{4} \tag{3.25}$$

3.2 Systems of Equations

For systems of $s > 1$ ordinary differential equations, $u(t) \in \mathbb{R}^s$ and $f(u,t)$ is a function mapping $\mathbb{R}^s \times \mathbb{R} \to \mathbb{R}^s$. We say the function f is Lipschitz continuous in u in some norm $\|\cdot\|$ if there is a constant L such that

$$\|f(u,t) - f(u^*,t)\| \leq L\|u - u^*\| \tag{3.26}$$

for all (u,t) and (u^*,t) inn some domain

$$\mathcal{D} = \{(u,t) : \|u - \eta\| \leq a, t_0 \leq t \leq t_1\} \tag{3.27}$$

By the equivalence of finite-dimensional norms, if f is Lipschitz continuous in one norm then it is Lipschitz continuous in any other norm, although the Lipschitz constant may depend on the norm chosen.

The theorems on existence and uniqueness carry over to systems of equations.

Example 3.4. Consider the pendulum problem

$$\theta''(t) = -\sin\theta(t) \tag{3.28}$$

which can be rewritten as a first order system of two equations by introducing $v(t) = \theta'(t)$,

$$u = \begin{bmatrix} \theta \\ v \end{bmatrix} \tag{3.29}$$

$$\frac{d}{dt}\begin{bmatrix} \theta \\ v \end{bmatrix} = \begin{bmatrix} v \\ -\sin\theta \end{bmatrix} \tag{3.30}$$

Consider the max-norm. We have

$$\|u - u^*\|_\infty = \max(|\theta - \theta^*|, |v - v^*|) \tag{3.31}$$

and

$$\|f(u) - f(u^*)\|_\infty = \max\left(|v - v^*|, |\sin\theta - \sin\theta^*|\right) \tag{3.32}$$

To bound $\|f(u) - f(u^*)\|_\infty$, first note that $|v - v^*| \le \|u - u^*\|_\infty$. We also have

$$|\sin\theta - \sin\theta^*| \le |\theta - \theta^*| \le \|u - u^*\|_\infty \tag{3.33}$$

since the derivative of $\sin\theta$ is bounded by 1. So we have Lipschitz continuity with $L = 1$,

$$\|f(u) - f(u^*)\|_\infty \le \|u - u^*\|_\infty \tag{3.34}$$

3.3 Significance of the Lipschitz Constant

The Lipschitz constant measures how much $f(u,t)$ changes if we perturb u (at some fixed time t). Since $f(u,t) = u'(t)$, the slope of the line tangent to the solution curve through the value u, this indicates how the slope of the solution curve will vary if we perturb u. The significance of this is best seen through some examples.

Example 3.5. Consider the trivial equation $u'(t) = g(t)$, which has Lipschitz constant $L = 0$ and solutions given by (2.6). Several solution curves are sketched in Figure 1.

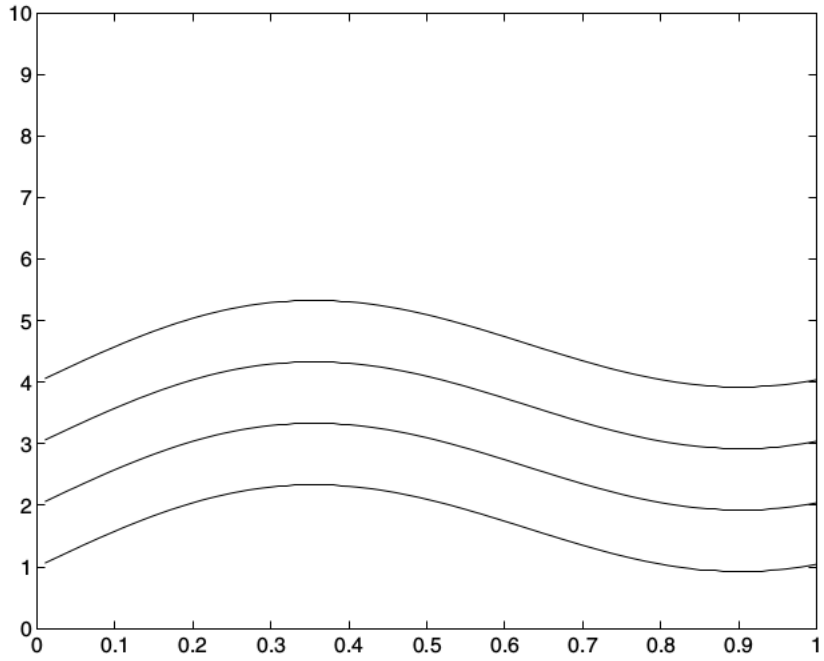

Figure 1: Solution curves for Example 3.5, where $L = 0$.

Note that all these curves are "parallel"; they are simply shifted depending on the initial data. Tangent lines to the curves at any particular time are all

parallel since $f(u,t) = g(t)$ is independent of u.

Example 3.6. Consider $u'(t) = \lambda u(t)$ with λ constant and $L = |\lambda|$. Then $u(t) = u(0)e^{\lambda t}$. Two situations are shown in Figure 2 and Figure 3 for negative and positive values of λ.

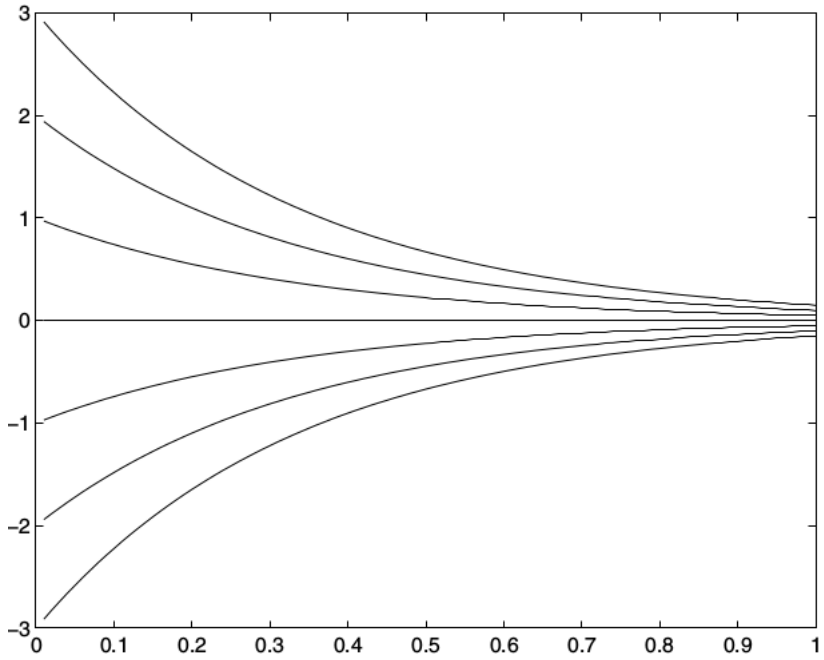

Figure 2: Solution curves for Example 3.6, with $\lambda = -3$.

