Chapter 16

# Differential Equations

A vast number of mathematical models in various areas of science and engineering involve differential equations. This chapter provides a starting point for a journey into the branch of scientific computing that is concerned with the simulation of differential problems.

We shall concentrate mostly on developing methods and concepts for solving initial value problems for ordinary differential equations: this is the simplest class (although, as you will see, it can be far from being simple), yet a very important one. It is the only class of differential problems for which, in our opinion, up-to-date numerical methods can be learned in an orderly and reasonably complete fashion within a first course text. Section 16.1 prepares the setting for the numerical treatment that follows in Sections 16.2–16.6 and beyond. It also contains a synopsis of what follows in this chapter.

Numerical methods for solving boundary value problems for ordinary differential equations receive a quick review in Section 16.7. More fundamental difficulties arise here, so this section is marked as advanced.

Most mathematical models that give rise to differential equations in practice involve partial differential equations, where there is more than one independent variable. The numerical treatment of partial differential equations is a vast and complex subject that relies directly on many of the methods introduced in various parts of this text. An orderly development belongs in a more advanced text, though, and our own description in Section 16.8 is downright anecdotal, relying in part on examples introduced earlier.

## 16.1 Initial value ordinary differential equations

Consider the problem of finding a function $y(t)$ that satisfies the *ordinary differential equation* (ODE)

$$\frac{dy}{dt} = f(t, y), \quad a \le t \le b.$$

The function $f(t, y)$ is given, and we denote the derivative of the sought solution by $y' = \frac{dy}{dt}$ and refer to $t$ as the *independent variable*.

Previous chapters have dealt with the question of how to numerically approximate, differentiate, or integrate an explicitly known function. Here, similarly, $f$ is given and the sought result is different from $f$ but relates to it. The main difference though is that $f$ depends on the unknown $y$ to be recovered. We would like to be able to compute the function $y(t)$, possibly for all $t$ in the interval $[a, b]$, given the ODE which characterizes the relationship between $y$ and some of its derivatives.

**Example 16.1.** The function $f(t,y) = -y + t$ defined for $t \geq 0$ and any real $y$ gives the ODE

$$y' = -y + t, \quad t \geq 0.$$

You can verify directly that for any scalar $\alpha$ the function

$$y(t) = t - 1 + \alpha e^{-t}$$

satisfies this ODE. So, the solution is not unique without further conditions.

Suppose next that a value $c$ is given in addition such that at the left end of the interval $y(0) = c$. Then $c = 0 - 1 + \alpha e^0$, and hence $\alpha = c + 1$, and the unique solution is

$$y(t) = t - 1 + (c+1)e^{-t}.$$

Such a solution function, seen as if spawned by the initial value, is sometimes referred to as a **trajectory**. ∎

### ODE systems

It will be convenient for presentation purposes in what follows to concentrate on a scalar ODE, because this makes the notation easier when introducing numerical methods. We should note though that scalar ODEs rarely appear in practice. ODEs almost always arise as systems, and these systems may sometimes be large. In the case of systems of ODEs we use vector notation and write our prototype ODE system as

$$\mathbf{y}' \equiv \frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}), \quad a \leq t \leq b. \tag{16.1a}$$

We shall assume that $\mathbf{y}$ has $m$ components, like $\mathbf{f}$.

In general, as in Example 16.1, there is a family of solutions depending on $m$ parameters for this ODE system. The solution becomes unique, under some mild assumptions, if $m$ *initial values* $\mathbf{c}$ are specified, so that

$$\mathbf{y}(a) = \mathbf{c}. \tag{16.1b}$$

This is then an **initial value problem**.

**Example 16.2.** Consider a tiny ball of mass 1 attached to the end of a rigid, massless rod of length $r = 1$. At its other end the rod's position is fixed at the origin of a planar coordinate system; see Figure 16.1.

Denoting by $\theta$ the angle between the pendulum and the negative vertical axis, the friction-free motion is governed by the ODE

$$\frac{d^2\theta}{dt^2} \equiv \theta'' = -g \sin(\theta),$$

where $g$ is the scaled constant of gravity, e.g., $g = 9.81$, and $t$ is time. This is a simple, nonlinear ODE for $\theta$. The initial position and velocity configuration translates into values for $\theta(0)$ and $\theta'(0)$.

We can write this ODE as a first order system: let

$$y_1(t) = \theta(t), \quad y_2(t) = \theta'(t).$$

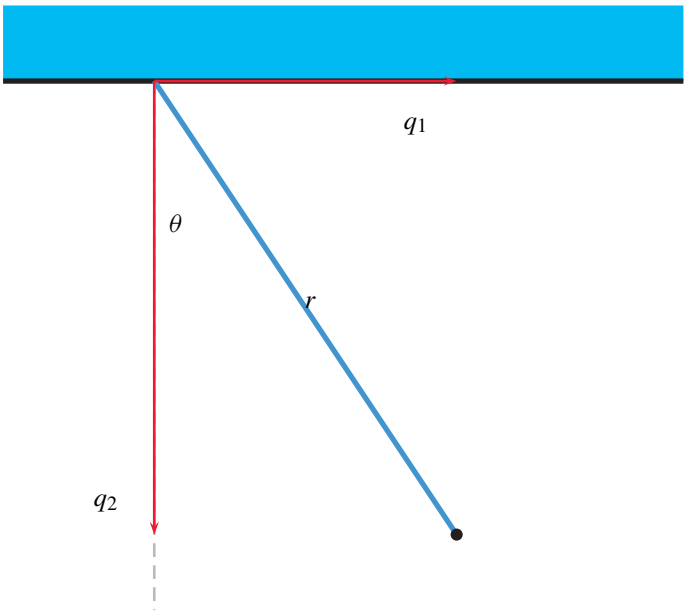



**Figure 16.1.** *A simple pendulum.*

Then $y_1' = y_2$ and $y_2' = -g\sin(y_1)$. The problem is then written in the form (16.1) with $a = 0$, where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ -g\sin(y_1) \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} \theta(0) \\ \theta'(0) \end{pmatrix}.$$

Note that even in this simple example, which seems to appear in just about any introductory text on ODEs, the dependence of $\mathbf{f}$ on $\mathbf{y}$ is *nonlinear*.

Moreover, the dependence of $\mathbf{f}$ on $t$ is only through $\mathbf{y}(t)$, and not explicitly or directly, so in this example $\mathbf{f} = \mathbf{f}(\mathbf{y})$. The ODE system is called **autonomous** in such a case. ∎

In the previous few chapters we have denoted the independent variable by $x$, but here we denote it by $t$. This is because it is convenient to think of initial value problems as depending on **time**, as in Example 16.2, thus also justifying the word "initial" in their name. However, $t$ does not have to correspond to physical time in a particular application.

Going back to scalar ODE notation, note that in principle we can write the initial value ODE in integral form: integrating both sides of the ODE from $a$ to $t$ we have

$$y(t) = c + \int_a^t f(s, y(s))ds, \quad a \le t \le b.$$

This highlights both similarities to and differences from the problem of integration considered in Sections 15.1–15.5. Specifically, on the one hand both prototype problems involve integration; indeed, choosing $f$ not to depend on $y$ and setting $t = b$ and $c = 0$ reveals integration as a special case of the problem considered here. But on the other hand, it is a very special case, because here we are recovering a function of $t$ rather than a value as in definite integration, and even more importantly when deriving numerical methods, the integrand $f$ depends on the unknown function. Numerical

methods for ODEs are in general significantly more varied and complex than those considered before for quadrature, as we shall soon see.

## Synopsis: Numerical methods described in this chapter

The simplest method for numerically integrating an initial value ODE is *forward Euler*. In fact, when teaching it we often have the feeling that everyone has seen it before. We nonetheless devote Section 16.2 to it, because it is a great vehicle for introducing a variety of important concepts that are relevant for other, more sophisticated methods as well.

One deficiency of the forward Euler method is its low accuracy order. Both Sections 16.3 and 16.4 are devoted to higher order families of methods. Particular members of these families are the methods of choice in almost any general-purpose ODE code today.

Issues of stability of the numerical method are considered in Section 16.2 and, more generally, in Section 16.5. For many problems, just about any reasonable method (including of course all those considered in this chapter) is basically stable. But for an important class of *stiff* problems there is an issue of *absolute stability* restriction which makes some methods preferable in a significant sense over others. Let us say no more at this early stage.

Of course, as in Section 15.4, writing a general-purpose code involves also estimating and controlling the error, and Section 16.6 provides a brief introduction to this topic. Things quickly get much messier here than for the case of numerical integration considered in Section 15.4.

### Boundary value ODEs

Sections 16.2–16.6 are all concerned with methods for initial value ODE systems of the form (16.1), where $\mathbf{y}(a)$ is given. Other problems where all of the solution components $\mathbf{y}$ are given at one point can basically be converted to an initial value problem. For example, the *terminal value problem* where $\mathbf{y}(b)$ is given is converted to an initial value problem by a change of variable $\tau = b - t$, which transforms the interval of definition from $[a,b]$ to $[0,b-a]$ with $\mathbf{y}(0)$ now given. But there are other, *boundary value problems* for the same prototype ODE system (16.1a), where some component of $\mathbf{y}$ is given at a different value of $t$ than another. For instance, in Example 16.2 we may be given two position values for $\theta(0)$ and $\theta(1)$ and none on the velocity $\theta'$. See also Examples 4.17 and 9.3 for instances of boundary value ODEs. Methods for this more difficult class of ODE problems that are more general than in Example 9.3 are briefly considered in Section 16.7.

### Partial differential equations

Most differential problems that arise in practice depend on more than one independent variable, giving rise to a *partial differential equation* (PDE). For instance, Example 7.1 considers the Poisson equation

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = g(x,y), \quad 0 < x, y < 1,$$

with the solution values $u(x,y)$ given on the boundary of the unit square. The independent variables here are $x$ and $y$.

Reminiscent of the situation with interpolation and integration briefly described in Sections 11.6 and 15.6, respectively, everything becomes more complicated in a hurry when moving from ODEs to PDEs. Indeed, one reason for concentrating in this chapter on the easiest case of initial value ODEs is that it gives significant taste and provides important expertise buildup necessary for PDE computations. In Section 16.8 we briefly review PDEs and some corresponding numerical methods.

## 16.2 Euler's method

Euler's method, which is also known as the **forward Euler** method (to distinguish it from its *backward Euler* counterpart, to be discussed later), is the simplest numerical method for approximately solving initial value ODEs. Here it is used as a vehicle for studying several important, basic notions.

In this section we introduce in the context of the forward Euler method the following **general concepts** for numerical ODE methods:

- method derivation,
- explicit vs. implicit methods,
- local truncation error and global error,
- order of accuracy,
- convergence, and
- absolute stability and stiffness.

### Method derivation

We first consider finding an approximate solution for a scalar initial value ODE at equidistant abscissae. Thus, define the points

$$t_0 = a, \;\; t_i = a + ih, \quad i = 0, 1, 2, \ldots, N,$$

where $h = \frac{b-a}{N}$ is the step size. Denote the approximate solution for $y(t_i)$ by $y_i$. The step size $h$ may vary in principle, $h = h_i$, but unless otherwise noted we keep it uniform (constant) in this section and consider varying it wisely in Section 16.6.

Recall from Example 1.2 and Section 14.1 the forward difference formula

$$y'(t_i) = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{h}{2} y''(\xi_i).$$

By the ODE, $y'(t_i) = f(t_i, y(t_i))$, so

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2} y''(\xi_i).$$

This is satisfied by the exact solution $y(t)$ of the ODE. Dropping the truncation term, we obtain the *forward Euler method*, which defines the approximate solution $\{y_i\}_{i=0}^N$ by

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_i, y_i), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

See Figure 16.2. We omit the details of the problem for which the graph in this figure was produced, since the demonstrated issues are not specific to one particular ODE problem.

This simple formula allows us to march forward in $t$. Assuming that the various parameters and the function `f` are specified, the following MATLAB script does the job:

```
t = [a:h:b];
y(1) = c;
for i=1:N
  y(i+1) = y(i) + h * f(t(i),y(i));
end
```

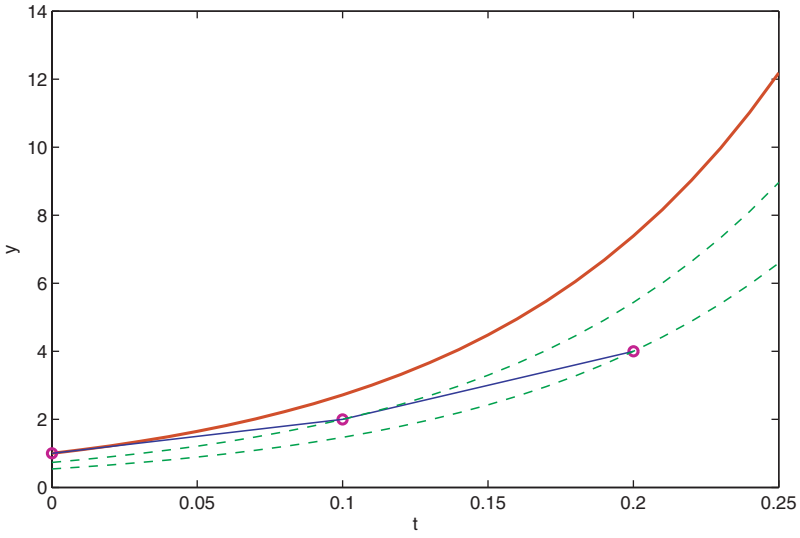



**Figure 16.2.** *Two steps of the forward Euler method. The exact solution is the curved solid line. The numerical values obtained by the Euler method are circled and lie at the nodes of a broken line that interpolates them. The broken line is tangential at the beginning of each step to the ODE trajectory passing through the corresponding node (dashed lines).*

This produces an array of abscissae and ordinates, $(t_i, y_i)$, which is good for plotting. In other applications, fewer output points may be required. For instance, if only the approximate value of $y(b)$ is desired, then we can save storage by the script

```
t = a; y = c;
for i=1:N
  y = y + h * f(t,y);
  t = t + h;
end
```

This script works also if `c`, `y`, and `f` are arrays of the same size, corresponding to an ODE system.

**Example 16.3.** Consider the simple, linear initial value ODE problem given by

$$y' = y, \quad y(0) = 1.$$

The exact solution is $y(t) = e^t$.

This ODE is autonomous, $f(t, y) = y$. Euler's method reads

$$\begin{aligned} y_0 &= 1, \\ y_{i+1} &= y_i + hy_i = (1+h)y_i, \quad i = 0, 1, 2, \ldots. \end{aligned}$$

We obtain the results listed in Table 16.1. The errors clearly reduce by a factor of roughly $1/2$ when $h$ is cut by the same factor from 0.2 to 0.1.

Note also that the absolute error increases as $t$ increases in this particular example. Even the relative error increases here with $t$, although not as fast. ∎

**Table 16.1.** *Absolute errors using the forward Euler method for the ODE $y' = y$. The values $e_i = y(t_i) - e_i$ are listed under Error.*

| | | $h = 0.2$ | | $h = 0.1$ | |
|---|---|---|---|---|---|
| $t_i$ | $y(t_i)$ | $y_i$ | Error | $y_i$ | Error |
| 0 | 1.000 | 1.000 | 0.0 | 1.000 | 0.0 |
| 0.1 | 1.105 | | | 1.100 | 0.005 |
| 0.2 | 1.221 | 1.200 | 0.021 | 1.210 | 0.011 |
| 0.3 | 1.350 | | | 1.331 | 0.019 |
| 0.4 | 1.492 | 1.440 | 0.052 | 1.464 | 0.028 |
| 0.5 | 1.694 | | | 1.611 | 0.038 |
| 0.6 | 1.822 | 1.728 | 0.094 | 1.772 | 0.051 |

### Explicit vs. implicit methods

What happens if we replace the forward difference formula that leads to the forward Euler method by the *backward* formula

$$y'(t_{i+1}) \approx \frac{y(t_{i+1}) - y(t_i)}{h},$$

which in the context of Section 14.1 is the most innocent modification imaginable? Here this leads to the **backward Euler** method, given by

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_{i+1}, y_{i+1}), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

It might be tempting to think of the backward Euler method as a minor variation, not much different from its forward counterpart. But there is a rather substantial difference here: in the backward Euler method, the computation of $y_{i+1}$ depends *implicitly* on $y_{i+1}$ itself! This leads to the important notion of *explicit* and *implicit* methods. If the evaluation of $y_{i+1}$ involves the evaluation of $f$ at the unknown value $y_{i+1}$ itself, then the method is implicit. Carrying out an integration step requires solving a usually nonlinear equation for $y_{i+1}$. If on the other hand the evaluation of $y_{i+1}$ involves the evaluation of $f$ only at known points, i.e., values such as $y_i$ obtained in previous steps or stages, then the method is explicit. Hence, the forward Euler method is explicit, whereas the backward Euler method is implicit.

Explicit methods are significantly easier to implement, and carrying out an integration step is typically much faster. This, and its general simplicity, is what has given the forward Euler method its great popularity in numerous areas of application. The backward Euler method, in contrast, while easy to understand conceptually, requires a deeper understanding of numerical issues and cannot be programmed as easily and seamlessly as the forward Euler method. Nonetheless, as we shall see later in this section and in Section 16.5, backward Euler and other implicit methods have numerical properties that in certain cases, and for certain applications, make them superior to explicit methods.

### Local truncation error, order, and global error

The *local truncation error*, $d_i$, is the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form, at integration step $i$. The concept is general and applies not just for Euler's method.

The method has *order of accuracy* $q$ if $q$ is the lowest positive integer such that for any sufficiently smooth exact solution $y(t)$ we have

$$\max_i |d_i| = \mathcal{O}(h^q).$$

The *global error* of the method is defined by

$$e_i = y(t_i) - y_i, \quad i = 0, 1, \ldots, N.$$

Let us demonstrate these concepts on the forward Euler method. Here we have

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - f(t_i, y(t_i)).$$

From the method's derivation, $d_i = \frac{h}{2} y''(\xi_i)$. This expression is linear in $h$, i.e., $d_i = \mathcal{O}(h)$. Hence the method is first order accurate, meaning $q = 1$.

The same holds for the backward Euler method, where $d_i = -\frac{h}{2} y''(\xi_i)$. This method is also first order accurate.

More generally, designing a numerical discretization of order $q$ would involve manipulating Taylor series expansions (see page 5) to achieve an appropriately small local truncation error. For the methods of Section 16.3 this can be a hair-raising job, more so than anything encountered in Chapter 14. But the principle, and for the Euler methods also the execution, are simple.

### Convergence

The method is said to *converge* if the maximum global error tends to 0 as $h$ tends to 0, provided the exact solution exists and is reasonably smooth.

In general, if nothing goes wrong we expect the global error $e_i = y(t_i) - y_i$ to be of the same order as the local truncation error. So, for the forward Euler method we expect

$$e_i = \mathcal{O}(h) \quad \forall\, i.$$

We have seen such behavior in Example 16.3. Let us show that this is true in general under very mild conditions on $f(t, y)$. Specifically, we assume the conditions specified in the Forward Euler Convergence Theorem given on the next page.

Since the local truncation error satisfies

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - f(t_i, y(t_i)), \quad \text{and also}$$
$$0 = \frac{y_{i+1} - y_i}{h} - f(t_i, y_i),$$

we can subtract the two expressions and obtain for the error the difference formula

$$d_i = \frac{e_{i+1} - e_i}{h} - [f(t_i, y(t_i)) - f(t_i, y_i)].$$

The assumption that $f$ satisfies a Lipschitz condition with a constant $L$ implies that we can write the error difference equation as

$$\begin{aligned} |e_{i+1}| &= |e_i + h[f(t_i, y(t_i)) - f(t_i, y_i)] + h d_i| \\ &\le |e_i| + hL|e_i| + hd, \end{aligned}$$

**Theorem: Forward Euler Convergence.**
Let $f(t,y)$ have bounded partial derivatives in a region $\mathcal{D} = \{a \le t \le b, |y| < \infty\}$.

Note that this implies Lipschitz continuity in $y$: there exists a constant $L$ such that for all $(t,y)$ and $(t,\hat{y})$ in $\mathcal{D}$ we have

$$|f(t,y) - f(t,\hat{y})| \le L|y - \hat{y}|.$$

Then Euler's method converges and its global error decreases linearly in $h$.
Moreover, assuming further that

$$|y''(t)| \le M, \quad a \le t \le b,$$

the global error satisfies

$$|e_i| \le \frac{Mh}{2L}[e^{L(t_i-a)} - 1], \quad i = 0, 1, \ldots, N.$$

where $d$ is a bound on the local truncation errors, $d \ge \max_{0\le i\le N-1} |d_i|$. Thus, if we know

$$M = \max_{a\le t\le b} |y''(t)|,$$

then set

$$d = \frac{M}{2}h.$$

It follows that

$$\begin{aligned}
|e_{i+1}| &\le (1+hL)|e_i| + hd \\
&\le (1+hL)[(1+hL)|e_{i-1}| + hd] + hd = (1+hL)^2|e_{i-1}| + (1+hL)hd + hd \\
&\le \cdots \le (1+hL)^{i+1}|e_0| + hd\sum_{j=0}^{i}(1+hL)^j \\
&\le d\left[e^{L(t_{i+1}-a)} - 1\right]/L \\
&\le \frac{Mh}{2L}\left[e^{L(t_{i+1}-a)} - 1\right].
\end{aligned}$$

Above, to arrive at the one inequality before last we have used $e_0 = 0$ and a bound for the geometric sum of powers of $1 + hL$. ♦

This provides a proof of convergence for the forward Euler method as stated in the theorem on this page. Note that the global error in Euler's method is indeed first order in $h$, as observed in Example 16.3.

Note also that the error bound may grow with $t$. This is realistic to expect when the exact solution grows; the relative error is more meaningful then. But when the exact solution decays, then the above bound on the absolute error becomes too pessimistic, in that the actual global error will be much smaller than its bound.

**Example 16.4.** A more challenging problem than that in Example 16.3 originates in plant physiology and is defined by the following MATLAB script:

```
function f = hires(t,y)
%
% f = hires(t,y)
%
% High irradiance response function arising in plant physiology
f = y;
f(1) = -1.71*y(1) + .43*y(2) + 8.32*y(3) + .0007;
f(2) = 1.71*y(1) - 8.75*y(2);
f(3) = -10.03*y(3) + .43*y(4) + .035*y(5);
f(4) = 8.32*y(2) + 1.71*y(3) - 1.12*y(4);
f(5) = -1.745*y(5) + .43*y(6) + .43*y(7);
f(6) = -280*y(6)*y(8) + .69*y(4) + 1.71*y(5) - .43*y(6) + .69*y(7);
f(7) = 280*y(6)*y(8) - 1.81*y(7);
f(8) = -280*y(6)*y(8) + 1.81*y(7);
```

This ODE system (which has $m = 8$ components) is to be integrated from $a = 0$ to $b = 322$ starting from $\mathbf{y}(0) = \mathbf{y}_0 = (1,0,0,0,0,0,0,.0057)^T$. The script

```
h = .001; t = 0:h:322;
y = y0 * ones(1,length(t));
for i = 1:length(t)-1
  y(:,i+1) = y(:,i) + h*hires(t(i),y(:,i));
end
plot(t,y(6,:))
```

(plus labeling) produces Figure 16.3. To gauge accuracy of the Euler method we repeat the solution process with a more accurate method from Section 16.3 below and regard the difference between these approximate solutions as the error for the less accurate forward Euler method. Based on this the maximum absolute error occurs after 447 steps at $t_* = 2.235$, where $y(t_*) = .4483$ and $|y(t_*) - y_{447}| = 4.67 \times 10^{-4}$. ∎

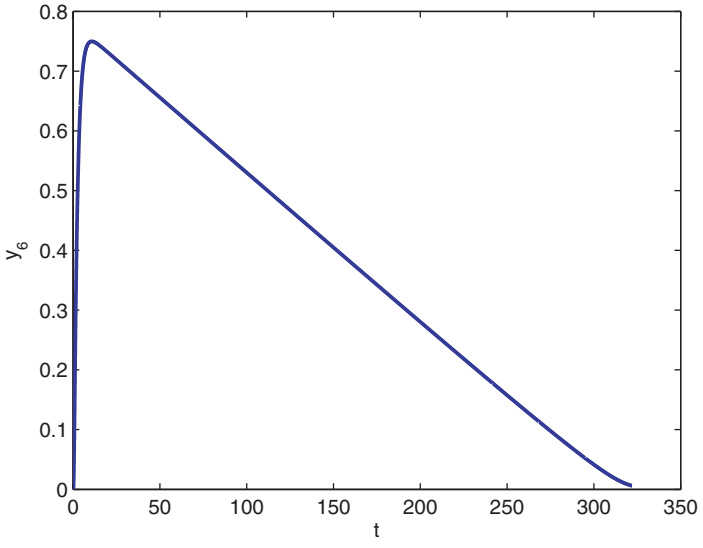



**Figure 16.3.** *The sixth solution component of the HIRES model.*

**Note:** Since deriving discretization methods for differential equations involves numerical differentiation, there is an unavoidable roundoff error that behaves like $\mathcal{O}(h^{-1})$; recall the discussion in Section 14.4.

However, roundoff error is typically a secondary concern here, so long as the method is absolutely stable as explained next. This is so both because $h$ is usually not incredibly small (for reasons of efficiency and modeling limitations) and because $y$, and not $y'$, is the function for which the approximations are sought.

### Absolute stability and stiffness

Let us consider next a simple scalar ODE known as the **test equation** and given by

$$y' = \lambda y,$$

where $\lambda$ is a real constant. The exact solution is $y(t) = e^{\lambda t} y(0)$. So, the exact solution increases if $\lambda > 0$ and decreases otherwise.

Euler's method gives

$$y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda) y_i = \cdots = (1 + h\lambda)^{i+1} y(0).$$

If $\lambda > 0$, then the approximate solution grows, and so does the absolute error, yet the relative error remains reasonably small. But if $\lambda < 0$, then the exact solution decays, so we must require at the very least that the approximate solution not grow. We then demand

$$|y_{i+1}| \le |y_i|.$$

For the forward Euler method this corresponds to insisting that

$$|1 + h\lambda| \le 1 \Rightarrow h \le \frac{2}{|\lambda|}.$$

It is a requirement of *absolute stability*: regardless of accuracy considerations, the constant step size $h$ must not exceed a certain bound that depends on the problem which is being approximately solved.

**Example 16.5.** For the initial value ODE problem

$$y' = -1000(y - \cos(t)) - \sin(t), \quad y(0) = 1,$$

the solution is $y(t) = \cos(t)$. A broken line interpolation of $y(t)$, which is what MATLAB uses by default for plotting, looks good already for $h = 0.1$. But stability decrees that for Euler's method we need

$$h \le \frac{1}{500}$$

for *any* reasonable accuracy!

For instance, using $h = .0005\pi$ and evaluating the numerical solution at $t = \pi/2$ (where the exact solution vanishes), we get the error $y_{1000} = 3.7\text{e-}10$. But using $h = .001\pi$ instead, which is only double the step size but no longer satisfies the absolute stability bound, we get $y_{500} = -3.7\text{e+}159$, meaning that essentially a blowup has occurred. This can be annoying. ∎

An ODE such as that of Example 16.5, where stability considerations make us take a much smaller step size for the forward Euler method than accuracy considerations would otherwise dictate, is called **stiff**.

Let us make two essential comments:[60]

- You may wonder how it is possible that the local truncation error would change so much upon making such a small change in $h$ in Example 16.5. Well, it doesn't. The error that accumulates wildly upon violation of the absolute stability condition is the unruly *roundoff error*!
- There is a fundamental difference between the error concepts introduced earlier and the absolute stability requirement: the former all ask what happens when $h \to 0$ for a fixed ODE problem, whereas the latter is concerned with the situation for a fixed and not necessarily small step size or, more precisely, with a bound involving $z = h\lambda$.

When deriving the forward Euler method, we have also introduced the implicit, *backward Euler* method. At the time it did not seem that anything good could come out of this method, being so much harder to use than its equally accurate forward Euler counterpart. But now it comes to life: applied to the test equation we get $y_{i+1} = y_i + h\lambda y_{i+1}$, and hence

$$y_{i+1} = \frac{1}{1-h\lambda} y_i.$$

Therefore, $|y_{i+1}| \le |y_i|$ for any $h > 0$ and $\lambda < 0$. There is no annoying absolute stability restriction here!

**Example 16.6.** For the problem of Example 16.5, applying the backward Euler method yields the following results:

1. Using $h = .0005\pi$ and evaluating the numerical solution at $t = \pi/2$, we get the error $y_{1000} = -1.2\text{e-}9$.
2. Using $h = .001\pi$, we get the error $y_{500} = -3.2\text{e-}9$.

The global error behaves as expected from local truncation error considerations alone. ∎

Let us return to forward Euler and ask what happens for a more general ODE system (16.1a). For the simplest case of interest of such a system we have $\mathbf{f}(\mathbf{y}) = A\mathbf{y}$, where $A$ is a constant $m \times m$ matrix, further assumed to have eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ and be diagonalizable (see Section 4.1). Thus, there is a transformation matrix $T$ such that $T^{-1}AT$ is diagonal with the eigenvalues on its main diagonal. Then it is not difficult to see that for the transformed unknowns $\mathbf{x} = T^{-1}\mathbf{y}$ the system decouples to give the scalar ODEs

$$x_j' = \lambda_j x_j, \quad j = 1, 2, \ldots, m.$$

The absolute stability requirement for the forward Euler method thence translates into the requirement that

$$|1 + h\lambda_j| \le 1, \quad j = 1, 2, \ldots, m.$$

The complication is, however, in that eigenvalues need not be real! In general we must therefore consider the test equation with *complex* $\lambda$, and we do so in Section 16.5.

*Specific exercises for this section*: Exercises 1–5.

[60] Do not be misled by the seeming simplicity of these comments: they involve complex, nontrivial issues.

## 16.3 Runge–Kutta methods

Euler's method is only first order accurate. This implies inefficiency for many applications, as the step size must be taken rather small, and thus $N$ becomes rather large, to achieve satisfactory accuracy.

To describe higher order methods, consider the integration of our prototype scalar ODE $y' = f(t,y)$ over one step, from $t_i$ to $t_{i+1}$. Thus, we assume that an approximation $y_i$ to $y(t_i)$ is known at the point $t_i$ and develop ways to obtain an approximation $y_{i+1}$ to $y(t_{i+1})$ at the next point $t_{i+1}$.

To obtain higher order methods there are extensions of the forward Euler method in several directions. A very common approach described in this section is to use only information in the current step $[t_i, t_{i+1}]$. Thus, a *Runge–Kutta* (RK) method is a **one-step method** in which repeated function evaluations are used to achieve a higher order; see Figure 16.4.

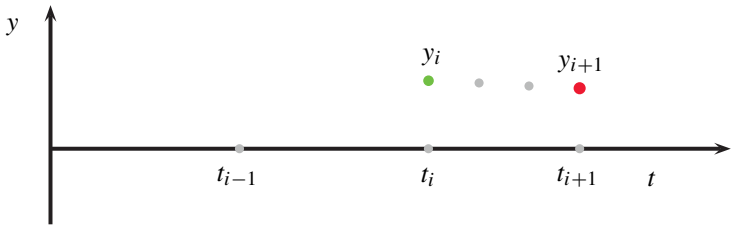



**Figure 16.4.** *RK method: repeated evaluations of the function $f$ in the current mesh subinterval $[t_i, t_{i+1}]$ are combined to yield a higher order approximation $y_{i+1}$ at $t_{i+1}$. (Reprinted from Ascher* [3]*.)*

Below we proceed to derive explicit and implicit RK methods, starting from simple and intuitive ones and gradually becoming more general.

### Two simple RK methods

Our purpose next is to derive a particular, second order accurate, explicit RK method as an example, to get the hang of it.

Integrating from $t_i$ to $t_{i+1}$ we can write the ODE as

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt.$$

Now, consider applying the *trapezoidal* quadrature rule as in Sections 15.1 and 15.2. This gives

$$y_{i+1} = y_i + \frac{h}{2}(f(t_i, y_i) + f(t_{i+1}, y_{i+1})),$$

which is referred to as the **implicit trapezoidal** method. By considering the local truncation error

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{1}{2}[f(t_i, y(t_i)) + f(t_{i+1}, y(t_{i+1}))],$$

you should be able to quickly convince yourself that $d_i = \mathcal{O}(h^2)$, i.e., this is a second order method.

However, a serious setback is that this method is implicit. Thus, the evaluation of $y_{i+1}$ requires solving a generally nonlinear equation. As discussed in Section 16.2, this is a major difference between quadrature and numerical integration of differential equations. For an ODE system of size $m$ we get a possibly nonlinear system of $m$ algebraic equations to solve for a vector $\mathbf{y}_{i+1}$ at each step

$i$. One would like, if possible, to avoid this expense and complication (although not at any cost, as becomes clear in Section 16.5).