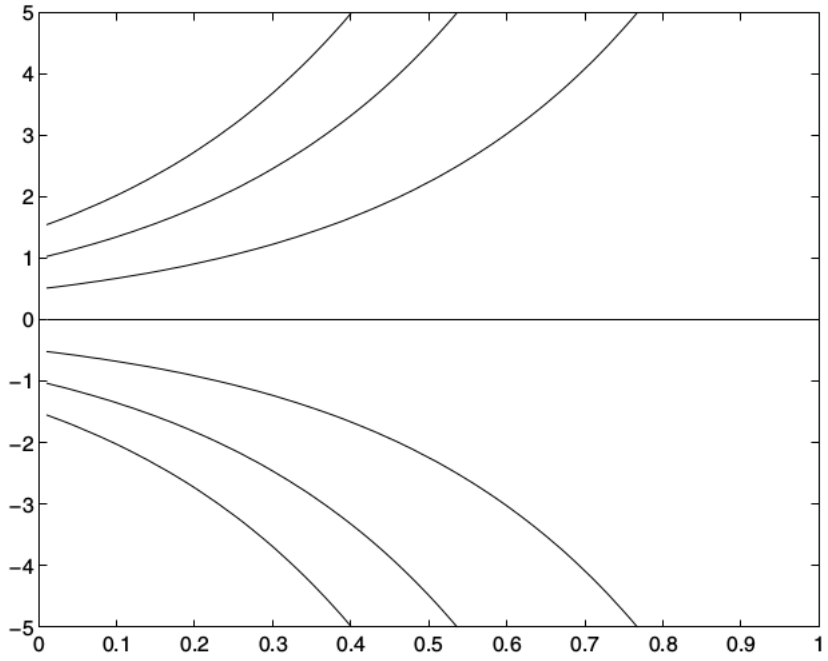

Figure 3: Solution curves for Example 3.6, with $\lambda = 3$.

Here the slope of the solution curve does vary depending on u. The variation in the slope with u (at fixed t) gives an indication of how rapidly the solution curves are converging toward one another (in the case $\lambda < 0$) or diverging away from one another (in the case $\lambda > 0$). If the magnitude of λ were increased, convergence or divergence would clearly be more rapid.

The size of the Lipschitz constant is significant if we intend to solve the problem numerically since our numerical approximation will almost certainly produce a value U^n at time t_n that is not exactly equal to the true value $u(t_n)$. Hence we are on a different solution curve than the true solution. The best we can hope for in the future is that we stay close to the solution curve that we are now on. The size of the Lipschitz constant gives an indication of whether solution curves that start close together can be expected to stay close together or might diverge rapidly.

3.4 Limitations

Actually, the Lipschitz constant is not the perfect tool for this purpose, since it does not distinguish between rapid divergence and rapid convergence of solution curves. In both Figure 2 and Figure 3 the Lipschitz constant has the same value $L = |\lambda| = 3$. But we would expect that rapidly convergent solution curves as in Figure 2 should be easier to handle numerically than rapidly divergent ones. If we make an error at some stage, then the effect of this error should decay at later times rather than growing. To some extent this is true and as a result error bounds based on the Lipschitz constant may be orders of magnitude too large in this situation.

However, rapidly converging solution curves can also give serious numerical difficulties, which one might not expect at first glance. This is discussed in detail in [1], Chapter 8, which covers stiff equations.

One should also keep in mind that a small value of the Lipschitz constant does not necessarily mean that two solution curves starting close together will stay close together forever.

Example 3.7. Consider two solutions to the pendulum problem from Example 3.4, one with initial data

$$\theta_1(0) = \pi - \varepsilon \tag{3.35}$$
$$v_1(0) = 0 \tag{3.36}$$

and the other with

$$\theta_2(0) = \pi + \varepsilon \tag{3.37}$$
$$v_2(0) = 0 \tag{3.38}$$

The Lipschitz constant is 1 and the data differ by 2ϵ, which can be arbitrarily small, and yet the solutions eventually diverge dramatically, as Solution 1 falls toward $\theta = 0$, while in Solution 2 the pendulum falls the other way, toward $\theta = 2\pi$.

In this case the IVP is very ill conditioned: small changes in the data can lead to order 1 changes in the solution. As always in numerical analysis, the solution of ill-conditioned problems can be very hard to compute accurately.

4 Some Basic Numerical Methods

We begin by listing a few standard approaches to discretizing (1.1). Note that the IVP differs from the BVP considered before in that we are given all the data at the initial time $t_0 = 0$ and from this we should be able to march forward in time, computing approximations at successive times $t_1, t_2, \ldots$. We will use k to denote the time step, so $t_n = nk$ for $n \geq 0$. It is convenient to use the symbol k, which is different from the spatial grid size h, since we will soon study PDEs which involve both spatial and temporal discretizations. Often the symbols Δt and Δx are used.

We are given initial data

$$U^0 = \eta \tag{4.1}$$

and want to compute approximations $U^1, U^2, \ldots$ satisfying

$$U^n \approx u(t_n) \tag{4.2}$$

We will use superscripts to denote the time step index, again anticipating the notation of PDEs where we will use subscripts for spatial indices.

The simplest method is *Euler's method* (also called *forward Euler*), based on replacing $u'(t_n)$ with

$$D_+ U^n = \frac{U^{n+1} - U^n}{k} \tag{4.3}$$

This gives the method

$$\frac{U^{n+1} - U^n}{k} = f\left(U^n\right), \quad n = 0, 1, \ldots \tag{4.4}$$

Rather than viewing this as a system of simultaneous equations as we did for the BVP, it is possible to solve this explicitly for U^{n+1} in terms of U^n,

$$U^{n+1} = U^n + kf\left(U^n\right) \tag{4.5}$$

From the initial data U^0 we can compute U^1, then U^2, and so on. This is called a *time-marching method*.

The *backward Euler* method is similar but is based on replacing $u'\left(t_{n+1}\right)$ with $D_- U^{n+1}$,

$$\frac{U^{n+1} - U^n}{k} = f\left(U^{n+1}\right) \tag{4.6}$$

or

$$U^{n+1} = U^n + kf\left(U^{n+1}\right) \tag{4.7}$$

Again we can march forward in time since computing U^{n+1} requires only that we know the previous value U^n. In the backward Euler method, however, (4.7) is an equation that must be solved for U^{n+1}, and in general $f(u)$ is a nonlinear function. We can view this as looking for a zero of the function

$$g(u) = u - kf(u) - U^n \tag{4.8}$$

which can be approximated using some iterative method such as *Newton's method*.

Because the backward Euler method gives an equation that must be solved for U^{n+1}, it is called an *implicit* method, whereas the forward Euler method (4.5) is an *explicit* method.

Another implicit method is the *trapezoidal method*, obtained by averaging the two Euler methods

$$\frac{U^{n+1} - U^n}{k} = \frac{1}{2}\left(f\left(U^n\right) + f\left(U^{n+1}\right)\right) \tag{4.9}$$

As one might expect, this symmetric approximation is second order accurate, whereas the Euler methods are only first order accurate.