Using the only explicit method we know thus far (yes, that would be forward Euler), we can approximate $y_{i+1}$ at first by

$$Y = y_i + hf(t_i, y_i).$$

Then plug this into the implicit trapezoidal formula, obtaining the **explicit trapezoidal** method

$$y_{i+1} = y_i + \frac{h}{2}(f(t_i, y_i) + f(t_{i+1}, Y)).$$

Note that there are two function evaluations involved here at the $i$th step, namely, of $f(t_i, y_i)$ and $f(t_{i+1}, Y)$: this is a two-stage, explicit RK method.

To ascertain that this method is second order accurate we would have to plug in the exact solution into the difference equation, obtaining

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{1}{2}[f(t_i, y(t_i)) + f(t_{i+1}, \hat{Y})],$$
$$\hat{Y} = y(t_i) + hy'(t_i).$$

Taylor expansions follow, and Exercise 6 completes the job. This gets a bit technical, but the result is intuitively clear, because the first order accurate intermediate step $Y$ gets multiplied by $h$ in the formula for $y_{i+1}$.

### Two more simple RK methods

Applying the trapezoidal rule in the way demonstrated above is perhaps the simplest way to obtain an explicit RK method of order 2. But using the midpoint rule instead is not much more complex.

A direct application of the midpoint quadrature rule gives the **implicit midpoint** method

$$y_{i+1} = y_i + hf(t_{i+1/2}, y_{i+1/2}),$$

where

$$t_{i+1/2} = \frac{t_i + t_{i+1}}{2} = t_i + h/2, \;\; y_{i+1/2} = \frac{y_i + y_{i+1}}{2}.$$

Then using forward Euler to approximate $y_{i+1/2}$ yields the two-stage, **explicit midpoint** method, which is an RK method of order 2 given by

$$y_{i+1} = y_i + hf(t_{i+1/2}, Y), \quad \text{where}$$
$$Y = y_i + \frac{h}{2} f(t_i, y_i).$$

At each step $i$ we evaluate $Y$ and then $y_{i+1}$. This requires two function evaluations per step, so the method is roughly twice as expensive as the forward Euler method per step. The explicit midpoint method is comparable to the explicit trapezoidal method, both in order and in expense.

### An explicit RK method of order 4

The **classical RK method** is based on the Simpson quadrature rule and uses four explicit stages, hence four function evaluations per step, to achieve $\mathcal{O}(h^4)$ accuracy. It is given by

$$\begin{aligned}
Y_1 &= y_i,\\
Y_2 &= y_i + \frac{h}{2} f(t_i, Y_1),\\
Y_3 &= y_i + \frac{h}{2} f(t_{i+1/2}, Y_2),\\
Y_4 &= y_i + h f(t_{i+1/2}, Y_3),\\
y_{i+1} &= y_i + \frac{h}{6}\big(f(t_i, Y_1) + 2f(t_{i+1/2}, Y_2) + 2f(t_{i+1/2}, Y_3) + f(t_{i+1}, Y_4)\big).
\end{aligned}$$

Showing that this formula is actually fourth order accurate is not a simple matter, unlike for the composite Simpson quadrature, and will not be discussed further in this text.

Here is a simple MATLAB function that implements the classical RK method using a fixed step size. It is written for an ODE system, with the extension from the scalar ODE method requiring almost no effort. Note that instead of storing the $Y_j$'s we evaluate and store $K_j = f(t_j, Y_j)$.

```
function [t,y] = rk4(f,tspan,y0,h)
%
% function [t,y] = rk4(f,tspan,y0,h)
%
% A simple integration routine to solve the
% initial value ODE   y' = f(t,y), y(a) = y0,
% using the classical 4-stage Runge-Kutta method
% with a fixed step size h.
% tspan = [a b] is the integration interval.
% Note that y and f can be vector functions

y0 = y0(:);         % make sure y0 is a column vector
m = length(y0);           % problem size
t = tspan(1):h:tspan(2);  % output abscissae
N = length(t)-1;          % number of steps
y = zeros(m,N+1);
y(:,1) = y0;   % initialize

% Integrate
for i=1:N
  % Calculate the four stages
  K1 = feval(f, t(i),y(:,i)    );
  K2 = feval(f, t(i)+.5*h, y(:,i)+.5*h*K1);
  K3 = feval(f, t(i)+.5*h, y(:,i)+.5*h*K2);
  K4 = feval(f, t(i)+h,    y(:,i)+h*K3   );

  % Evaluate approximate solution at next step
  y(:,i+1) = y(:,i) + h/6 *(K1+2*K2+2*K3+K4);
end
```

A script that employs our function `rk4` is given in Example 16.8.

**Note:** The appearance of methods of different orders in this section motivates exploring the question of testing whether a particular method at least comes close to reflecting its order in a given application. A way to go about this is the following. If the error at fixed $t$ is $e(h) \approx \gamma h^q$, with $\gamma$ depending on $t$ but not on $h$, then with step size $2h$, $e(2h) \approx \gamma(2h)^q \approx 2^q e(h)$. Thus, calculate

$$\text{rate}(h) = \log_2\left(\frac{e(2h)}{e(h)}\right).$$

This **observed order**, or **rate**, is compared to the predicted order $q$ of the given method. See also Exercise 9 for a generalization.

**Example 16.7.** Consider the scalar problem

$$y' = -y^2, \qquad y(1) = 1.$$

The exact solution is $y(t) = \frac{1}{t}$. We compute and list absolute errors at $t = 10$ in Table 16.2.

**Table 16.2.** *Errors and calculated observed orders (rates) for the forward Euler, the explicit midpoint (RK2), and the classical Runge–Kutta (RK4) methods.*

| $h$ | Euler | Rate | RK2 | Rate | RK4 | Rate |
|---|---|---|---|---|---|---|
| 0.2 | 4.7e-3 | | 3.3e-4 | | 2.0e-7 | |
| 0.1 | 2.3e-3 | 1.01 | 7.4e-5 | 2.15 | 1.4e-8 | 3.90 |
| 0.05 | 1.2e-3 | 1.01 | 1.8e-5 | 2.07 | 8.6e-10 | 3.98 |
| 0.02 | 4.6e-4 | 1.00 | 2.8e-6 | 2.03 | 2.2-11 | 4.00 |
| 0.01 | 2.3e-4 | 1.00 | 6.8e-7 | 2.01 | 1.4e-12 | 4.00 |
| 0.005 | 1.2e-4 | 1.00 | 1.7e-7 | 2.01 | 8.7e-14 | 4.00 |
| 0.002 | 4.6e-5 | 1.00 | 2.7e-8 | 2.00 | 1.9e-15 | 4.19 |

By using the approach described above for computing observed orders, we see that indeed the three methods introduced demonstrate orders 1, 2, and 4, respectively. Note that for $h$ very small, roundoff error effects show up in the error with the more accurate formula RK4. Given the cost per step, a fair comparison with roughly equal computational effort would be of Euler with $h = .005$, RK2 with $h = .01$, and RK4 with $h = .02$. Clearly the higher order methods are better if an accurate approximation (say, error below $10^{-7}$) is sought.

This example notwithstanding, let us bear in mind that for lower accuracy requirements, or for rougher ODEs, lower order methods may become more competitive. ∎

**Example 16.8.** Next, we unleash our function `rk4` on the ODE problem given by

$$y_1' = .25y_1 - .01y_1y_2, \quad y_1(0) = 80,$$
$$y_2' = -y_2 + .01y_1y_2, \quad y_2(0) = 30.$$

Integrating from $a = 0$ to $b = 100$ with step size $h = 0.01$, the resulting solution is displayed in Figure 16.5.

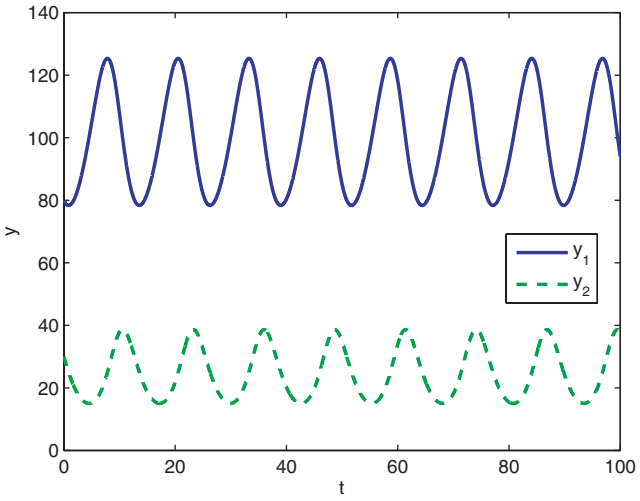



**Figure 16.5.** *Predator-prey model: $y_1(t)$ is number of prey, $y_2(t)$ is number of predator.*

Here is the MATLAB script used for this example, yielding both Figure 16.5 and Figure 16.6.

```
y0 = [80,30]';   % initial data
tspan = [0,100]; % integration interval
h = .01;         % constant step size

[tout,yout] = rk4(@func,tspan,y0,h);

figure(1)
plot(tout,yout)
xlabel('t')
ylabel('y')
legend('y_1','y_2')

figure(2)
plot(yout(1,:),yout(2,:))
xlabel('y_1')
ylabel('y_2')

function f = func(t,y)

a = .25; b = -.01; c = -1; d = .01;
f(1) = a*y(1) + b*y(1)*y(2);
f(2) = c*y(2) + d*y(1)*y(2);
```

This is a simple *predator-prey* model, originally considered independently by A. Lotka and V. Volterra. There is one prey species whose number at any given time is $y_1(t)$. The number of prey grows unboundedly in time if unhunted by the predator. There is only one predator species whose number at any given time is $y_2(t)$. The number of predators would shrink to extinction if they do not encounter prey. But they do, and thus a life cycle forms. Note the way the peaks and lows in $y_1(t)$ are related to those of $y_2(t)$.

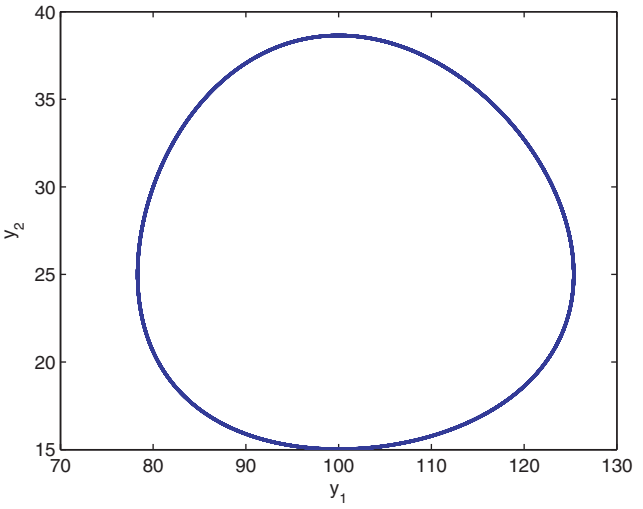



**Figure 16.6.** *Predator-prey solution in phase plane: the curve of $y_1(t)$ vs. $y_2(t)$ yields a limit cycle.*

In Figure 16.6 we plot $y_1$ vs. $y_2$. A *limit cycle* is obtained, suggesting that at least according to this rather simple model neither species will become extinct or grow unboundedly at any future time. ∎

**Note:** It is often convenient to suppress the explicit dependence of $\mathbf{f}(t,\mathbf{y})$ on $t$ and to write the ODE (16.1a) as

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}).$$

Indeed, there are many problem instances, such as those in Examples 16.2–16.8, where there is no explicit dependence on $t$ in $\mathbf{f}$.

Even if there is such dependence, it is possible to imagine $t$ as yet another dependent variable, i.e., define a new independent variable $x = t$, and let $\tilde{\mathbf{y}}^T = (\mathbf{y}^T, t)$ and $\tilde{\mathbf{f}}^T = (\mathbf{f}^T, 1)$. Then $\mathbf{y}' = \mathbf{f}(t,\mathbf{y})$ becomes

$$\frac{d\tilde{\mathbf{y}}}{dx} = \tilde{\mathbf{f}}(\tilde{\mathbf{y}}).$$

We are not proposing to actually carry out such a transformation, only to imagine it in case you don't know how to handle $t$ when implementing a particular discretization formula.

### General *s*-stage RK methods

In the explicit RK methods we have seen so far, each internal stage $Y_j$ depends on the previous $Y_{j-1}$. More generally, each internal stage $Y_j$ depends on all previously computed stages $Y_k$, so an **explicit, *s*-stage RK** method looks like

$$y_{i+1} = y_i + h\sum_{j=1}^{s} b_j f(Y_j), \quad \text{where}$$

$$Y_j = y_i + h\sum_{k=1}^{j-1} a_{jk} f(Y_k).$$

Here we assume no explicit dependence of $f$ on the independent variable $t$, to save on notation. For instance, the explicit midpoint method is written in this form with $s = 2$, $a_{11} = a_{12} = a_{22} = 0$, $a_{21} = 1/2$, $b_1 = 0$, and $b_2 = 1$.

The even more general **implicit, $s$-stage RK** method is written as

$$y_{i+1} = y_i + h\sum_{j=1}^{s} b_j f(Y_j), \quad \text{where}$$

$$Y_j = y_i + h\sum_{k=1}^{s} a_{jk} f(Y_k).$$

The implicit midpoint method is an instance of this, with $s = 1$, $a_{11} = 1/2$, and $b_1 = 1$. Implicit RK methods become interesting when considering stiff problems; see Section 16.5.

It is common to gather the coefficients of an RK method into a *tableau*

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

where $c_j = \sum_{k=1}^{s} a_{jk}$ for $j = 1, 2, \ldots, s$. Thus, the explicit midpoint method is written as

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$$

and the implicit midpoint method is

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

In Exercise 10 you are asked to construct corresponding tableaus for the explicit and the implicit trapezoidal methods, both of which satisfy $s = 2$. In general the method is explicit if and only if $a_{j,k} = 0$ for all $j \le k$. (Why?)

The definitions of local truncation error and order, as well as the proof of convergence and global error bound given in Section 16.2, readily extend for any RK method.

Some of the advantages and disadvantages of RK methods are evident by now. Advantages include:

- Simplicity in concept and in starting the integration process.
- Flexibility in varying the step size.
- Flexibility in handling discontinuities in $f(t, y)$ and other events (e.g., collision of bodies whose motion is being simulated).

Disadvantages of RK methods include:

- The number of function evaluations required for higher order RK methods—our rough measure of work expense—is relatively high, compared to multistep methods.
- Deriving higher order RK methods and proving their order of accuracy can be challenging.
- There are difficulties with adaptive order variation. So, in practice people settle on one method of a certain order and vary only the step size to achieve error control, as described in Section 16.6.
- More involved and possibly more costly procedures are required for stiff problems, the topic of Section 16.5, than for multistep methods.

*Specific exercises for this section*: Exercises 6–11.

## 16.4 Multistep methods

**Note:** It is possible to read almost all of Sections 16.5 and 16.6 without first going through Section 16.4 in detail.

Linear multistep methods are the traditional rival family to RK methods. While the latter are based on repeated evaluations of the function $f(t,y)$ within one step, the methods considered here are simpler in form but use past information as well.

The basic idea behind them is very simple. Observe that at the start of step $i$ we usually have knowledge not only of the approximate solution $y_i$ at $t_i$ but also of previous solution values: $y_{i-1}$ at $t_{i-1}$, $y_{i-2}$ at $t_{i-2}$, and so on. So, use polynomial interpolation of these values, or of their corresponding values of $f(t,y)$, in order to obtain cheap yet accurate approximations for the next unknown, $y_{i+1}$ at $t_{i+1}$. See Figure 16.7.