The above methods are all *one-step methods*, meaning that U^{n+1} is determined from U^n alone and previous values of U are not needed. One way to get higher order accuracy is to use a *multistep* method that involves other previous values. For example, using the approximation

$$\frac{u(t+k) - u(t-k)}{2k} = u'(t) + \frac{1}{6}k^2 u'''(t) + O\left(k^3\right) \tag{4.10}$$

yields the *midpoint method* (also called the *leapfrog method*),

$$\frac{U^{n+1} - U^{n-1}}{2k} = f\left(U^n\right) \tag{4.11}$$

or

$$U^{n+1} = U^{n-1} + 2kf\left(U^n\right) \tag{4.12}$$

which is a second order accurate explicit 2-step method. The approximation $D_2 u$ rewritten in the form

$$\frac{3u\left(t+k\right) - 4u\left(t\right) + u\left(t-k\right)}{2k} = u'\left(t+k\right) + \frac{1}{12}k^2 u'''\left(t+k\right) + O\left(k^4\right) \tag{4.13}$$

yields a second order implicit 2-step method

$$\frac{3U^{n+1} - 4U^n + U^{n-1}}{2k} = f\left(U^{n+1}\right) \tag{4.14}$$

This is one of the backward differentiation formula (BDF) methods that will be discussed further in [1], Chapter 8.

5 Truncation Errors

The truncation error for these methods is defined in the same way as in [1], Chapter 2. We write the difference equation in the form that directly models the derivatives (e.g., in the form (4.11) rather than (4.12)) and then insert the true solution to the ODE into the difference equation. We then use Taylor series expansion and cancel out common terms.

Example 5.1. The local truncation error (LTE) of the midpoint method (4.11) is defined by

$$\tau^n = \frac{u\left(t_{n+1}\right) - u\left(t_{n-1}\right)}{2k} - f\left(u\left(t_n\right)\right) \tag{5.1}$$

$$= \left[u'\left(t_n\right) + \frac{1}{6}k^2 u'''\left(t_n\right) + O\left(k^4\right)\right] - u'\left(t_n\right) \tag{5.2}$$

$$= \frac{1}{6}k^2 u'''\left(t_n\right) + O\left(k^4\right) \tag{5.3}$$

Note that since $u\left(t\right)$ is the true solution of the ODE, $u'\left(t_n\right) = f\left(u\left(t_n\right)\right)$. The $O\left(k^3\right)$ term drops out by symmetry. The truncation error is $O\left(k^2\right)$ and so we say the method is *second order accurate*, although it is not yet clear that the global error will have this behavior. As always, we need some form of *stability* to guarantee that the global error will exhibit the same rate of convergence as the local truncation error. This will be discussed below.

6 One-Step Errors

In much of the literature concerning numerical methods for ODEs, a slightly different definition of the local truncation error is used that is based on the form (4.12), for example, rather than (4.11). Denoting this value by $\mathcal{L}^n$, we have

$$\mathcal{L}^n = u\left(t_{n+1}\right) - u\left(t_{n-1}\right) - 2kf\left(u\left(t_n\right)\right) \tag{6.1}$$

$$= \frac{1}{3}k^3 u'''(t_n) + O\left(k^5\right) \tag{6.2}$$

Since $\mathcal{L}^n = 2k\tau^n$, this local error is $O\left(k^3\right)$ rather than $O\left(k^2\right)$, but of course the global error remains the same and will be $O\left(k^2\right)$. Using this alternative definition, many standard results in ODE theory say that a pth order accurate method should have and LTE that is $O\left(k^{p+1}\right)$. With the notation we are using, a pth order accurate method has an LTE that is $O\left(k^p\right)$. The notation used here is consistent with the standard practice for PDEs and leads to a more coherent theory, but one should be aware of this possible source of confusion.

In the book [1] $\mathcal{L}^n$ will be called the *one-step error*, since this can be viewed as the error that would be introduced in one time step if the past values $U^n, U^{n-1}, \ldots$ were all taken to be the exact values from $u(t)$. For example, in the midpoint method (4.12) we suppose that

$$U^n = u(t_n) \tag{6.3}$$

$$U^{n-1} = u(t_{n-1}) \tag{6.4}$$

and we now use these values to compute U^{n+1}, an approximation to $u(t_{n+1})$,

$$U^{n+1} = u(t_{n-1}) + 2kf(u(t_n)) \tag{6.5}$$

$$= u(t_{n-1}) + 2ku'(t_n) \tag{6.6}$$

Then the error is

$$u(t_{n+1}) - U^{n+1} = u(t_{n+1}) - u(t_{n-1}) - 2ku'(t_n) \tag{6.7}$$

$$= \mathcal{L}^n \tag{6.8}$$

From (6.1) we see that in one step the error introduced is $O\left(k^3\right)$. This is consistent with second order accuracy in the global error if we think of trying to compute an approximation to the true solution $u(T)$ at some fixed time $T > 0$. To compute from time $t = 0$ up to time T, we need to take $\frac{T}{k}$ time steps of length k. A rough estimate of the error at time T might be obtained by assuming that a new error of size $\mathcal{L}^n$ is introduced in the nth time step and is the n simply carried along in later time steps without affecting the size of future local errors and without growing or diminishing itself. Then we would expect the resulting global error at time T to be simply the sum of all these local errors. Since each local error is $O\left(k^3\right)$ and we are adding up $\frac{T}{k}$ of them, we end up with a global error that is $O\left(k^2\right)$.

This viewpoint is in fact exactly right for the simplest ODE (2.5), in which $f(u,t) = g(t)$ is independent of u and the solution is simply the integral of g, but it is a bit too simplistic for more interesting equations since the error at each time feeds back into the computation at the next step in the case where $f(u,t)$ depends on u. Nonetheless, it is essentially right in terms of the expected order of accuracy, provided the method is stable. In fact, it is useful to think of *stability* as exactly what is needed to make this naive analysis correct, by ensuring that the old errors from previous time steps do not grow too rapidly in future time steps. This will be investigated in detail in the following chapters, [1].

7 Taylor Series Methods

The forward Euler method (4.5) can be derived using a Taylor series expansion of $u(t_{n+1})$ about $u(t_n)$,

$$u(t_{n+1}) = u(t_n) + ku'(t_n) + \frac{1}{2}k^2u''(t_n) + O\left(k^3\right) \tag{7.1}$$

If we drop all terms of order k^2 and higher and use the differential equation to replace $u'(t_n)$ with $f(u(t_n), t_n)$, we obtain

$$u(t_{n+1}) \approx u(t_n) + kf(u(t_n), t_n) \tag{7.2}$$

This suggests the method (4.5). The 1-step error is $O\left(k^2\right)$ since we dropped terms of this order.

A *Taylor series method* of higher accuracy can be derived by keeping more terms in the Taylor series. If we keep the first $p+1$ terms of the Taylor series expansion

$$u(t_{n+1}) \approx \sum_{j=0}^{p} \frac{1}{j!}k^j u^{(j)}(t_n) \tag{7.3}$$

we obtain a pth order accurate method. The problem is that we are given only

$$u'(t) = f(u(t), t) \tag{7.4}$$

and we must compute the higher derivatives by repeated differentiation of this function. For example, we can compute

$$u''(t) = f_u(u(t), t)\, u'(t) + f_t(u(t), t) \tag{7.5}$$

$$= f_u(u(t), t)\, f(u(t), t) + f_t(u(t), t) \tag{7.6}$$

This can result in very messy expressions that must be worked out for each equation, and as a result this approach is not often used in practice. However, it is such an obvious approach that it is worth mentioning, and in some cases it may be useful. An example should suffice to illustrate the technique and its limitations.

Example 7.1. Suppose we want to solve the equation

$$u'(t) = t^2 \sin(u(t)) \tag{7.7}$$

Then we can compute

$$u''(t) = 2t \sin(u(t)) + t^2 \cos(u(t))\, u'(t) \tag{7.8}$$

$$= 2t \sin(u(t)) + t^4 \cos(u(t)) \sin(u(t)) \tag{7.9}$$

A second order method is given by

$$U^{n+1} = U^n + kt_n^2 \sin(U^n) + \frac{1}{2}k^2 \left[2t_n \sin(U^n) + t_n^4 \cos(U^n) \sin(U^n)\right] \tag{7.10}$$

Clearly higher order derivatives can be computed and used, but this is cumbersome even for this simple example. For systems of equations the method becomes still more complicated.

This Taylor series approach does get used in some situations, however - for example, in the derivation of the Lax-Wendroff method for hyperbolic PDEs; see [1] Section 10.3. See also [1] Section 11.3.

8 Runge-Kutta Methods

Most methods used in practice do not require that the user explicitly calculate higher order derivatives. Instead a higher order finite difference approximation is designed that typically models these terms automatically.

A multistep method of the sort we will study in [1] Section 5.9 can achieve high accuracy by using high order polynomial interpolation through several previous values of the solution and/or its derivatives. To achieve the same effect with a 1-step method it is typically necessary to use a *multistage* method, where intermediate values of the solution and its derivative are generated and used within a single time step.

Example 8.1. A two-stage explicit Runge-Kutta method is given by

$$U^* = U^n + \frac{1}{2}kf\left(U^n\right) \tag{8.1}$$

$$U^{n+1} = U^n + kf\left(U^*\right) \tag{8.2}$$

In the first stage an intermediate value is generated that approximates $u\begin{pmatrix} t \\ n+\frac{1}{2} \end{pmatrix}$ via Euler's method. In the second step the function f is evaluated at this midpoint to estimate the slope over the full time step. Since this now looks like a centered approximation to the derivative we might hope for second order accuracy, as we will now verify by computing the LTE.

Combining the two steps above, we can rewrite the method as

$$U^{n+1} = U^n + kf\left(U^n + \frac{1}{2}kf\left(U^n\right)\right) \tag{8.3}$$

Viewed this way, this is clearly a 1-step explicit method. The truncation error is

$$\tau^n = \frac{1}{k}\left(u\left(t_{n+1}\right) - u\left(t_n\right)\right) - f\left(u\left(t_n\right) + \frac{1}{2}kf\left(u\left(t_n\right)\right)\right) \tag{8.4}$$

Note that

$$f\left(u\left(t_n\right) + \frac{1}{2}kf\left(u\left(t_n\right)\right)\right) \tag{8.5}$$

$$= f\left(u\left(t_n\right) + \frac{1}{2}ku'\left(t_n\right)\right) \tag{8.6}$$

$$= f\left(u\left(t_n\right)\right) + \frac{1}{2}ku'\left(t_n\right)f'\left(u\left(t_n\right)\right) + \frac{1}{8}k^2\left(u'\left(t_n\right)\right)^2 f''\left(u\left(t_n\right)\right) + O\left(k^3\right) \tag{8.7}$$

Since $f\left(u\left(t_n\right)\right) = u'\left(t_n\right)$ and differentiating gives $f'\left(u\right)u' = u''$, we obtain

$$f\left(u\left(t_n\right) + \frac{1}{2}kf\left(u\left(t_n\right)\right)\right) = u'\left(t_n\right) + \frac{1}{2}ku''\left(t_n\right) + O\left(k^2\right) \tag{8.8}$$

Using this in (8.4) gives

$$\tau^n = \frac{1}{k}\left(ku'\left(t_n\right) + \frac{1}{2}k^2u''\left(t_n\right) + O\left(k^3\right)\right) \tag{8.9}$$

$$- \left(u'(t_n) + \frac{1}{2} k u''(t_n) + O\left(k^2\right) \right) \tag{8.10}$$

$$= O\left(k^2\right) \tag{8.11}$$

and the method is second order accurate. Check the $O\left(k^2\right)$ term to see that this does not vanish. Indeed, we use Taylor series expansion $O\left(k^4\right)$,

$$f\left(u(t_n) + \frac{1}{2} k f(u(t_n)) \right) \tag{8.12}$$

$$= f\left(u(t_n) + \frac{1}{2} k u'(t_n) \right) \tag{8.13}$$

$$= f(u(t_n)) + \frac{1}{2} k u'(t_n) f'(u(t_n)) + \frac{1}{8} k^2 (u'(t_n))^2 f''(u(t_n)) \tag{8.14}$$

$$+ \frac{1}{48} k^3 (u'(t_n))^3 f'''(u(t_n)) + O\left(k^4\right) \tag{8.15}$$

and $f''(u) u' + f'(u) u'' = u'''$ gives

$$f''(u)(u')^2 = u'(u''' - f'(u) u'') \tag{8.16}$$

$$= u' u''' - f'(u) u' u'' \tag{8.17}$$

$$= u' u''' - (u'')^2 \tag{8.18}$$

Inserting (8.16)-(8.18) into (8.12)-(8.15) yields

$$f\left(u(t_n) + \frac{1}{2} k f(u(t_n)) \right) = u'(t_n) + \frac{1}{2} k u''(t_n) \tag{8.19}$$

$$+ \frac{1}{8} k^2 \left(u'(t_n) u'''(t_n) - (u''(t_n))^2 \right) + O\left(k^3\right) \tag{8.20}$$

Using this in (8.4) again gives

$$\tau^n \tag{8.21}$$

$$= \frac{1}{k} \left(k u'(t_n) + \frac{1}{2} k^2 u''(t_n) + \frac{1}{6} k^3 u'''(t_n) + O\left(k^4\right) \right) \tag{8.22}$$

$$- \left(u'(t_n) + \frac{1}{2} k u''(t_n) + \frac{1}{8} k^2 \left(u'(t_n) u'''(t_n) - (u''(t_n))^2 \right) + O\left(k^3\right) \right) \tag{8.23}$$

$$= k^2 \left(\frac{1}{6} u'''(t_n) - \frac{1}{8} u'(t_n) u'''(t_n) + \frac{1}{8} (u''(t_n))^2 \right) + O\left(k^3\right) \tag{8.24}$$