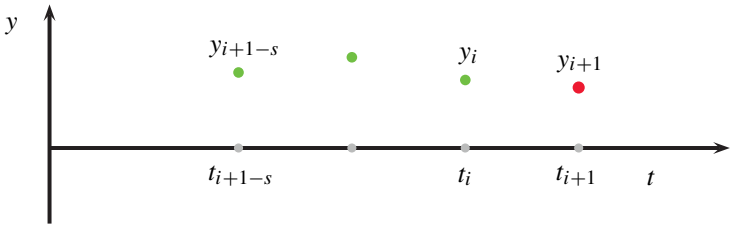



**Figure 16.7.** *Multistep method: known solution values of $y$ or $f(t,y)$ at the current location $t_i$ and previous locations $t_{i-1},\ldots,t_{i+1-s}$ are used to form an approximation for the next unknown $y_{i+1}$ at $t_{i+1}$. (Reprinted from Ascher* [3].*)*

### General form of a linear multistep method

Assuming for simplicity a uniform step size $h$, so that

$$t_i = a + ih, i = 0,1,\ldots,$$

an $s$-step method reads

$$\sum_{j=0}^{s} \alpha_j y_{i+1-j} = h \sum_{j=0}^{s} \beta_j f_{i+1-j}.$$

Here, $f_{i+1-j} = f(t_{i+1-j}, y_{i+1-j})$ and $\alpha_j, \beta_j$ are coefficients which specify the actual multistep method; let us set $\alpha_0 = 1$ for definiteness, because the entire formula can obviously be rescaled.

The need to evaluate $f$ at the unknown point $y_{i+1}$ depends on $\beta_0$: the method is **explicit** if $\beta_0 = 0$ and **implicit** otherwise. So, for the explicit variant we have the solution at the next step defined by known quantities as

$$y_{i+1} = -\sum_{j=1}^{s} \alpha_j y_{i+1-j} + h \sum_{j=1}^{s} \beta_j f_{i+1-j},$$

whereas the implicit formula can be written as

$$y_{i+1} - h\beta_0 f(t_{i+1}, y_{i+1}) = -\sum_{j=1}^{s} \alpha_j y_{i+1-j} + h \sum_{j=1}^{s} \beta_j f_{i+1-j},$$

with all the unknown terms gathered at the left-hand side.

Such methods are called *linear* because, unlike general RK, the expression in the multistep formula is linear in $f$. Note that $f$ itself may still be nonlinear in $y$.

**Example 16.9.** The forward Euler method is a particular instance of a linear multistep method, with $s = 1$, $\alpha_1 = -1$, $\beta_0 = 0$, and $\beta_1 = 1$.

The backward Euler method is also a particular instance of a linear multistep method, with $s = 1$, $\alpha_1 = -1$, $\beta_0 = 1$, and $\beta_1 = 0$. Note that $\beta_0 = 0$ for the explicit method and $\beta_0 \neq 0$ for the implicit one.

The other explicit RK methods that we have seen in Section 16.3 are not linear multistep methods. ∎

Let us define the **local truncation error** for the linear multistep method to be

$$d_i = h^{-1} \sum_{j=0}^{s} \alpha_j y(t_{i+1-j}) - \sum_{j=0}^{s} \beta_j y'(t_{i+1-j}).$$

As before, this is the amount by which the exact solution fails to satisfy the difference equation, divided by $h$. Thus, the method has **accuracy order** $q$ if

$$d_i = \mathcal{O}(h^q)$$

for all problems with sufficiently smooth exact solutions $y(t)$.

The most popular families of linear multistep methods are the *Adams family* and the *backward differentiation formula* (BDF) *family*.

### The Adams family

The Adams methods are derived by considering integration of the ODE over the most recent subinterval, yielding

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt$$

and then approximating the integrand $f(t, y)$ by an interpolating polynomial through previously computed values of $f(t_l, y_l)$. In the general form of multistep methods we therefore set, for all Adams methods, the $\alpha_j$ values to be

$$\alpha_0 = 1, \alpha_1 = -1, \quad \text{and} \quad \alpha_j = 0,\ j > 1.$$

The $s$-step *explicit Adams* method, also called the **Adams–Bashforth** method, is obtained by interpolating $f$ through the previous points $t = t_i, t_{i-1}, \ldots, t_{i+1-s}$, thus giving an explicit method. Since there are $s$ interpolation points which are $\mathcal{O}(h)$ apart we expect (and obtain) order of accuracy $q = s$.

**Example 16.10.** With $s = 2$ we interpolate $f$ at the points $t_i$ and $t_{i-1}$. This gives the straight line

$$p(t) = f_i + \frac{f_i - f_{i-1}}{h}(t - t_i),$$

which is then integrated to yield

$$\begin{aligned}
&\int_{t_i}^{t_{i+1}} \left( f_i + \frac{f_i - f_{i-1}}{h}(t - t_i) \right) dt \\
&= \left[ f_i t + \frac{f_i - f_{i-1}}{2h}(t - t_i)^2 \right]_{t_i}^{t_{i+1}} \\
&= h \left( \frac{3 f_i}{2} - \frac{f_{i-1}}{2} \right).
\end{aligned}$$

Therefore, the obtained formula is

$$y_{i+1} = y_i + \frac{h}{2}(3 f_i - f_{i-1}).$$

This is the two-step Adams–Bashforth method.

The four-step and five-step Adams–Bashforth methods can be similarly derived using polynomial interpolation of degrees 3 and 4, respectively. Any of the representations described in Sections 10.2–10.4 would do. Integration from $t_i$ to $t_{i+1}$ follows. The obtained methods (of orders 4 and 5) are

$$y_{i+1} = y_i + \frac{h}{24}(55 f_i - 59 f_{i-1} + 37 f_{i-2} - 9 f_{i-3}) \quad \text{and}$$

$$y_{i+1} = y_i + \frac{h}{720}(1901 f_i - 2774 f_{i-1} + 2616 f_{i-2} - 1274 f_{i-3} + 251 f_{i-4}).$$

For $s = 1$ we obtain the forward Euler method. ∎

The $s$-step *implicit Adams* method, also called **Adams–Moulton** method, is obtained by interpolating $f$ through the previous points plus the next one, $t = t_{i+1}, t_i, t_{i-1}, \ldots, t_{i+1-s}$, giving an implicit method. Since there are $s + 1$ interpolation points which are $\mathcal{O}(h)$ apart we expect (and obtain) order of accuracy $q = s + 1$.

**Example 16.11.** Interpolating $(t_{i+1}, f_{i+1})$ by a constant and integrating, we clearly obtain the backward Euler method ($s = q = 1$).

Passing a straight line through $(t_i, f_i)$ and $(t_{i+1}, f_{i+1})$ and integrating, we get, just like in Section 15.1, the implicit trapezoidal method ($s = 1, q = 2$). Thus, there are two one-step methods in the Adams–Moulton family. Life gets more interesting only with $s > 1$.

Exercise 12 asks you to derive the two-step formula by passing a quadratic through the points $(t_{i+1}, f_{i+1})$, $(t_i, f_i)$, and $(t_{i-1}, f_{i-1})$. The three-step and four-step Adams–Moulton methods can be similarly derived using polynomial interpolation of degrees 3 and 4, respectively. Integration from

$t_i$ to $t_{i+1}$ follows, and the obtained methods (of orders 4 and 5) are

$$y_{i+1} = y_i + \frac{h}{24}(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}) \quad \text{and}$$

$$y_{i+1} = y_i + \frac{h}{720}(251f_{i+1} + 646f_i - 264f_{i-1} + 106f_{i-2} - 19f_{i-3}).$$

Compare these formulas to their explicit counterparts in Example 16.10. ∎

**Example 16.12.** For the scalar problem

$$y' = -y^2, \qquad y(1) = 1,$$

the exact solution is $y(t) = \frac{1}{t}$. Tables 16.3 and 16.4 list absolute errors at $t = 10$. These should be compared against the RK methods demonstrated in Table 16.2 of Example 16.7. For the additional initial values necessary we use the exact solution, excusing this form of cheating by claiming that we are interested here only in the global error behavior.

**Table 16.3.** *Example* 16.12*: errors and calculated rates for Adams–Bashforth methods;* $(s,q)$ *denotes the* $s$*-step method of order* $q$*.*

| Step $h$ | $(1,1)$ error | Rate | $(2,2)$ error | Rate | $(4,4)$ error | Rate |
|---|---|---|---|---|---|---|
| 0.2 | 4.7e-3 | | 9.3e-4 | | 1.6e-4 | |
| 0.1 | 2.3e-3 | 1.01 | 2.3e-4 | 2.02 | 1.2e-5 | 3.76 |
| 0.05 | 1.2e-3 | 1.01 | 5.7e-5 | 2.01 | 7.9e-7 | 3.87 |
| 0.02 | 4.6e-4 | 1.00 | 9.0e-6 | 2.01 | 2.1e-8 | 3.94 |
| 0.01 | 2.3e-4 | 1.00 | 2.3e-6 | 2.00 | 1.4e-9 | 3.97 |
| 0.005 | 1.2e-4 | 1.00 | 5.6e-7 | 2.00 | 8.6e-11 | 3.99 |
| 0.002 | 4.6e-5 | 1.00 | 9.0e-8 | 2.00 | 2.2e-12 | 3.99 |

**Table 16.4.** *Example* 16.12*: errors and calculated rates for Adams–Moulton methods;* $(s,q)$ *denotes the* $s$*-step method of order* $q$*.*

| Step $h$ | $(1,1)$ error | Rate | $(1,2)$ error | Rate | $(3,4)$ error | Rate |
|---|---|---|---|---|---|---|
| 0.2 | 6.0e-3 | | 1.8e-4 | | 1.1e-5 | |
| 0.1 | 2.4e-3 | 1.35 | 4.5e-5 | 2.00 | 8.4e-7 | 3.73 |
| 0.05 | 1.2e-3 | 1.00 | 1.1e-5 | 2.00 | 5.9e-8 | 3.85 |
| 0.02 | 4.6e-4 | 1.00 | 1.8e-6 | 2.00 | 1.6e-9 | 3.92 |
| 0.01 | 2.3e-4 | 1.00 | 4.5e-7 | 2.00 | 1.0e-10 | 3.97 |
| 0.005 | 1.2e-4 | 1.00 | 1.1e-7 | 2.00 | 6.5e-12 | 3.98 |
| 0.002 | 4.6e-5 | 1.00 | 1.8e-8 | 2.00 | 1.7e-13 | 3.99 |

**Theorem: Multistep Method Order.**
Let

$$C_0 = \sum_{j=0}^{s} \alpha_j,$$

$$C_i = (-1)^i \left[ \frac{1}{i!} \sum_{j=1}^{s} j^i \alpha_j + \frac{1}{(i-1)!} \sum_{j=0}^{s} j^{i-1} \beta_j \right], \quad i = 1, 2, \ldots.$$

Then the linear multistep method has order $p$ if and only if

$$C_0 = C_1 = \cdots = C_p = 0,\ C_{p+1} \neq 0.$$

Furthermore, the local truncation error is then given by

$$d_i = C_{p+1} h^p y^{(p+1)}(t_i) + \mathcal{O}(h^{p+1}).$$

We can see that the observed order of the methods is as advertised. The error constant (i.e., $\gamma$ in $e(h) \approx \gamma h^q$) is smaller for the Adams–Moulton method of order $q > 1$ than the corresponding Adams–Bashforth method of the same order. This is not surprising: the $s$ interpolation points are more centered with respect to the interval $[t_i, t_{i+1}]$, where the ensuing integration takes place.

The error constant in RK2 is comparable to that of the implicit trapezoidal method in Table 16.4. The error constant in RK4 is smaller than that in the corresponding fourth order methods here. (Can you intuitively explain why?) ∎

A pleasant theoretical advantage of linear multistep methods is that deriving such a method to an arbitrarily high, proven order is straightforward; this is unlike the situation for RK methods. In fact, the rather general theorem given on the current page can be proved in a straightforward way using Taylor expansions of $y(t - jh)$ and $y'(t - jh)$ about $t$. The theorem gives not only conditions for the method's order but also the leading term of the truncation error.

**Example 16.13.** For the two-step Adams–Bashforth formula we have $C_1 = -(-1 + 3/2 - 1/2) = 0$, $C_2 = 1/2(-1) + 3/2 - 1 = 0$, $C_3 = -[1/6(-1) + 1/2(3/2 - 2)] = \frac{5}{12}$.

Hence the local truncation error is

$$d_i = \frac{5}{12} h^2 y'''(t_i).$$

See also Exercise 15. ∎

A far less pleasant practical requirement for an $s$-step method is that of supplying *startup values*: in principle all $s$ initial values $y_0, y_1, \ldots, y_{s-1}$ must be $\mathcal{O}(h^q)$-accurate for a method of order $q$, and this is an issue when $s > 1$. There are various ways for obtaining the additional initial values $y_1, \ldots, y_{s-1}$ (whereas $y_0 = c$ is specified as usual). This typically involves using another method which requires fewer initial values. Moreover, abandoning the notion of accuracy order in favor of maintaining an a posteriori error estimate under control, sufficiently accurate initial values may be obtained using a lower order method with a smaller step size.

### Backward differentiation formulas (BDF)

Here is another family of linear multistep methods. The real power of this family will only be realized in the next section.

The $s$-step BDF method is obtained by evaluating $f$ only at the right end of the current step, $(t_{i+1}, y_{i+1})$, driving an interpolating polynomial of $y$ (rather than $f$) through the points $t = t_{i+1}, t_i, t_{i-1}, \ldots, t_{i+1-s}$, and differentiating it. This gives an implicit method of accuracy order $q = s$. Table 16.5 gives the coefficients of the BDF methods for $s$ up to 6.[61] For $s = 1$ we obtain again the backward Euler method.

**Table 16.5.** *Coefficients of BDF methods up to order* 6.

| $s$ | $\beta_0$ | $\alpha_0$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ | $\alpha_5$ | $\alpha_6$ | $q$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | $-1$ | | | | | | 1 |
| 2 | $\frac{2}{3}$ | 1 | $-\frac{4}{3}$ | $\frac{1}{3}$ | | | | | 2 |
| 3 | $\frac{6}{11}$ | 1 | $-\frac{18}{11}$ | $\frac{9}{11}$ | $-\frac{2}{11}$ | | | | 3 |
| 4 | $\frac{12}{25}$ | 1 | $-\frac{48}{25}$ | $\frac{36}{25}$ | $-\frac{16}{25}$ | $\frac{3}{25}$ | | | 4 |
| 5 | $\frac{60}{137}$ | 1 | $-\frac{300}{137}$ | $\frac{300}{137}$ | $-\frac{200}{137}$ | $\frac{75}{137}$ | $-\frac{12}{137}$ | | 5 |
| 6 | $\frac{60}{147}$ | 1 | $-\frac{360}{147}$ | $\frac{450}{147}$ | $-\frac{400}{147}$ | $\frac{225}{147}$ | $-\frac{72}{147}$ | $\frac{10}{147}$ | 6 |

**Example 16.14.** Continuing with Example 16.12, we now compute errors for the same ODE problem using three BDF methods.

The results in Table 16.6 are clearly comparable to those in Tables 16.3 and 16.4, although the error constants for methods of similar order are worse here. ∎

**Table 16.6.** *Example* 16.14*: errors and calculated rates for BDF methods;* $(s,q)$ *denotes the* $s$*-step method of order* $q$*.*

| Step $h$ | $(1,1)$ error | Rate | $(2,2)$ error | Rate | $(4,4)$ error | Rate |
|---|---|---|---|---|---|---|
| 0.2 | 6.0e-3 | | 7.3e-4 | | 7.6e-5 | |
| 0.1 | 2.4e-3 | 1.35 | 1.8e-4 | 2.01 | 6.1e-6 | 3.65 |
| 0.05 | 1.2e-3 | 1.00 | 4.5e-5 | 2.00 | 4.3e-7 | 3.81 |
| 0.02 | 4.6e-4 | 1.00 | 7.2e-6 | 2.00 | 1.2e-8 | 3.91 |
| 0.01 | 2.3e-4 | 1.00 | 1.8e-6 | 2.00 | 7.8e-10 | 3.96 |
| 0.005 | 1.2e-4 | 1.00 | 4.5e-7 | 2.00 | 4.9e-11 | 3.98 |
| 0.002 | 4.6e-5 | 1.00 | 1.8e-8 | 2.00 | 1.3e-12 | 3.99 |

## Predictor-corrector methods

The great practical advantage of higher order multistep methods is their cost per step in terms of function evaluations. For instance, only one function evaluation is required to advance the explicit Adams–Bashforth formula by one step. However, for the higher order Adams–Bashforth methods there are some serious limitations in terms of absolute stability, so much so that they are usually not employed as stand-alone discretizations with $s > 2$.

[61] We do not consider BDF with $s > 6$ because then these methods become unstable in a basic way that simply cannot happen for RK methods and will not be discussed further in this text.

For the implicit Adams–Moulton methods, which are naturally more accurate and more stable than the corresponding explicit methods of the same order, we need to solve nonlinear algebraic equations for $y_{i+1}$. Worse, for ODE systems we generally have a nonlinear system of algebraic equations at each step. So, are we ahead yet?