Remark 8.2. Another way to determine the order of accuracy of this simple method is to apply it to the special test equation $u' = \lambda u$, which has solution $u(t_{n+1}) = e^{\lambda k} u(t_n)$, and determine the error on this problem. Here we obtain

$$U^{n+1} = U^n + k\lambda \left(U^n + \frac{1}{2} k \lambda U^n \right) \tag{8.25}$$

$$= U^n + (k\lambda) U^n + \frac{1}{2} (k\lambda)^2 U^n \tag{8.26}$$

$$= e^{k\lambda} U^n + O\left(k^3\right) \tag{8.27}$$

The one-step error is $O\left(k^3\right)$ and hence the LTE is $PO\left(k^2\right)$. Of course we have checked only that the LTE is $O\left(k^2\right)$ on one particular function $u\left(t\right) = e^{\lambda t}$, not on all smooth solutions, and for general Runge-Kutta methods for nonautonomous problems this approach gives only an upper bound on the method's order of accuracy. Applying a method to this special equation is also a fundamental tool in stability analysis - see [1] Chapter 7.

Example 8.3. The Runge-Kutta method (8.1)-(8.2) can be extended to nonautonomous equations of the form $u'\left(t\right) = f\left(u\left(t\right), t\right)$,

$$U^* = U^n + \frac{1}{2}kf\left(U^n, t_n\right) \tag{8.28}$$

$$U^{n+1} = U^n + kf\left(U^*, t_n + \frac{k}{2}\right) \tag{8.29}$$

This is again second order accurate, as can be verified by expanding as above, but it is slightly more complicated since Taylor series in two variables must be used.

Example 8.4. One simple higher order Runge-Kutta method is the fourth order four-stage method given by

$$Y_1 = U^n \tag{8.30}$$

$$Y_2 = U^n + \frac{1}{2}kf\left(Y_1, t_n\right) \tag{8.31}$$

$$Y_3 = U^n + \frac{1}{2}kf\left(Y_2, t_n + \frac{k}{2}\right) \tag{8.32}$$

$$Y_4 = U^n + kf\left(Y_3, t_n + \frac{k}{2}\right) \tag{8.33}$$

$$U^{n+1} = U^n + \frac{k}{6}\left[\begin{array}{l} f\left(Y_1, t_n\right) + 2f\left(Y_2, t_n + \frac{k}{2}\right) \\ +2f\left(Y_3, t_n + \frac{k}{2}\right) + f\left(Y_4, t_n + k\right) \end{array}\right] \tag{8.34}$$

Note that if $f\left(u, t\right) = f\left(t\right)$ does not depend on u, then this reduces to Simpson's rule for the integral. This method was particularly popular in the precomputer era, when computations were done by hand, because the coefficients are so simple. Today there is no need to keep the coefficients simple and other Runge-Kutta methods have advantages.

A general r-stage Runge-Kutta method has the form

$$Y_i = U^n + k\sum_{j=1}^{r} a_{ij} f\left(Y_j, t_n + c_j k\right), \quad i = 1, 2, \ldots, r \tag{8.35}$$

$$U^{n+1} = U^n + k\sum_{j=1}^{r} b_j f\left(Y_j, t_n + c_j k\right) \tag{8.36}$$

Consistency requires

$$\sum_{j=1}^{r} a_{ij} = c_i, \quad i = 1, 2, \ldots, r \tag{8.37}$$

$$\sum_{j=1}^{r} b_j = 1 \tag{8.38}$$

If these conditions are satisfied, then the method will be at least first order accurate.

The coefficients for a Runge-Kutta method are often displayed in a so-called Butcher tableau,

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1r} \\ \vdots & \vdots & \ddots & \vdots \\ c_r & a_{r1} & \cdots & a_{rr} \\ \hline & b_1 & \cdots & b_r \end{array} \tag{8.39}$$

For example, the fourth order Runge-Kutta method given in (8.30)-(8.34) has the following tableau (entries not shown are all 0),

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{8.40}$$

An important class of Runge-Kutta methods consists of the *explicit methods* for which $a_{ij} = 0$ for $j \geq i$. For an explicit method, the elements on and above the diagonal in the a_{ij} portion of the Butcher tableau are all equal to zero, as, for example, with the fourth order method displayed above. With an explicit method, each of the Y_i values is computed using only the previously computed Y_j.

Fully implicit Runge-Kutta methods, in which each Y_i depends on all the Y_j, can be expensive to implement on systems of ODEs. For a system of s equations (where each Y_i is in $\mathbb{R}^s$), a system of sr equations must be solved to compute the r vectors Y_i simultaneously.

One subclass of implicit methods that are simpler to implement are the *diagonally implicit* Runge-Kutta methods (DIRK methods) in which Y_i depends on Y_j for $j \leq i$, i.e., $a_{ij} = 0$ for $j > i$. For a system of s equations, DIRK methods require solving a sequence of r implicit systems, each of size s, rather than a coupled set of sr equation as would be required in a fully implicit Runge-Kutta method. DIRK methods are so named because their tableau has zero values above the diagonal but possibly nonzero diagonal elements.

Example 8.5. A second order accurate DIRK method is given by

$$Y_1 = U^n \tag{8.41}$$

$$Y_2 = U^n + \frac{k}{4}\left[f\left(Y_1, t_n\right) + f\left(Y_2, t_n + \frac{k}{2}\right)\right] \tag{8.42}$$

$$Y_3 = U^n + \frac{k}{3}\left[f\left(Y_1, t_n\right) + f\left(Y_2, t_n + \frac{k}{2}\right) + f\left(Y_3, t_n + k\right)\right] \tag{8.43}$$

$$U^{n+1} = Y_3 \tag{8.44}$$

$$= U^n + \frac{k}{3}\left[f\left(Y_1, t_n\right) + f\left(Y_2, t_n + \frac{k}{2}\right) + f\left(Y_3, t_n + k\right)\right] \tag{8.45}$$

This method is known as the TR-BDF2 method and is derived in a different form in [1] Section 8.5. Its tableau is

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ 1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \hline & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{array} \tag{8.46}$$

In addition to the conditions (8.37)-(8.38), a Runge-Kutta method is second order accurate if

$$\sum_{j=1}^{r} b_j c_j = \frac{1}{2} \tag{8.47}$$

as is satisfied for the method (8.41)-(8.45). Third order accuracy requires two additional conditions

$$\sum_{j=1}^{r} b_j c_j^2 = \frac{1}{3} \tag{8.48}$$

$$\sum_{i=1}^{r}\sum_{j=1}^{r} b_i a_{ij} c_j = \frac{1}{6} \tag{8.49}$$

Fourth order accuracy requires an additional four conditions on the coefficients, and higher order methods require an exponentially growing number of conditions.