Fortunately, everything implicit and nonlinear is multiplied by $h$ in the implicit ODE method, because only $y$ and not $y'$ may appear nonlinearly in the ODE. Thus, for $h$ sufficiently small a simple **fixed point iteration** of the form considered in Sections 3.3 and 7.3 converges under fairly mild conditions. This works in practice unless the ODE system is *stiff*; see Section 16.5 for the latter case.

To start the fixed point iteration for a given $s$-step Adams–Moulton formula we need a starting iterate, and since all those previous values of $f$ are stored anyway we may well use them to apply the corresponding Adams–Bashforth formula. This explicit formula yields a *predicted* value for $y_{i+1}$, which is then *corrected* by the fixed-point iteration based on Adams–Moulton.

But next, note that the fixed point iteration need not be carried to convergence: all it yields in the end is $y_{i+1}$, which is just an approximation for $y(t_{i+1})$ anyway. Applying only a fixed number of such iterations creates a new, explicit, *predictor-corrector* method. The most popular predictor-corrector variant, denoted PECE, carries out only one fixed point iteration. The algorithm is given below.

**Algorithm: Predictor-Corrector Step (PECE).**
At step $i$, given $y_i, f_i, f_{i-1}, \ldots, f_{i+1-s}$:

1. Use an $s$-step Adams–Bashforth method to calculate $y^0_{i+1}$, calling the result the Predicted value.
2. Evaluate $f^0_{i+1} = f(t_{i+1}, y^0_{i+1})$.
3. Apply an $s$-step Adams–Moulton method using $f^0_{i+1}$ for the unknown, calling the result the Corrected value $y_{i+1}$.
4. Evaluate $f_{i+1} = f(t_{i+1}, y_{i+1})$.

The last evaluation in the PECE algorithm is carried out in preparation for the next time step and to maintain an acceptable measure of absolute stability, a concept further discussed in Section 16.5.

**Example 16.15.** Combining the two-step Adams–Bashforth formula with the second order one-step Adams–Moulton formula (i.e., the implicit trapezoidal method), we obtain the following method for advancing one time step.

Given $y_i, f_i, f_{i-1}$, set

1. $y^0_{i+1} = y_i + \frac{h}{2}(3f_i - f_{i-1})$,
2. $f^0_{i+1} = f(t_{i+1}, y^0_{i+1})$,
3. $y_{i+1} = y_i + \frac{h}{2}(f_i + f^0_{i+1})$,
4. $f_{i+1} = f(t_{i+1}, y_{i+1})$.

This is an explicit, second order method which has the local truncation error

$$d_i = -\frac{1}{12}h^2 y'''(t_{i+1}) + \mathcal{O}(h^3).$$

This method looks much like an explicit trapezoidal variant of an RK method. The predictor-corrector approach really shines when more steps are involved, because the same simplicity—and cost of two function evaluations per step!—holds for higher order methods, whereas higher order RK methods get significantly more complex and expensive per step. ∎

In the PECE algorithm given on the facing page the order of accuracy is $s+1$, one higher than that of the predictor. This allows for local error estimation. Error control in the spirit of Section 16.6 is then facilitated. Another common situation is where the orders of the predictor formula and of the corrector formula are the same. In the latter case the principal term of the local truncation error for the PECE method is the same as that of the corrector. It is then possible to estimate the local error in a very simple manner.

In fact, it is also possible to adaptively vary the method order, $q = s$, of the PECE pair. However, varying the step size is more complicated, though certainly possible, than in the case of the one-step RK methods.

**Note:** Comparing multistep methods to RK methods, the comments made at the end of Section 16.3 are relevant here. Briefly, the important advantages of the multistep family are the cheap high order PECE pairs with the local error estimation that comes for free.

The important disadvantages are the need for additional startup procedure and the relatively cumbersome adjustment to local changes such as lower continuity, event location, and drastically adapting the step size.

In the 1980s, linear multistep methods were the methods of choice for most general-purpose ODE codes. With less emphasis on cost and more on flexibility, however, RK methods have taken the popularity lead since. This is true except for stiff problems, for which BDF methods are still popular.

*Specific exercises for this section*: Exercises 12–15.

## 16.5 Absolute stability and stiffness

The methods described in Sections 16.3 and 16.4 are routinely used in many applications, and they often give satisfactory results in practice. An exception is the case of stiff problems, where explicit RK methods are forced to use a very small step size and thus become unreasonably expensive. Adams–Bashforth and Adams predictor-corrector multistep methods meet a similar fate. But before turning to stiffness we reintroduce absolute stability, in a more general context than in Section 16.2.

### Complex-coefficient test equation and absolute stability region

In this section we consider the *test equation*

$$y' = \lambda y,$$

where $\lambda$ is a complex scalar. The exact solution is still $y(t) = e^{\lambda t} y(0)$ (compare to page 491). The solution magnitude depends only on the real part of $\lambda$: we have

$$|y(t)| = e^{\Re(\lambda)t} |y(0)|.$$

So $|y(t_{i+1})| \le |y(t_i)|$, i.e., the solution decays in magnitude as $t$ grows, if $\Re(\lambda) \le 0$.

For the forward Euler method the absolute stability condition still reads

$$|1 + h\lambda| \le 1,$$

but the meaning is different from that in Section 16.2 because $z = h\lambda$ now roams in the *complex plane*, and the absolute stability condition is particularly important in the left half of that plane, namely, $\Re(z) \le 0$. The condition $|1+z| \le 1$ in fact describes a disk of radius 1 in the $z$-plane, centered at $z = (-1,0)$ and depicted in red in Figure 16.8. Using a constant step size $h$, we must choose it so that $z$ stays inside that red disk.

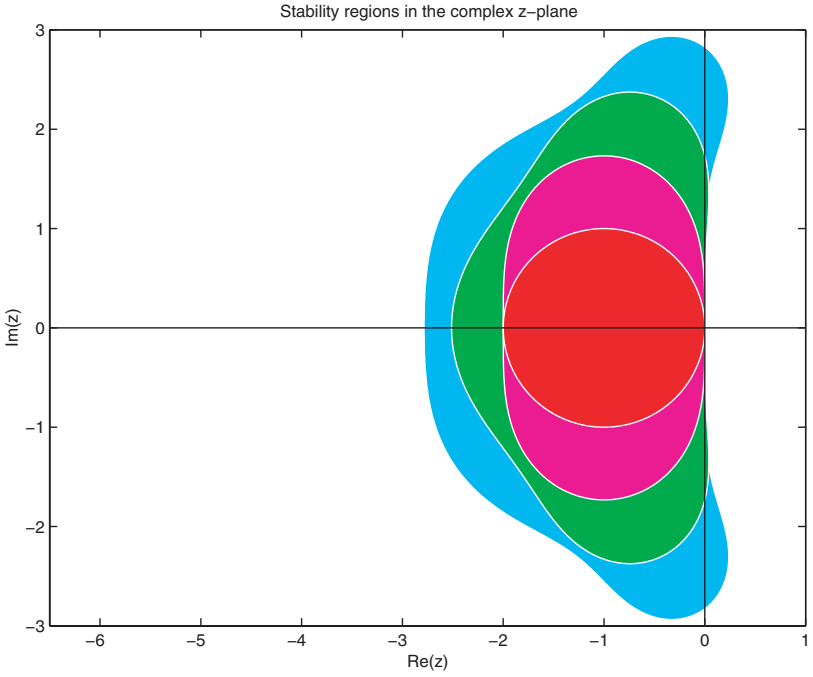



**Figure 16.8.** *Stability regions for $q$-stage explicit RK methods of order $q$, $q = 1,2,3,4$. The inner circle corresponds to forward Euler, $q = 1$. The larger $q$ is, the larger the stability region. Note the "ear lobes" of the fourth-order method protruding into the right half plane.*

Is the test equation too simple to be of practical use?

In general we must consider the nonlinear ODE system (16.1a) given on page 482. Here, absolute stability properties are determined by the Jacobian matrix, defined by

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \cdots & \frac{\partial f_1}{\partial y_m} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_m}{\partial y_1} & \cdots & \cdots & \frac{\partial f_m}{\partial y_m} \end{pmatrix}.$$

Why is that so? Because stability is all about the propagation in time $t$ of perturbations to the solution. So, if our solution $\mathbf{y}(t)$ is a perturbation of a nearby function $\hat{\mathbf{y}}(t)$ satisfying the same differential equation, then we should consider the progress of their difference

$$\mathbf{w}(t) = \mathbf{y}(t) - \hat{\mathbf{y}}(t).$$

Expanding in (what else!) Taylor series in $m$ variables (see page 253), we can write

$$\mathbf{f}(\mathbf{y}) = \mathbf{f}(\hat{\mathbf{y}}) + J(\hat{\mathbf{y}})\mathbf{w} + \mathcal{O}\|\mathbf{w}\|^2,$$

so the perturbation difference obeys the ODE

$$\mathbf{w}' = J(\hat{\mathbf{y}})\mathbf{w} + \mathcal{O}\|\mathbf{w}\|^2 \approx J(\hat{\mathbf{y}})\mathbf{w}.$$

Hence we obtain, for small perturbations, an ODE system that looks like the test equation with the Jacobian matrix $J$ in place of $\lambda$.

A surprisingly good practical estimate of the stability properties of a numerical method for the general nonlinear ODE system is obtained by considering the test equation for each of the eigenvalues of the Jacobian matrix. This is also the reason, though, why we must consider a complex scalar $\lambda$: eigenvalues are in general complex scalars. Thus, instead of an absolute stability bound on $z = h\lambda$ as in Section 16.2, we are really looking at an **absolute stability region** in the complex plane.

Let us consider next the application of the explicit trapezoidal method to the test equation. We have $f(y) = \lambda y$, and our task is to find $R(z)$ such that $y_{i+1} = R(z)y_i$. For the first stage we get again

$$Y = (1+z)y_i,$$

and then

$$y_{i+1} = y_i + \frac{z}{2}(y_i + Y) = \left[1 + z + \frac{z^2}{2}\right] y_i.$$

Thus, $R(z) = 1 + z + \frac{z^2}{2}$. A similar expression is obtained for the explicit midpoint method. Exercise 17 reveals the larger picture.

Stability regions for explicit $q$-stage RK methods of orders $q$ up to 4 are depicted in Figure 16.8.

**Example 16.16.** Let us return to the mildly nonlinear problem of Example 16.4. The $8 \times 8$ Jacobian matrix is

$$J(\mathbf{y}) = \begin{pmatrix} -1.71 & .43 & 8.32 & 0 & 0 & 0 & 0 & 0 \\ 1.71 & -8.75 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10.03 & .43 & .035 & 0 & 0 & 0 \\ 0 & 8.32 & 1.71 & -1.12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.745 & .43 & .43 & 0 \\ 0 & 0 & 0 & .69 & 1.71 & -280y_8 - .43 & .69 & -280y_6 \\ 0 & 0 & 0 & 0 & 0 & 280y_8 & -1.81 & 280y_6 \\ 0 & 0 & 0 & 0 & 0 & -280y_8 & 1.81 & -280y_6 \end{pmatrix}.$$

The eigenvalues of $J$, like $J$ itself, depend on the solution $\mathbf{y}(t)$. The MATLAB command `eig` reveals that at the initial state $t = 0$, where $\mathbf{y} = \mathbf{y}_0$, the eigenvalues of $J$ up to the first few leading digits are

$$0,\ -10.4841,\ -8.2780,\ -0.2595,\ -0.5058,\ -2.6745 \pm 0.1499\imath,\ -2.3147.$$

There is a conjugate pair of eigenvalues, while the rest are real. To get them all into the disk of absolute stability for the forward Euler method, the most demanding condition is

$$-10.4841\, h > -2,$$

implying that $h = .1$ would be a safe choice.

However, integration of this problem with a constant step size $h = .1$ yields a huge error: the integration process becomes unstable. Good results as in Example 16.4 are obtained by carrying out the integration process with the much smaller $h = .005$.

Indeed it turns out that at $t \approx 10.7$, where things are at about their worst, the eigenvalues equal

$$-211.7697, -10.4841, -8.2780, -2.3923, -2.1400, -0.4907, -3 \times 10^{-5}, -3 \times 10^{-12}.$$

The stability bound that the most negative eigenvalue yields is

$$-211.7697\,h > -2,$$

implying that $h = .005$ would be a safe choice, but even $h > .01$ may not be. ∎

## Stiffness

An intuitive definition of the concept of stiffness is given on the current page. A more precise definition turns out to be surprisingly elusive.

**Stiffness.**
The initial-value ODE problem is **stiff** if the step size needed to maintain absolute stability of the forward Euler method is much smaller than the step size needed to represent the solution accurately.

The problem of Examples 16.4 and 16.16 is stiff. The simple problem of Example 16.5 is even stiffer, intuitively speaking.

From Figure 16.8 we see that increasing the order of an explicit RK method does not do much good when it comes to solving stiff problems. In fact this turns out to be true for any of the explicit methods that we have seen.

For stiff problems, we therefore seek other methods. Implicit methods such as backward Euler, implicit trapezoidal, or implicit midpoint become more attractive then, because their region of absolute stability contains the entire left half $z$-plane. (See Exercise 16.) A method whose domain of absolute stability contains the entire left half plane is called **A-stable**.

**Note:** The case study below considers a PDE and relies directly on Example 7.1. If you are not familiar with the contents of Chapter 7, especially Example 7.1, then your best bet at this point may well be to just skip Example 16.17. Otherwise please don't, because it demonstrates several important issues.

**Example 16.17 (heat equation).** The *heat equation* in its simplest form is written as the PDE

$$\frac{\partial u}{\partial t} = \left( \frac{\partial^2 u}{\partial {x_1}^2} + \frac{\partial^2 u}{\partial {x_2}^2} \right).$$

Here $x_1$ and $x_2$ are spatial variables in the unit square, $0 < x_1, x_2 < 1$, and $t$ is time, $t \geq 0$.

Imagine a square electric blanket that has been heated up to a uniform temperature of 25°. The blanket is located in a room temperature of 0°C (it's a room in the high Andes), and the plug is pulled at the time $t = 0$ when our story begins. The temperature at future times $t > 0$ is governed by the heat equation subject to the *boundary conditions*

$$u(t, x_1, 0) = u(t, x_1, 1) = u(t, 0, x_2) = u(t, 1, x_2) = 0 \quad \forall t \geq 0$$

and the *initial conditions* specifying that $u(0,x_1,x_2) = 25$ for any $0 < x_1, x_2 < 1$. Thus, it is an initial-boundary value problem.

To estimate the heat distribution over the blanket at positive times we first discretize the spatial derivatives as in Example 7.1. Using a uniform step size $\Delta x = 1/M$ in both spatial directions, we are looking for a two-dimensional *grid function*, $\{u_{i,j}(t),\ i = 0,\ldots,M,\ j = 0,\ldots,M\}$, such that $u_{i,j}(t) \approx u(t, i\Delta x, j\Delta x)$. Next, we discretize the second spatial derivatives using a centered three-point formula as in Section 14.1. Thus, holding $x_2 = j\Delta x$ fixed we approximate $\frac{\partial^2 u}{\partial {x_1}^2}$ at $x_1 = i\Delta x$ by $\frac{1}{\Delta x^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$, and likewise holding $x_1 = i\Delta x$ fixed we approximate $\frac{\partial^2 u}{\partial {x_2}^2}$ at $x_2 = j\Delta x$ by $\frac{1}{\Delta x^2}(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})$. The heat equation plus boundary conditions is correspondingly approximated by the ODE system

$$\begin{aligned} &\frac{du_{i,j}}{dt} = \frac{1}{\Delta x^2}(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}), \quad 1 \le i, j \le M-1, \\ &u_{i,j} = 0 \quad \text{otherwise.} \end{aligned}$$

This is a linear system of $m = (M-1)^2$ ODEs, subject to the initial conditions

$$u_{i,j}(0) = 25, \quad 1 \le i, j \le M-1.$$

Furthermore, we can reshape the array of $\{u_{i,j}\}$ into a vector $\mathbf{y}(t)$ and write the ODE system as

$$\mathbf{y}' = A\mathbf{y},$$

the sparse matrix $A$ being the one derived in Example 7.1 with an obvious slight notational modification. Note that for $\Delta x = .01$, say, we obtain $m = 99^2 = 9801$ ODEs. It is not difficult to think of situations where even larger ODE systems are encountered (think three space variables, for one thing).