An r-stage explicit Runge-Kutta method can have order at most r, although for $r \geq 5$ the order is strictly less than the number of stages. Among implicit Runge-Kutta methods, r-stage methods of order $2r$ exist. There typically are many ways that the coefficients a_{ij} and b_j can be chosen to achieve a given accuracy, provided the number of stages is sufficiently large. Many different classes of Runge-Kutta methods have been developed over the years with various advantages. The order conditions are quite complicated for higher-order methods and an extensive theory has been developed by Butcher for analyzing these methods and their stability properties.

Using more stages to increase the order of a methjod is an obvious strategy. For some problems, however, we will also see that it can be advantageous to use a large number of stages to increase the *stability* of the method while keeping the order of accuracy relatively low. This is the idea behind the *Runge-Kutta-Chebyshev methods*, for example, discussed in [1] Section 8.6.

8.1 Embedded Methods and Error Estimation

Most practical software for solving ODEs does not use a fixed time step but rather adjusts the time step during the integration process to try to achieve some specified error bound. One common way to estimate the error in the

computation is to compute using two different methods and compare the results. Knowing something about the error behavior of each method often allows the possibility of estimating the error in at least one of the two results.

A simple way to do this for ODEs is to take a time step with two different methods, one of order p and one of a different order, say, $p+1$. Assuming that the time step is small enough that the higher order method is really generating a better approximation, then the difference between the two results will be an estimate of the one-step error in the lower order method. This can be used as the basis for choosing an appropriate time step for the lower order approximation. Often the time step is chosen in this manner, but then the higher order solution is used as the actual approximation at this time and as the starting point for the next time step. This is sometimes called *local extrapolation*. Once this is done there is no estimate of the error, but presumably it is even smaller than the error in the lower order method and so the approximation generated will be even better than the required tolerance.

Note, however, that the procedure of using two different methods in every time step could easily double the cost of the computation unless we choose the methods carefully. Since the main cost in a Runge-Kutta method is often in evaluating the function $f(u,t)$, it makes sense to reuse function values as much as possible and look for methods that provide two approximations to U^{n+1} of different order based on the same set of function evaluations, by simply taking different linear combinations of the $f(Y_j, t_n + c_j k)$ values in the final stage of the Runge-Kutta method (8.35)-(8.36). So in addition to the value U^{n+1} given there we would like to also compute a value

$$\hat{U}^{n+1} = U^n + k\sum_{j=1}^{r} \hat{b}_j f(Y_j, t_n + c_j k) \tag{8.50}$$

that gives an approximation of a different order than can be used for error estimation. These are called *embedded Runge-Kutta methods* and are often displayed in a tableau of the form

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1r} \\ \vdots & \vdots & & \vdots \\ c_r & a_{r1} & \cdots & a_{rr} \\ \hline & b_1 & \cdots & b_r \\ \hline & \hat{b}_1 & \cdots & \hat{b}_r \end{array} \tag{8.51}$$

As a very simple example, the second order Runge-Kutta method (8.28)-(8.29) could be combined with the first order Euler method,

$$Y_1 = U^n \tag{8.52}$$

$$Y_2 = U^n + \frac{1}{2}kf(Y_1, t_n) \tag{8.53}$$

$$U^{n+1} = U^n + kf\left(Y_2, t_n + \frac{k}{2}\right) \tag{8.54}$$

$$\hat{U}^{n+1} = U^n + kf(Y_1, t_n) \tag{8.55}$$

Note that the computation of $\hat{U}^{n+1}$ reuses the value $f(Y_1, t_n)$ obtained in com-

puting Y_2 and is essentially free. Also note that

$$\hat{U}^{n+1} - U^{n+1} = k\left(f\left(Y_1, t_n\right) - f\left(Y_2, t_n + \frac{k}{2}\right)\right) \tag{8.56}$$

$$\approx k\left(u'\left(t_n\right) - u'\left(t_{n+\frac{1}{2}}\right)\right) \tag{8.57}$$

$$\approx \frac{1}{2}k^2 u''\left(t_n\right) \tag{8.58}$$

which is approximately the one-step error for Euler's method.

Most software based on Runge-Kutta methods uses embedded methods of higher order. For example, the `ode45` routine in Matlab uses a pair of embedded Runge-Kutta methods of order 4 and 5 due to Dormand and Prince.

9 One-Step versus Multistep Methods

Taylor series and Runge-Kutta methods are *one-step methods*; the approximation U^{n+1} depends on U^n but not on previous values $U^{n-1}, U^{n-2}, \ldots$. In the next section we will consider a class of multistep methods where previous values are also used (one example is the midpoint method (4.12)).

One-step methods have several advantages over multistep methods:

- The methods are *self-starting:* from the initial data U^0 the desired method can be applied immediately. Multistep methods require that some other method be used initially, as discussed in [1] Section 5.9.3.
- The time step k can be changed at any point, based on an error estimate, for example. The time step can also be changed with a multistep method but more care is required since the previous values are assumed to be equally spaced in the standard form of these methods given below.
- If the solution $u\left(t\right)$ is not smooth at some isolated points t^* (for example, because $f\left(u, t\right)$ is discontinuous at t^*), then with a one-step method it is often possible to get full accuracy simply by ensuring that t^* is a grid point. With a multistep method that uses data from both sides of t^* in approximating derivatives, a loss of accuracy may occur.

On the other hand, one-step methods have some disadvantages. The disadvantages of Taylor series methods is that they require differentiating the given equation and are cumbersome and often expensive to implement. Runge-Kutta methods only use evaluations of the function f, but a higher order multistage method requires evaluating f several times each time step. For simple equations this may not be a problem, but if function values are expensive to compute, then high order Runge-Kutta methods may be quite expensive as well. This is particularly true for implicit methods, where an implicit nonlinear system must be solved in each stage.

An alternative is to use a multistep method in which values of f already computed in previous time steps are reused to obtain higher order accuracy. Typically only one new f evaluation is required in each time step. The popular class of *linear multistep methods* is discussed in the next section.

10 Linear Multistep Methods

All the methods introduced in [1] Section 5.3 are members of a class of methods called linear multistep methods (LMMs). In general, an r-step LMM has the form

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r} \beta_j f\left(U^{n+j}, t_{n+j}\right) \tag{10.1}$$

The value U^{n+r} is computed from this equation in terms of the previous values $U^{n+r-1}, U^{n+r-2}, \ldots, U^n$ and f values at these points (which can be stored and reused if f is expensive to evaluate).

If $\beta_r = 0$, then the method (10.1) is explicit; otherwise it is implicit. Note that we can multiply both sides by any constant and have essentially the same method, although the coefficients α_j and β_j would change. The normalization $\alpha_r = 1$ is often assumed to fix this scale factor.

There are special classes of methods of this form that are particularly useful and have distinctive names. These will be written out for the autonomous case where $f(u,t) = f(u)$ to simplify the formulas, but each can be used more generally by replacing $f\left(U^{n+j}\right)$ with $f\left(U^{n+j}, t_{n+j}\right)$ in any of the formulas.

Example 10.1. The *Adams methods* have the form

$$U^{n+r} = U^{n+r-1} + k \sum_{j=0}^{r} \beta_j f\left(U^{n+j}\right) \tag{10.2}$$

These methods all have

$$\alpha_r = 1 \tag{10.3}$$

$$\alpha_{r-1} = -1 \tag{10.4}$$

$$\alpha_j = 0 \text{ for } j < r-1 \tag{10.5}$$

The β_j coefficients are chosen to maximize the order of accuracy. If we require $\beta_r = 0$ so the method is explicit, then the r coefficients $\beta_0, \beta_1, \ldots, \beta_{r-1}$ can be chosen so that the method has order r. This can be done by using Taylor series expansion of the local truncation error and then choosing the β_j to eliminate as many terms as possible. This gives the explicit *Adams-Bashforth methods*.

Another way to derive the Adams-Bashforth methods is by writing

$$u\left(t_{n+r}\right) = u\left(t_{n+r-1}\right) + \int_{t_{n+r-1}}^{t_{n+r}} u'(t)\, dt \tag{10.6}$$

$$= u\left(t_{n+r-1}\right) + \int_{t_{n+r-1}}^{t_{n+r}} f\left(u(t)\right) dt \tag{10.7}$$

and then applying a quadrature rule to this integral to approximate

$$\int_{t_{n+r-1}}^{t_{n+r}} f\left(u(t)\right) dt \approx k \sum_{j=1}^{r-1} \beta_j f\left(u\left(t_{n+j}\right)\right) \tag{10.8}$$

This quadrature rule can be derived by interpolating $f(u(t))$ by a polynomial $p(t)$ of degree $r-1$ at the points $t_n, t_{n+1}, \ldots, t_{n+r-1}$ and then integrating the interpolating polynomial.

Either approach gives the same r-step explicit method. The first few are given below.

Explicit Adams-Bashforth methods.

- **1-step (Euler).**

$$U^{n+1} = U^n + kf(U^n) \tag{10.9}$$

- **2-step.**

$$U^{n+2} = U^{n+1} + \frac{k}{2}\left(-f(U^n) + 3f\left(U^{n+1}\right)\right) \tag{10.10}$$

- **3-step.**

$$U^{n+3} = U^{n+2} + \frac{k}{12}\left(5f(U^n) - 16f\left(U^{n+1}\right) + 23f\left(U^{n+2}\right)\right) \tag{10.11}$$

- **4-step.**

$$U^{n+4} = U^{n+3} + \frac{k}{720}\left(-9f(U^n) + 37f\left(U^{n+1}\right) - 59f\left(U^{n+2}\right) + 55f\left(U^{n+3}\right)\right) \tag{10.12}$$

If we allow β_r to be nonzero, then we have one more free parameter and so we can eliminate an additional term in the LTE. This gives an implicit method of order $r+1$ called the r-step *Adams-Moulton*. These methods can again be derived by polynomial interpolation, now using a polynomial $p(t)$ of degree r that interpolates $f(u(t))$ at the points $t_n, t_{n+1}, \ldots, t_{n+r}$ and then integrating the interpolating polynomial.

Implicit Adams-Moulton methods

- **1-step (trapezoidal method).**

$$U^{n+1} = U^n + \frac{k}{2}\left(f(U^n) + f\left(U^{n+1}\right)\right) \tag{10.13}$$

- **2-step.**

$$U^{n+2} = U^{n+1} + \frac{k}{12}\left(-f(U^n) + 8f\left(U^{n+1}\right) + 5f\left(U^{n+2}\right)\right) \tag{10.14}$$

- **3-step.**

$$U^{n+3} = U^{n+2} + \frac{k}{24}\left(f(U^n) - 5f\left(U^{n+1}\right) + 19f\left(U^{n+2}\right) + 9f\left(U^{n+3}\right)\right) \tag{10.15}$$

- **4-step.**

$$U^{n+4} = U^{n+3} + \frac{k}{720}\begin{pmatrix} -19f\left(U^n\right) + 106f\left(U^{n+1}\right) - 264f\left(U^{n+2}\right) \\ +646f\left(U^{n+3}\right) + 251f\left(U^{n+4}\right) \end{pmatrix} \tag{10.16}$$

Example 10.2. The explicit *Nyström methods* have the form

$$U^{n+r} = U^{n+r-2} + k\sum_{j=0}^{r-1}\beta_j f\left(U^{n+j}\right) \tag{10.17}$$

with the β_j chosen to give order r. The midpoint method (4.11) is a two-step explicit Nyström method. A two-step implicit Nyström method is *Simpson's rule*,

$$U^{n+2} = U^n + \frac{2k}{6}\left(f\left(U^n\right) + 4f\left(U^{n+1}\right) + f\left(U^{n+2}\right)\right) \tag{10.18}$$

This reduces to Simpson's rule for quadrature if applied to the ODE $u'(t) = f(t)$.

10.1 Local Truncation Error

For LMMs it is easy to derive a general formula for the LTE. We have

$$\tau\left(t_{n+r}\right) = \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j u\left(t_{n+j}\right) - k\sum_{j=0}^{r}\beta_j f\left(u\left(t_{n+j}\right)\right)\right) \tag{10.19}$$

$$= \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j u\left(t_{n+j}\right) - k\sum_{j=0}^{r}\beta_j u'\left(t_{n+j}\right)\right) \tag{10.20}$$

We have used $f\left(u\left(t_{n+j}\right)\right) = u'\left(t_{n+j}\right)$ since $u(t)$ is the exact solution of the ODE. Assuming u is smooth and expanding in Taylor series gives

$$u\left(t_{n+j}\right) = u\left(t_n\right) + \sum_{i=1}^{p}\frac{1}{i!}(jk)^i u^{(i)}\left(t_n\right) + O\left(k^{p+1}\right) \tag{10.21}$$

$$u'\left(t_{n+j}\right) = \sum_{i=1}^{p}\frac{1}{(i-1)!}(jk)^{i-1}u^{(i)}\left(t_n\right) + O\left(k^p\right) \tag{10.22}$$

and also

$$\tau\left(t_{n+r}\right) \tag{10.23}$$

$$= \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j u\left(t_{n+j}\right) - k\sum_{j=0}^{r}\beta_j u'\left(t_{n+j}\right)\right) \tag{10.24}$$

$$= \frac{1}{k}\begin{pmatrix} \sum_{j=0}^{r}\alpha_j\left(u\left(t_n\right) + \sum_{i=1}^{p}\frac{1}{i!}(jk)^i u^{(i)}\left(t_n\right) + O\left(k^{p+1}\right)\right) \\ -k\sum_{j=0}^{r}\beta_j\left(\sum_{i=1}^{p}\frac{1}{(i-1)!}(jk)^{i-1}u^{(i)}\left(t_n\right) + O\left(k^p\right)\right) \end{pmatrix} \tag{10.25}$$

$$= \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j\right)u(t_n) + \sum_{i=1}^{p}\left(\sum_{j=0}^{r}\frac{1}{i!}j^i k^{i-1}\alpha_j u^{(i)}(t_n)\right) \tag{10.26}$$

$$-\sum_{i=1}^{p}\left(\sum_{j=0}^{r}\frac{1}{(i-1)!}(jk)^{i-1}\beta_j u^{(i)}(t_n)\right) + O(k^p) \tag{10.27}$$

$$= \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j\right)u(t_n) \tag{10.28}$$

$$+\sum_{i=1}^{p}k^{i-1}\left(\sum_{j=0}^{r}\left(\frac{1}{i!}j^i\alpha_j - \frac{1}{(i-1)!}j^{i-1}\beta_j\right)\right)u^{(i)}(t_n) + O(k^p) \tag{10.29}$$

The method is *consistent* if $\tau \to 0$ as $k \to 0$, which requires that at least the first two terms in this expansion vanish,

$$\sum_{j=0}^{r}\alpha_j = 0 \tag{10.30}$$

$$\sum_{j=0}^{r}j\alpha_j = \sum_{j=0}^{r}\beta_j \tag{10.31}$$

If the first $p+1$ terms vanish, then the method will be pth order accurate. Note that these conditions depend only on the coefficients α_j and β_j of the method and not on the particular differential equation being solved.