The next question is how to solve this ODE system. Certainly there is a good incentive here to use an explicit method if possible, thus avoiding the need to solve linear systems. The forward Euler method reads

$$\mathbf{y}_{i+1} = \mathbf{y}_i + hA\mathbf{y}_i, \quad i = 0, 1, \ldots,$$

where $h$ is the step size in time, so at each time step only one matrix-vector multiplication is required.

The eigenvalues of the matrix $A$, which is scaled by $\Delta x^2$ as compared to Example 7.1, are given by

$$\lambda_{l,m} = \frac{1}{\Delta x^2}[4 - 2(\cos(l\pi\Delta x) + \cos(j\pi\Delta x))], \quad 1 \le l, j \le M-1.$$

Thus, $\max \lambda_{l,j} = \lambda_{M-1,M-1} \approx \frac{24}{\Delta x^2}$. (What's important here is the relation to $\Delta x^2$, less so the constant 24.) The absolute stability requirement for the forward Euler method then dictates setting

$$h \le \frac{\Delta x^2}{12}.$$

This can hurt if $\Delta x$ is small. For $\Delta x = .01$ it translates to needing 12,000 time steps just to reach $t = .1$. Note that a similar restriction in essence holds for higher order explicit methods, which works to increase frustration because the separation between what is required for accuracy and what is demanded by stability increases even further.

Instead, consider the implicit trapezoidal method, which in the current context is called the **Crank–Nicolson** scheme. Now there is no absolute stability requirement on $h$, and accuracy considerations suggest setting it in general to be of the same order as $\Delta x$. (In fact, in this particular

application it makes sense to set $h$ smaller at first and letting it grow because the solution gets smoother and more uniform with time; but let's not get into such further refinement.)

For a rough comparison, let us set $h = \Delta x$. Please verify that the trapezoidal, or Crank–Nicolson, method reads

$$\left(I - \frac{h}{2}A\right)\mathbf{y}_{i+1} = \left(I + \frac{h}{2}A\right)\mathbf{y}_i.$$

So, in comparison with the forward Euler method the Crank–Nicolson method will be more efficient provided that the solution of the linear system it requires at each step can be achieved in less that $12/\Delta x$ matrix-vector multiplications. Employing an appropriately preconditioned CG method for this purpose (see Section 7.4) can easily achieve this and much more for a fine resolution. Thus, the implicit method wins!

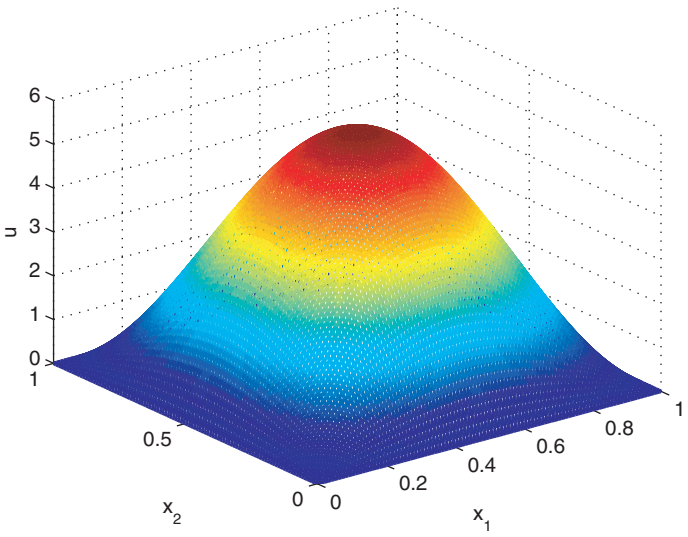



**Figure 16.9.** *Solution of the heat problem of Example* 16.17 *at* $t = 0.1$.

Figure 16.9 depicts the solution at time $t = .1$. The maximum value is $u(.1,.5,.5) \approx 5.63$. The shape of the temperature distribution is similar to that seen at other times $t$, the main difference being the maximum value. For an earlier time, $u(.05,.5,.5) \approx 15$, while $u(.2,.5,.5) < 1$. The temperature is rapidly decreasing everywhere towards a uniform zero level. ∎

### Backward Euler and L-stability

The backward Euler method, briefly introduced in Section 16.2, is of fundamental importance for stiff problems. It is written here for an ODE system (16.1a) as

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h\mathbf{f}(t_{i+1}, \mathbf{y}_{i+1}), \quad i = 1, 2, \ldots.$$

For the test equation we obtain $y_{i+1} = y_i + h\lambda y_{i+1}$, and hence

$$y_{i+1} = \frac{1}{1 - h\lambda} y_i.$$

Therefore, the region of absolute stability is defined by the inequality

$$|1 - z| = |1 - h\lambda| \geq 1.$$

This includes in particular the entire left half $z$-plane, so the method is A-stable. Even more importantly in cases where the real part of $z$ is large and negative, we have $\frac{1}{1-h\lambda} \to 0$ as $|z| \to \infty$. Therefore $y_{i+1} \to 0$, as does the exact solution $y(h)$ under these circumstances. Such a method is called *L-stable*.

BDF methods are L-stable, whereas the implicit trapezoidal and midpoint methods are not.

### Carrying out an integration step

The inherent difficulty in implicit methods is that the unknown solution at the next step, $y_{i+1}$, is defined implicitly. Using the backward Euler method, for instance, we have to solve an often nonlinear equation at each step. This difficulty gets significantly worse when an ODE system is considered. Now there is a nonlinear system of algebraic equations to solve; see Section 9.1. Even if the ODE is linear there is a system of linear equations to solve at each integration step. This can get costly if the size of the system, $m$, is large; recall Example 16.17.

For instance, applying the backward Euler method to our ODE system we are facing at the $i$th step a system of algebraic equations given by

$$\mathbf{g}(\mathbf{y}_{i+1}) \equiv \mathbf{y}_{i+1} - h\mathbf{f}(\mathbf{y}_{i+1}) - \mathbf{y}_i = \mathbf{0}.$$

Unfortunately, the obvious fixed point iteration utilized in Section 16.4 for a predictor-corrector method will not work here unless $h$ is unreasonably small. Fortunately, however, Newton's method as described in Section 9.1 does work also in stiff circumstances. Indeed, with our current notation $J$ for the Jacobian matrix of $\mathbf{f}(\mathbf{y})$, we have the Jacobian matrix $I - hJ$ for an iterative method for $\mathbf{y}_{i+1}$. For a good starting iterate we can take the current approximate solution, $\mathbf{y}_{i+1}^{(0)} = \mathbf{y}_i$. The Newton iteration algorithm is given on this page.

**Algorithm: Newton's Method for Stiff ODEs.**
Set $\mathbf{y}_{i+1}^{(0)} = \mathbf{y}_i$.
for $k = 0, 1, \ldots,$ until convergence

$$\text{solve linear system } \left(I - hJ(\mathbf{y}_{i+1}^{(k)})\right)\mathbf{p}_k = -\mathbf{g}\left(\mathbf{y}_{i+1}^{(k)}\right) \text{ for } \mathbf{p}_k;$$

$$\text{set } \mathbf{y}_{i+1}^{(k+1)} = \mathbf{y}_{i+1}^{(k)} + \mathbf{p}_k.$$

end

All this is to be done at each and every integration step! The saving grace is that the initial guess $\mathbf{y}_{i+1}^{(0)}$ is only $\mathcal{O}(h)$ away from the solution $\mathbf{y}_{i+1}$ of this nonlinear system, so we may expect an $\mathcal{O}(h^2)$ closeness after only one iteration due to the quadratic convergence of Newton's method. Therefore, one such iteration is usually sufficient, at least for the backward Euler method whose local error (see page 516) at each step is $\mathcal{O}(h^2)$ anyway.

Applying one Newton iteration per time step yields the **semi-implicit backward Euler** method

$$(I - hJ(\mathbf{y}_i))\mathbf{y}_{i+1} = (I - hJ(\mathbf{y}_i))\mathbf{y}_i + h\mathbf{f}(\mathbf{y}_i) \quad \text{or}$$
$$\mathbf{y}_{i+1} = \mathbf{y}_i + h(I - hJ(\mathbf{y}_i))^{-1}\mathbf{f}(\mathbf{y}_i).$$

This semi-implicit method still involves the solution of a linear system of equations at each step. Note also that it is no longer guaranteed to be stable for any $z$ in the left half plane. (Can you see why?!)

**Example 16.18.** The problem of Examples 16.4 and 16.16 is stiff, to recall. With a uniform step size the forward Euler method requires $322/.005 = 64,400$ integration steps in Example 16.4.

A solution that is qualitatively similar to the one displayed in Figure 16.3 is obtained by the semi-implicit backward Euler method using $h = 0.1$, resulting in only 3,220 steps. The script is

```
h = 0.1; t = 0:h:322;
y = y0 * ones(1,length(t));
for i = 2:length(t)
  A = eye(8) - h*hiresj(y(:,i-1));
  y(:,i) =  y(:,i-1) + h* A \ hires(y(:,i-1)));
end
plot(t,y)
```

The function `hiresj` returns the Jacobian matrix given in Example 16.16.

Thus, if an accuracy level of around .01 is satisfactory, then for this example the implicit method is much more efficient than the explicit one despite the need to solve a linear system of equations at each integration step. ∎

For higher order methods, required for the efficient computation of high accuracy solutions for stiff problems, BDF are often the methods of choice, although there are also some popular implicit RK methods that can be manipulated to yield competitive performance. The implementation of BDF methods is not much more complicated than backward Euler, except for the need for additional initial values and the cumbersome step size modification typical of all multistep methods. Their behavior, especially the two- and three-step methods, is generally similar to that of backward Euler as well. In sophisticated codes, even if utilizing more accurate methods than backward Euler, usually less than one Jacobian evaluation per step is applied on average.

There is a MATLAB code called `ode23s` for stiff problems. As the name suggests it uses a pair of methods of order 2 and 3. See also Section 16.6 if you wonder why a pair of methods is employed.

### Problems with purely imaginary eigenvalues

Let us shift our attention somewhat. Here we are concerned not with what is usually regarded as a stiff problem, but rather with problems whose Jacobian matrices have purely imaginary eigenvalues. These eigenvalues are such that it is reasonable to select a step size $h$ which yields $|z| = \mathcal{O}(1)$, with $z$ imaginary.

**Example 16.19.** The test equation $y' = \lambda y = \imath\alpha y$, for a real scalar $\alpha \neq 0$, may look artificial. But it arises already when considering the simple ODE

$$u'' + \alpha^2 u = 0.$$

To see this, note that defining $y_1 = u$ and $y_2 = u'/\alpha$, we have the equivalent first order system

$$\mathbf{y}' = A\mathbf{y}, \quad A = \begin{pmatrix} 0 & \alpha \\ -\alpha & 0 \end{pmatrix}.$$

The skew-symmetric matrix $A$ has $\pm\imath\alpha$ for eigenvalues. ∎

In Figure 16.8 we are now looking at behavior along the imaginary (i.e., the vertical) axis. We see that trouble may arise if the forward Euler or explicit trapezoidal or midpoint method is used for

such a problem, because the imaginary axis save the origin is outside the absolute stability regions of these methods. Thus, roundoff error gets amplified by a factor larger than 1 at each time step. The fourth order method RK4 is now much preferred over the simpler explicit methods, not just because of its higher order but because of its superior stability properties!

But, one may well ask, are there interesting practical problems with such imaginary eigenvalues?

Yes, there are, and many. The simplest instance is in Example 16.19 and Exercise 8. A far more complex problem of a similar type is discussed in Example 16.21. Indeed, attempting to reproduce Figure 16.12 using the explicit Euler or explicit midpoint method can prove to be a major frustration builder. Finally, a large class of hyperbolic PDEs with smooth solutions yield such ODE systems upon discretizing spatial derivatives using centered differences; see Section 16.8. The classical RK4 is very popular for the numerical solution of such problems.

*Specific exercises for this section*: Exercises 16–18.

## 16.6 Error control and estimation

Thus far we have assumed that the step size $h$ is constant, independently of the location $t_i$ of the $i$th step. But an examination of the form of a typical truncation error (see especially the Multistep Method Order Theorem on page 504) suggests that to keep the local truncation error roughly the same at each step, thus obtaining a quality approximation efficiently, it makes sense to take $h$ small where the solution varies rapidly, and relatively large where the solution variation is slow. Any modern ODE package varies the step size in order to achieve some error control, and we next describe some of the issues involved.

As with quadrature and the function `quads` described in Section 15.6, it is desirable here to write a mathematical software package where a user would specify only the function $f(t,y)$ and the initial value information $a$ and $c$, as well as a tolerance `tol` and a set of output points $\hat{t}_1,\ldots,\hat{t}_{\hat{N}}$ where the solution is desired. The general-purpose code would subsequently calculate $\{\hat{y}_i\}_{i=1}^{\hat{N}}$, a set of output values such that $\hat{y}_i \approx y(\hat{t}_i)$ accurate to within `tol` for each $i$. Note that the actual mesh $\{t_0,t_1,\ldots,t_N\}$ used by such a code is expected to contain the output points $\hat{t}_i$ as a subset, but the precise values of all the $t_j$'s is not the user's concern. What the user wants are results, efficiently obtained, and the latter desire dictates a quest for a small $N$.

Throughout this section we consider a scalar ODE during the presentation to simplify notation. However, the numerical examples all involve ODE systems. The extension should be clear.

### Types of error control

Despite superficial similarities with adaptive quadrature, however, here the situation is significantly more complex:

- The global error $e_i = y(t_i) - y_i$ does not provide a direct, simple indication on how to choose the next step size $h_i$ such that $t_{i+1} = t_i + h_i$.

- The global error at $t_i$ is not simply a sum of the local errors made at each previous step $j$ for $j = 0,\ldots,i-1$. Indeed, as the error bound that we have derived for the forward Euler method (page 489) indicates, the global error may grow exponentially in time, which means that the contribution of each local error may also grow in time.

The latter concern can occasionally be traumatic; see, for instance, Example 16.21. There are methods to estimate the global error by recalculating the entire solution on the interval $[a,b]$. But more often in practice the global error does not accumulate nastily. Moreover, the user typically

does not have a firm idea of a global error tolerance anyway. Then it is the first concern above that matters, because we still want for efficiency reasons a local error control, i.e., we want to adapt the step size locally! Relating this to the global error is hard, though, so we adjust expectations and are typically content to control the **local error**

$$l_{i+1} = \bar{y}(t_{i+1}) - y_{i+1},$$

where $\bar{y}(t)$ is the solution of the ODE $y' = f(t, y)$ which satisfies $\bar{y}(t_i) = y_i$, but $\bar{y}(0) \neq c$ in general. See Figures 16.10 and 16.2.

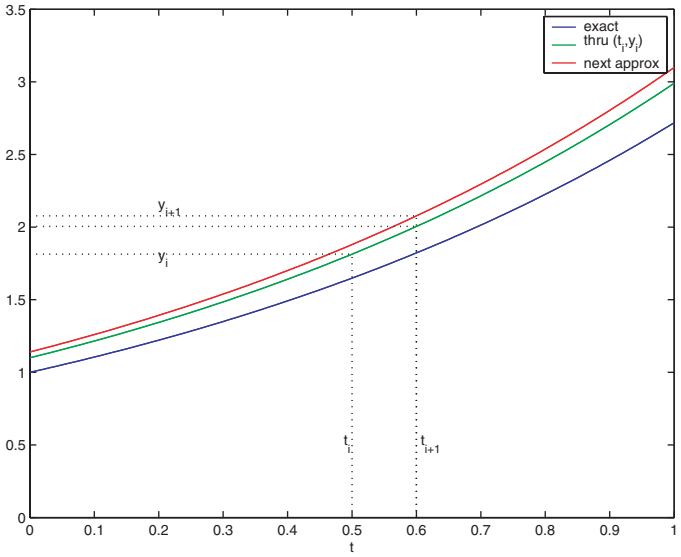



**Figure 16.10.** *The exact solution $y(t_i)$, which lies on the lowest of the three curves, is approximated by $y_i$, which lies on the middle curve. If we integrate the next step exactly, starting from $(t_i, y_i)$, then we obtain $(t_{i+1}, \bar{y}(t_{i+1}))$, which also lies on the middle curve. But we don't: rather, we integrate the next step approximately as well, obtaining $(t_{i+1}, y_{i+1})$, which lies on the top curve. The difference between the two curve values at the argument $t_{i+1}$ is the local error.*

### Adaptive step size selection for local error control

Thus, we consider the $i$th subinterval as if there is no error accumulated at its left end $t_i$ and wonder what might result at the right end $t_{i+1}$. To control the local error we keep the step size small enough, so in particular, $h = h_i$ is no longer the same for all steps $i$.