10.2 Characteristic Polynomials

It is convenient at this point to introduce the so-called characteristic polynomial $\rho(\zeta)$ and $\sigma(\zeta)$ for the LMM,

$$\rho(\zeta) = \sum_{j=0}^{r}\alpha_j\zeta^j \tag{10.32}$$

$$\sigma(\zeta) = \sum_{j=0}^{r}\beta_j\zeta^j \tag{10.33}$$

The first of these is a polynomial of degree r. So is $\sigma(\zeta)$ if the method is implicit; otherwise its degree is less than r. Note that

$$\rho(1) = \sum_{j=0}^{r}\alpha_j \tag{10.34}$$

$$\rho'(\zeta) = \sum_{j=0}^{r}j\alpha_j\zeta^{j-1} \tag{10.35}$$

so that the consistency conditions (10.30)-(10.31) can be written quite concisely as conditions on these two polynomials,

$$\rho(1) = 0 \tag{10.36}$$

$$\rho'(1) = \sigma(1) \tag{10.37}$$

This, however, is not the main reason for introducing these polynomials. The location of the roots of certain polynomials related to ρ and σ plays a fundamental role in stability theory as we will see in the next two chapters [1].

Example 10.3. The two-step Adams-Moulton method

$$U^{n+2} = U^{n+1} + \frac{k}{12}\left(-f(U^n) + 8f\left(U^{n+1}\right) + 5f\left(U^{n+2}\right)\right) \tag{10.38}$$

has characteristic polynomials

$$\rho(\zeta) = \zeta^2 - \zeta \tag{10.39}$$

$$\sigma(\zeta) = \frac{1}{12}\left(-1 + 8\zeta + 5\zeta^2\right) \tag{10.40}$$

10.3 Starting Values

One difficulty with using LMMs if $r > 1$ is that we need the values $U^0, U^1, \ldots, U^{r-1}$ before we can begin to apply the multistep method. The value $U^0 = \eta$ is known from the initial data for the problem, but the other values are not and typically must be generated by some other numerical method or methods.

Example 10.4. If we want to use the midpoint method (4.11), then we need to generate U^1 by some other method before we begin to apply (4.11) with $n = 1$. We can obtain U^1 from U^0 using any one-step method, such as Euler's method or the trapezoidal method, or a higher order Taylor series or Runge-Kutta method. Since the midpoint method is second order accurate we need to make sure that the value U^1 we generate is sufficiently accurate so that this second order accuracy will not be lost. Out first impulse might be to conclude that we need to use a second order accurate method such as trapezoidal method rather than the first order accurate Euler method, but this is wrong. The overall method is second order in either case. The reason that we achieve second order accuracy even if Euler is used in the first step is exactly analogous to what was observed earlier for boundary value problems, where we found that we can often get away with one order of accuracy lower in the local error at a single point than what we have elsewhere.

In the present context this is easiest to explain in terms of the one-step error. The midpoint method has a one-step error that is $O\left(k^3\right)$ and because this method is applied in $O\left(\frac{T}{k}\right)$ time steps, the global error is expected to be $O\left(k^2\right)$. Euler's method has a one-step error that is $O\left(k^2\right)$, but we are applying this method only once.

If $U^0 = \eta = u(0)$, then the error in U^1 obtained with Euler will be $O\left(k^2\right)$. If the midpoint method is stable, then this error will not be magnified unduly in later steps and its contribution to the global error will be only $O\left(k^2\right)$. The overall second order accuracy will not be affected.

More generally, with an r-step method of order p, we need r starting values

$$U^0, U^1, \ldots, U^{r-1} \tag{10.41}$$

and we need to generate these values using a method that has a *one-step error* that is $O(k^p)$ (corresponding to an LTE that is $O\left(k^{p-1}\right)$). Since the number of times we apply this method $(r-1)$ is independent of k as $k \to 0$, this is sufficient to give an $O(k^p)$ global error. Of course somewhat better accuracy (a smaller error constant) may be achieved by using a pth order accurate method for the starting values, which takes little additional work.

In software for the IVP, multistep methods generally are implemented in a form that allows changing the time step during the integration process, as is often required to efficiently solve the problem. Typically the order of the method is also allowed to vary, depending on how the solution is behaving. In such software it is then natural to solve the staring-value problem by initially taking a small time step with a one-step method and then ramping up to higher order methods and longer time steps as the integration proceeds and more past data are available.

10.4 Predictor-Corrector Methods

The idea of comparing results obtained with methods of different order as a way to choose the time step, discussed in [1] Section 5.7.1 for Runge-Kutta methods, is also used with LMMs. One approach is to use a *predictor-corrector method*, in which an explicit Adams-Bashforth method of some order is used to predict of some order is used to predict a value $\hat{U}^{n+1}$ and then the Adams-Moulton method of the same order is used to "correct" this value. This is done by using $\hat{U}^{n+1}$ on the right-hand side of the Adams-Moulton method inside the f evaluation, so that the Adams-Moulton formula is no longer implicit. For example, the one-step Adams-Bashforth (Euler's method) and the one-step Adams-Moulton method (the trapezoidal method) could be combined into

$$\hat{U}^{n+1} = U^n + kf(U^n) \tag{10.42}$$

$$U^{n+1} = U^n + \frac{1}{2}k\left(f(U^n) + f\left(\hat{U}^{n+1}\right)\right) \tag{10.43}$$

It can be shown that this method is second order accurate, like the trapezoidal method, but it also generates a lower order approximation and the difference between the two can be used to estimate the error. The Matlab routine `ode113` uses this approach, with Adams-Bashforth-Moulton methoremarkable teamwork reportsds of orders 1-12.

References

[1] Randall J. Leveque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, SIAM Society for Industrial and Applied Mathematics, 2007.

[2] Nguyen Quan Ba Hong, Doan Tran Nguyen Tung, Nguyen An Thinh, *Runge Kutta Methods for Ordinary Differential Equations*, 2016.

[3] `https://en.wikipedia.org/wiki/Pendulum_(mathematics)`