Suppose now that we use a pair of RK methods, one of order $q$ and the other of order $q+1$, to calculate at $t_{i+1}$ two approximations, both starting from $y_i$ at $t_i$. Let us denote the obtained values by $y_{i+1}$ and $\hat{y}_{i+1}$, respectively. Then we can estimate

$$|l_{i+1}| \approx |\hat{y}_{i+1} - y_{i+1}|.$$

So, at the $i$th step we calculate these two approximations and compare against a given error tolerance. If

$$|\hat{y}_{i+1} - y_{i+1}| \leq h\,\mathtt{tol},$$

then the step is accepted: set $y_{i+1} \leftarrow \hat{y}_{i+1}$, i.e., the more accurate of the two values calculated,[62] and $i \leftarrow i+1$.

If the step is not accepted, then we decrease $h$ to $\tilde{h}$ and repeat the procedure starting from the same $(t_i, y_i)$. This is done as follows. Since the local error in the less accurate method is $l_{i+1} \approx \gamma h^{q+1}$, upon decreasing $h$ to a satisfactory $\tilde{h}$ the local error will become $\gamma \tilde{h}^{q+1} \approx \mu \tilde{h}\,\mathtt{tol}$, where the factor $\mu < 1$ is for safety, say $\mu = 0.9$. Dividing, we get

$$\frac{\tilde{h}^{q+1}}{h^{q+1}} \approx \frac{\mu \tilde{h}\,\mathtt{tol}}{|\hat{y}_{i+1} - y_{i+1}|},$$

which then yields the expression

$$\tilde{h} = h \left( \frac{\mu h\,\mathtt{tol}}{|\hat{y}_{i+1} - y_{i+1}|} \right)^{\frac{1}{q}}.$$

We caution again that the meaning of `tol` here is different from that in the adaptive quadrature case of Section 15.4. Here we attempt to control the local, not the global, error.

How is the value of $h$ selected upon starting the $i$th time step? A reasonable choice is the final step size of the previous time step. But then the sequence of step sizes only decreases and never grows! So, some mechanism must be introduced that allows occasional increase of the starting current step size. For instance, if in the past two time steps no decrease was needed, then we can hazard doubling the initial step size for the current step.

The only question left is how to choose the pair of formulas of orders $q$ and $q+1$ wisely. In the multistep PECE context the answer is obvious. For RK methods this is achieved by searching for a pair of formulas which *share the internal stages* $Y_j$ as much as possible. A good RK pair of orders 4 and 5 would use only 6 (rather than 9 or 10) function evaluations per step. Such pairs of formulas are implemented in MATLAB's routine `ode45`, which carries out a local error control in the spirit described above.

### Two case studies

The rest of this section is devoted to two longish examples, both involving use of the adaptive routine `ode45` in applications. The first of these demonstrates a great advantage to be had, while the second flashes out some warning signals.

**Example 16.20 (astronomical example).** The following classical example from astronomy gives a strong motivation to integrate initial value ODEs with local step size control.

Consider two bodies of masses $\mu = 0.012277471$ and $\hat{\mu} = 1 - \mu$ (earth and sun) in a planar motion, and a third body of negligible mass (moon) moving in the same plane. The motion is governed by the equations

$$\begin{aligned}
u_1'' &= u_1 + 2u_2' - \hat{\mu}\frac{u_1+\mu}{D_1} - \mu\frac{u_1-\hat{\mu}}{D_2},\\
u_2'' &= u_2 - 2u_1' - \hat{\mu}\frac{u_2}{D_1} - \mu\frac{u_2}{D_2},\\
D_1 &= ((u_1+\mu)^2 + u_2^2)^{3/2},\\
D_2 &= ((u_1-\hat{\mu})^2 + u_2^2)^{3/2}.
\end{aligned}$$

[62] We are "cheating" here twice. Once by quietly hoping that `tol` relates to the global error, because often in practice $|l_{i+1}| \simeq h|e_{i+1}|$. And then we are cheating by using the more accurate method for the next $y_{i+1}$, even though the error estimate relates to the less accurate method. But, as the politician said, it's all for the greater good.

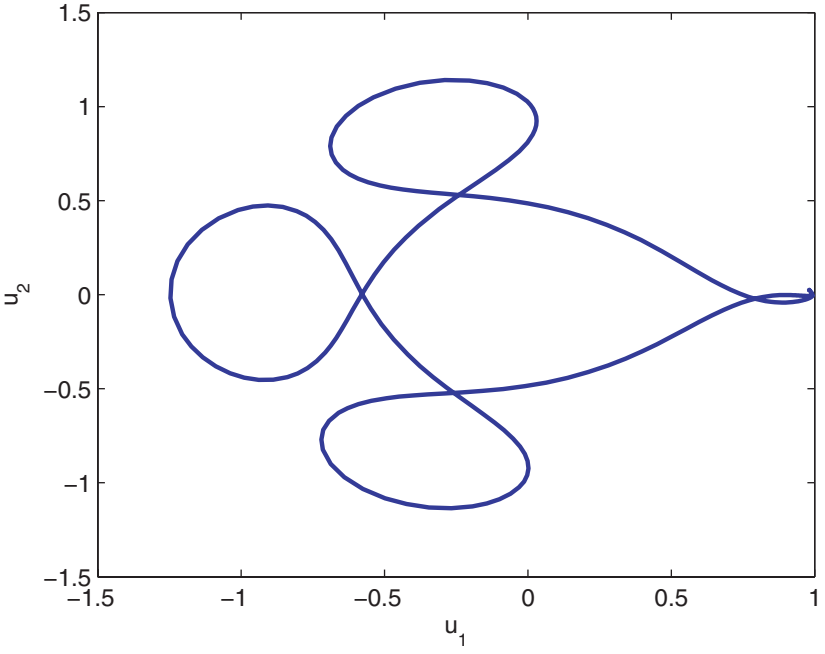



**Figure 16.11.** *Astronomical orbit using* `ode45`*; see Example* 16.20. *(Reprinted from Ascher and Petzold* [5]*.)*

Starting with the initial conditions

$$u_1(0) = 0.994,\ u_2(0) = 0,\ u_1'(0) = 0,$$
$$u_2'(0) = -2.00158510637908252240537862224,$$

the solution is periodic with period $< 17.1$. Note that $D_1 = 0$ at $(-\mu, 0)$ and $D_2 = 0$ at $(\hat{\mu}, 0)$, so we need to be careful when the orbit passes near these singularity points.

To apply our codes we write the problem as a first order system, $\mathbf{y}' = \mathbf{f}(\mathbf{y})$, where $\mathbf{y} = (u_1, u_1', u_2, u_2')^T$. The conversion to first order form is similar to that demonstrated in Example 16.2 and need not be repeated here.

The orbit is depicted in Figure 16.11. It was obtained using `ode45` with the default absolute error tolerance of 1.e-6 and relative error tolerance of 1.e-3 to integrate the problem on the interval $[0, 17.1]$. This necessitated 309 time steps with $\max h_i = .1892$ and $\min h_i = 1.5921\text{e-}7$. Additional runs with stricter tolerances suggest that the plot in Figure 16.11 is qualitatively correct.

We have also integrated the same problem employing our simple routine `rk4` (see page 495) with a uniform step size. Using 1,000 steps yields nonsensical results. Even 5,000 uniform steps yield qualitatively incorrect results. Only 10,000 uniform steps yield a qualitatively correct figure, similar as far as the naked eye is concerned to Figure 16.11. ∎

The success of the adaptive black box code in Example 16.20 is gratifying after all the effort that we have invested in studying numerical methods. But there are other examples, or problem instances, too. Often in practice, a uniform step size is not a bad choice. Perhaps more important, all those shortcuts described above in devising an efficient adaptive step size selection procedure may combine unexpectedly to surprise the unaware. Here is an example.

**Example 16.21 (FPU).** Consider a chain of $\hat{m}$ mass points connected with springs that have alternating characteristics: the odd ones are soft and nonlinear, whereas the even ones are stiff and linear. (This is often referred to as the *Fermi–Pasta–Ulam* (FPU) problem.) We next describe the details of the corresponding ODE system, which may look a bit complex to the point of being prohibitive. But fear not: all we really want to do here eventually is concentrate on the performance of `ode45`.

There are variables $q_1,\ldots,q_{2\hat{m}}$ and $p_1,\ldots,p_{2\hat{m}}$ in which the ODE system of size $m = 4\hat{m}$ is written as

$$q_j' = \frac{\partial H}{\partial p_j}, \quad p_j' = -\frac{\partial H}{\partial q_j}, \quad j = 1,\ldots,2\hat{m},$$

where $H$ is a scalar function of the $p_j$ and $q_j$ variables. This function is called the associated Hamiltonian and is given by

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{4}\left[ 2\sum_{j=1}^{2\hat{m}} p_j^2 + 2\omega^2 \sum_{j=1}^{\hat{m}} q_{\hat{m}+j}^2 + (q_1 - q_{\hat{m}+1})^4 + (q_{\hat{m}} + q_{2\hat{m}})^4 \right.$$
$$\left. + \sum_{j=1}^{\hat{m}-1} (q_{j+1} - q_{\hat{m}+1+j} - q_i - q_{\hat{m}+j})^4 \right].$$

The parameter $\omega = 100$ relates to the stiff spring constant.

For our purpose here, the details of $H$ are unimportant: what matters is that we are able to construct the corresponding ODE system (16.1a) for the $m$ unknowns

$$\mathbf{y} = \begin{pmatrix} \mathbf{q} \\ \mathbf{p} \end{pmatrix}$$

from the above prescription. Further, let us define

$$I_i = \frac{1}{2}(p_{\hat{m}+i}^2 + \omega^2 q_{\hat{m}+i}^2), \quad I = \sum_{i=1}^{\hat{m}} I_i.$$

Then it turns out that

$$I(\mathbf{q}(t),\mathbf{p}(t)) = I(\mathbf{q}(0),\mathbf{p}(0)) + \mathcal{O}(\omega^{-1})$$

for very long times $t$. (Such a property makes $I$ an *adiabatic invariant*, but this again is not our concern here.) Thus, we can run an ODE code over a relatively long time interval, plot the total energy $I$, and observe whether or not its trend to remain almost constant in time is honored in the computation. This provides a qualitative measure for the numerical simulation.

So, let us set $\hat{m} = 3$, yielding an ODE system of size $m = 12$, and select the initial conditions $\mathbf{q}(0) = (1,0,0,\omega^{-1},0,0)^T$, $\mathbf{p}(0) = (1,0,0,1,0,0)^T$. Integrating this ODE system from $a = 0$ to $b = 500$ using `rk4` with a constant step size $k = .00025$, i.e., using $2,000,000$ time steps, yields the qualitatively correct Figure 16.12. The curves depicted in the figure are exact as far as the eye can tell (trust us). The "noise" is not a numerical artifact. Rather, small "bubbles" of rapid oscillations occasionally flare up and quickly die away.

Next, we integrate this ODE system using `ode45` with default tolerances, as in Example 16.20. This requires $112,085$ time steps. The result, depicted in Figure 16.13, is a disaster, especially because it does not even look conspicuously wrong: it just *is* wrong.

Running `ode45` with tighter tolerances, specifically setting the relative error tolerance to 1.e-6, which is the same value as the absolute tolerance, does produce good results in $402,045$ steps. But the lesson remains: do not trust a complex mathematical software package blindly! ∎

*Specific exercises for this section*: Exercises 19–20.

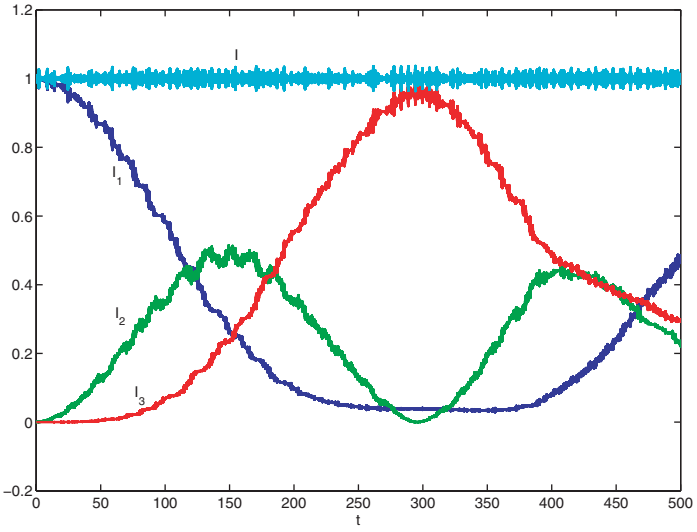



**Figure 16.12.** *Oscillatory energies for the FPU problem; see Example* 16.21.

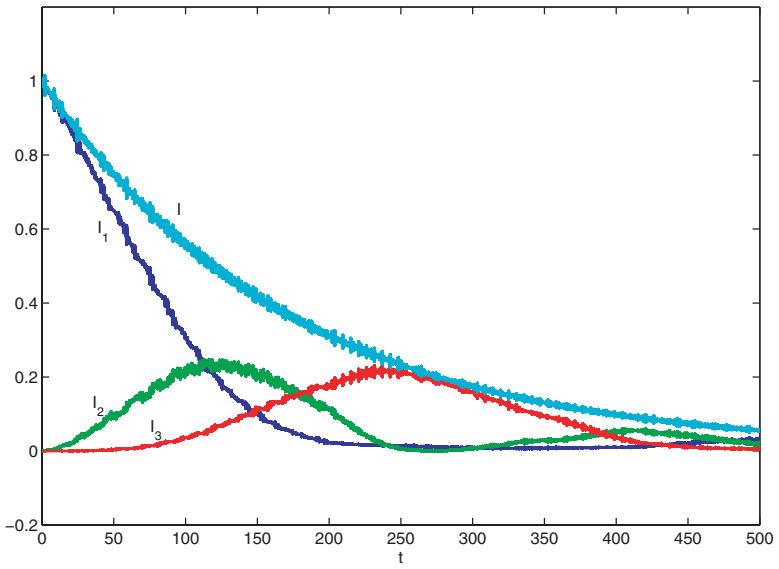



**Figure 16.13.** *Oscillatory energies for the FPU problem obtained using MATLAB's* `ode45` *with default tolerances. The deviation from Figure* 16.12 *depicts a significant, nonrandom error.*

## 16.7 *Boundary value ODEs

Generally, an ODE system with $m$ components, given as in (16.1a) by

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \quad a < t < b,$$

yields a family of solutions that can be specified in principle using $m$ parameters, as in Example 16.1. To obtain a locally unique solution, meaning that an arbitrarily small perturbation does not yield an-

other solution for the same ODE problem, requires the specification of $m$ side conditions, or specific solution values. In the easiest, and fortunately most important, case of an initial value problem all $m$ solution values are specified at the same initial point $t = a$. Thus, at $t = a$ all information about $\mathbf{y}(t)$ is available, and we can subsequently march along in $t > a$, both theoretically and numerically, with the entire solution information at hand. For instance, a step of the forward Euler method yields, starting from knowing an approximation for all components of $\mathbf{y}(t_i)$, an approximation for all components of $\mathbf{y}(t_i + h)$. Theoretically, this allows for conclusions about uniqueness and existence of a solution for the initial value ODE problem under general, mild assumptions. Practically, this allows for devising *explicit* numerical discretizations, an evolutionary solution process, and a local error control and step size determination.

The celebration ends when considering a *boundary value problem* (BVP),[63] where in its simplest incarnation there are $l$ components of $\mathbf{y}$ specified at $t = a$ and $m - l$ components of $\mathbf{y}$ specified at $t = b$. The case $l = m$ brings us back to an initial value problem, but here we concentrate on the case $0 < l < m$. Thus, necessarily $m > 1$, accounting for the boldface notation. Numerically, envisioning a discretization mesh

$$a = t_0 < t_1 < \cdots < t_N = b, \;\; h_i = t_i - t_{i-1},$$

on which the solution is sought, we expect all solution values $\{\mathbf{y}_i\}_{i=0}^N$ to become known *simultaneously* rather than gradually.

**Example 16.22.** Returning to Example 9.3, we can write the ODE

$$v'' + e^v = 0, \quad 0 < t < 1,$$

in first order form as in Example 16.2 and (16.1a). This yields

$$\mathbf{y} \equiv \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} v \\ v' \end{pmatrix}, \;\; \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ -e^{y_1} \end{pmatrix}.$$

The boundary conditions are

$$y_1(0) = 0, \;\; y_1(1) = 0.$$

If we were given not the value of $y_1$ at $t = 1$ but instead the value of $c_2$ such that $y_2(0) = c_2$, then the resulting initial value problem would have had a unique solution over the interval $[0, 1]$ and beyond for any finite value $c_2$. In contrast, for the BVP there are two distinct solutions; see Figure 9.3.

The simple discretization described in Examples 9.3 (page 257) and 4.17 (page 87) attempts to find all solution values of $y_1$ on a given mesh, in contrast to the marching strategy used for initial value ODE problems. This means that the size of the algebraic system to be solved for the BVP grows like $N$, which in turn is governed by accuracy considerations. In general, this algebraic system size is about $mN$. In contrast, upon applying an implicit method for the initial value problem there are about $N$ algebraic systems to be solved, each of a size that grows like $m$. Think, say, of 500 mesh points for a system of 5 ODEs to picture the difference. ∎

## Finite difference methods

The case of a scalar ODE of second order as in Example 16.22 is special. Having only $v$ and not $v'$ appear explicitly is even more special. The reason that all our previous BVP examples look like this

[63] Unfortunately, just about any PDE in practice is subject to boundary conditions, and not just initial conditions, as, for instance, in Examples 16.17 and 7.1. Correspondingly, there are no general-purpose codes, of the quality and generality available for initial value ODEs for any other class of differential problems.

is that a simple, straightforward discretization suggests itself, and this allowed us to concentrate on other solution aspects in previous chapters. Let us now turn to the more general BVP case where the ODE system is in the general first order form, and consider designing finite difference methods.

In principle, there are no explicit *finite difference* discretizations for BVPs. Since all methods are now implicit, and for reasons of maintaining sparsity in the resulting large linear algebraic systems to be solved, **implicit Runge–Kutta** discretizations are typically employed. For instance, consider the implicit midpoint method, for which we have

$$\frac{\mathbf{y}_i - \mathbf{y}_{i-1}}{h_i} = \mathbf{f}\left(t_{i-1/2}, \frac{\mathbf{y}_i + \mathbf{y}_{i-1}}{2}\right), \quad i = 1, \ldots, N.$$

Let us write the boundary conditions as

$$B_a \mathbf{y}_0 = \mathbf{c}_a, \quad B_b \mathbf{y}_N = \mathbf{c}_b,$$

where $B_a$ is an $l \times m$ given matrix and $B_b$ is likewise $(m-l) \times m$. The given boundary values $\mathbf{c}_a$ and $\mathbf{c}_b$ are of size $l$ and $m-l$, respectively. These difference equations plus the boundary conditions then yield a system of $m(N+1)$ algebraic equations in $m(N+1)$ unknowns.

Newton's method comes to mind in the case that $\mathbf{f}$ is nonlinear. However, the resulting algebraic problem may be much harder to solve than the nonlinear algebraic systems encountered earlier in this chapter around page 513 for stiff equations, because here there is no good initial guess in general (such a guess would have to be for the entire solution profile, as in Example 9.3), and because the system is rather large. Moreover, cutting down the step size makes the resulting algebraic system both larger and not necessarily easier to solve, in contrast to certain situations in the stiff initial value ODE context. Methods to handle the nonlinear BVP situation in general fall beyond the scope of the present quick presentation.

Let us concentrate instead on the case where $\mathbf{f}$ is linear, writing it as

$$\mathbf{f}(t, \mathbf{y}) = A(t)\mathbf{y} + \mathbf{q}(t),$$

with $A$ a known $m \times m$ matrix and $\mathbf{q}$ a known vector function of $t$, both potentially obtained from applying one step of a modified Newton method. Then the difference equations can be rearranged to read

$$\begin{aligned} &R_i \mathbf{y}_i + S_i \mathbf{y}_{i-1} = \mathbf{b}_i, \quad i = 1, \ldots, N, \\ &R_i = I - \frac{h_i}{2} A(t_{i-1/2}), \;\; S_i = I + \frac{h_i}{2} A(t_{i-1/2}), \;\; \mathbf{b}_i = h_i \mathbf{q}(t_{i-1/2}). \end{aligned}$$

Putting all these linear equations together yields a sparse, block-diagonal linear system of equations given by

$$\begin{pmatrix} B_a & & & & & & \\ S_1 & R_1 & & & & & \\ & S_2 & R_2 & & & & \\ & & \ddots & \ddots & & & \\ & & & & S_N & R_N & \\ & & & & & B_b & \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{c}_a \\ \mathbf{b}_1 \\ \vdots \\ \vdots \\ \mathbf{b}_N \\ \mathbf{c}_2 \end{pmatrix}.$$

This matrix can be viewed as banded, and it is amenable to direct linear algebra solution techniques, for which we refer the reader to Section 5.6. The solution cost is only $\mathcal{O}(N)$, so this is usually *not* the Achilles heel of BVP methods.

For a linearized version of Example 16.22, Exercise 24 shows that the obtained linear algebraic system using the midpoint method is pentadiagonal, rather than tridiagonal, as in Examples 4.17 and 9.3. Such is the price paid for generality.

## Shooting methods

The above description should make it clear that treating boundary value ODE problems using finite difference techniques is possible, but it is significantly more cumbersome than solving corresponding initial value ODE problems in the form (16.1), i.e., for the same $\mathbf{f}(t,\mathbf{y})$ with $\mathbf{y}(a)$ known. This brings about the following intuitive idea for solving the BVP: find the missing initial values by repeatedly solving the corresponding initial value ODEs for different initial values, matching the boundary values given at the other end $b$. This yields a generally nonlinear algebraic system of equations of size only $m-l$, the number of missing initial values. The resulting method is called *shooting*. When it works it has an unbeatable charm of apparent simplicity and the direct reliance on past knowledge and available software.

**Example 16.23.** Let us solve the problem of Examples 16.22 and 9.3 using the shooting method.

For a given value $c$, denote by $\mathbf{y}(t;c)$ the solution of the initial value problem with $y_1(0)=0$ and $y_2(0)=c$. Then we are looking for that value of $c$ such that

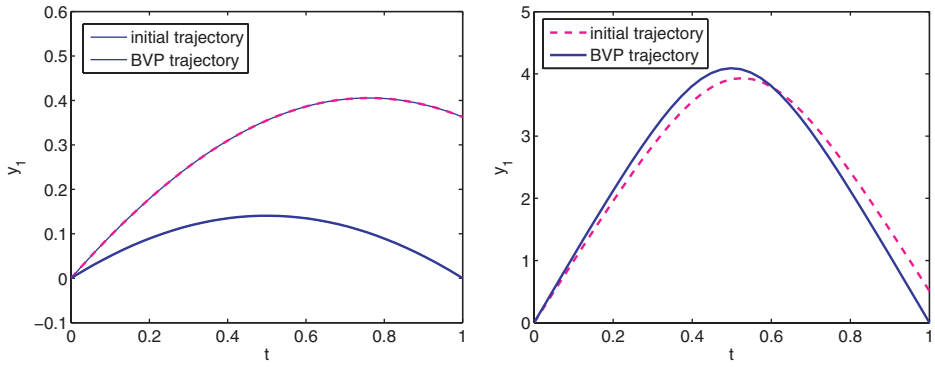
$$g(c) = y_1(1;c) = 0$$

to match the given boundary condition at $t=1$. The following MATLAB script uses `fzero` and `ode45` to carry out the shooting method in this very simple case:

```
c0 = input(' guess missing initial value for y_2 : ');

% solve IVP for plotting purposes
tspan = [0 1];
y0 = [0 ; c0];
[tot0,yot0] = ode45(@func, tspan, y0);

% solve BVP using fzero to solve g(c) = y_1(1;c) = 0.
c = fzero(@func14,c0);

% plot obtained BVP solution
y0 = [0 ; c];
[tot,yot] = ode45(@func, tspan, y0);
plot(tot0,yot0(:,1)','m--',tot,yot(:,1)','b')
xlabel('t')
ylabel('y_1')
legend('initial trajectory','BVP trajectory')

function f = func(t,y)
f(1) = y(2);
f(2) = -exp(y(1));
f = f(:);

function g = func14(c)
[tot,yot] = ode45(@func, [0 1], [0 c]);
g = yot(end,1);
```

The script invokes `fzero` to find a root for the function defined in `func14`, and this in turn invokes `ode45` to solve the initial value ODE defined in `func` with $y_2(0)=c$.

The results are plotted in Figure 16.14. The initial guess $c_0 = 1$ yields convergence to $c = .5494$ and Figure 16.14(a), whereas the guess $c_0 = 10$ yields convergence to $c = 10.8472$ and Figure 16.14(b). ∎




(a) $c_0 = 1$. (b) $c_0 = 10$.

**Figure 16.14.** *The two solutions of Examples* 16.22 *and* 9.3 *obtained using simple shooting starting from two different initial guesses $c_0$ for $v'(0)$. Plotted are the trajectories for the initial guesses (dashed magenta), as well as the final BVP solutions (solid blue). The latter are qualitatively the same as in Figure* 9.3. *Note the different vertical scales in the two subfigures.*

Example 16.23 is misleadingly simple. In the first place, the nonlinear function $g(c)$ is usually not scalar, so normally the relevant solution techniques are not the simple ones of Chapter 3 but rather the much more complex ones of Section 9.1. Moreover, the function `fzero` which has been successfully employed here does not use the derivative of $g$ and as such is suitable only for very simple problems. More generally the Jacobian matrix of a system of algebraic equations is required, and this leads to the solution of a larger auxiliary initial value ODE system, called the *variational equations*.

Still, the shooting method has attractive generality. However, it also has serious limitations. The first is that it relies on stability of the initial value ODE systems encountered, whereas all that we are legitimately allowed to assume is that the given boundary value ODE problem is stable. The other limitation is that the nonlinear algebraic problem can get exceedingly tough to solve as the interval of integration $[a,b]$ gets longer. Both of these limitations are mitigated by an approach called **multiple shooting**, which can be viewed as a method located somewhere between shooting and finite differences, where the interval $[a,b]$ is subdivided by a mesh as before, and the simple shooting method is applied on each subinterval $[t_{i-1},t_i]$. The number of shooting points $N$ no longer relates directly to accuracy requirements, and can often (though not always!) be taken significantly smaller than in a finite difference approach. The plot thickens here, however, so we must end our brief exposition lest we get carried away.

*Specific exercises for this section*: Exercises 24–25.

## 16.8 *Partial differential equations

The previous sections of this chapter are all concerned with differential equations that depend on just one independent variable. They provide a treatment of several approaches and methods, and also give some insight and useful practical knowledge for solving subproblems associated with PDEs.

PDEs and their associated numerical methods are in general much more complex than ODEs, and we proceed here to provide but a very quick description of what is today still a vast research area.

### Elliptic, parabolic, and hyperbolic PDEs

Some examples of PDEs and their numerical treatment have already made their way into previous chapters and sections of this text. The problem presented in Example 7.1 is the simplest example of an **elliptic** PDE. Endowing this PDE with *boundary conditions* (*not* initial conditions) yields a well-posed PDE problem, amenable to numerical treatment by methods that can be derived from basic principles. Such problems arise when modeling physical processes at steady state, i.e., when there is no dependence on time.

The simple finite difference discretization (7.1) (which may be derived by a direct application of the methods of Section 14.1, yielding second order accuracy) gives rise to a large, sparse system of linear algebraic equations. The latter fact is the reason why it stars in Chapter 7: see Examples 7.3, 7.5, 7.7, 7.8, 7.10, 7.11, and 7.17. More complicated elliptic PDEs are treated in Examples 7.13–7.15 and Exercises 7.18, 9.4, and 9.8.

In Example 16.17 we have discussed some numerical methods for the heat equation. This is a simple example of a **parabolic** PDE, which is typically endowed with both initial and boundary conditions. It is useful to consider a parabolic PDE as having both time and space variables, as in Example 16.17. If there is a steady state for the model problem, then we have a situation where $\frac{\partial u}{\partial t} = 0$, or $\frac{\partial u}{\partial t} \to 0$ as $t \to \infty$, and the parabolic problem (in more than one space variable) becomes an elliptic one.[64]

For a simple boundary value ODE such as that of Example 4.17 it is obvious that the solution $v(t)$ has two more derivatives than the right-hand-side function. Thus, in the notation of that example, if

$$g(t) = \begin{cases} 1, & 0 \le t < .5, \\ 2, & .5 \le t \le 1, \end{cases}$$

then $g$ is (bounded but) not differentiable, and yet $v$ and its derivative are differentiable. This **smoothing property** of the inverse of the differential equation operator is inherited by elliptic PDEs, especially when defined on domains without corners, and it is crucial to the success of preconditioners for iterative methods as described in Sections 7.4–7.6. Likewise, if the boundary conditions are not homogeneous, say, $u(x, y) = \hat{g}(x, y)$ on the domain boundary, and $\hat{g}$ has a jump discontinuity, say, then the solution inside the domain is smoother, even in locations very close to the boundary jump.

Parabolic problems also have a smoothing property. Often in practice the given initial and boundary value functions do not agree where they meet (i.e., at the boundary for $t = 0$; see Example 16.17), but this discontinuity is immediately smoothed inside the domain where the PDE is defined. This property is important for the design of numerical methods for such problems.

There is a third class of scalar, linear PDEs that do not possess a smoothing property. A **hyperbolic** PDE is typically well-posed with initial or initial-boundary conditions, but not with boundary conditions around its entire domain of definition.

**Example 16.24.** The first order PDE

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}$$

is an instance of the **advection equation**. Let us consider it on the half plane $t \ge 0$, $-\infty < x < \infty$, with initial conditions $u(t = 0, x) = u_0(x)$.

[64] However, a parabolic problem does not have to be in more than one space variable: for instance, the equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ is a parabolic PDE.

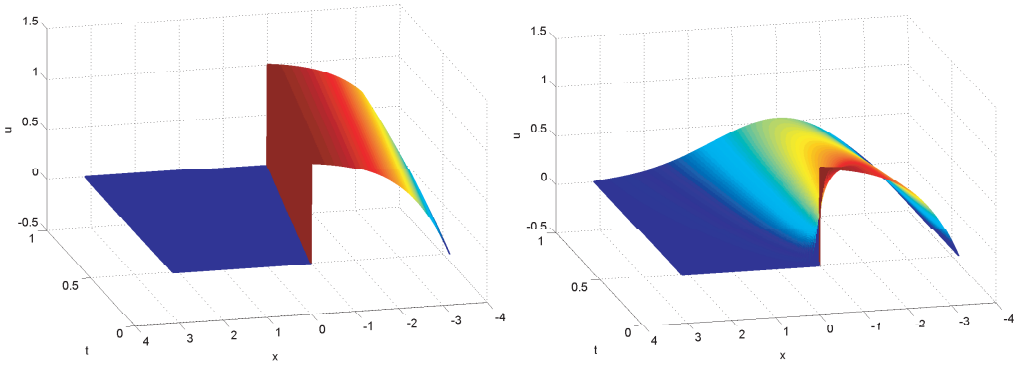
It is easy to see that the solution of the PDE can be written in general as $u(t,x) = q(t+x)$. Furthermore, setting $t = 0$ we get $q(x) = u_0(x)$, so the solution for this initial value problem is

$$u(t,x) = u_0(t+x).$$

Thus, the initial profile $u_0(x)$ is propagated in time leftward (e.g., setting $t = 1$ the value of $u$ at $x$ is the initial value at $x+1$) unchanged in shape, like a ripple on the ocean surface or a water wave.

In particular, if the initial value function contains a discontinuity, then this discontinuity propagates unsmoothed into the domain in $(t,x)$. Also, thinking of a finite domain we cannot arbitrarily set boundary values for $u$ at the top (i.e., for some finite $T > 0$) or the left wall (i.e., for some finite $x_L$ such that $x_L \le x$), because the values there are dictated by $u_0$. See Figure 16.15(a), which displays the solution for

$$u_0(x) = \begin{cases} (1-\exp(-x-\pi))/(1-\exp(-\pi)), & x < 0, \\ 0, & x \ge 0. \end{cases}$$




(a) Advection. (b) Heat.

**Figure 16.15.** *Hyperbolic and parabolic solution profiles starting from a discontinuous initial value function. For the advection equation the exact solution is displayed. For the heat equation we have integrated numerically, using homogeneous BC at $x = \pm\pi$ and a rather fine discretization mesh.*

The solution profile for the advection equation should be compared with the solution starting from the same initial value function $u_0(x)$ of the parabolic PDE

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}.$$

For the latter we set the boundary conditions $u = 0$ at $x = \pm\pi$; see Figure 16.15(b). Note the immediate smoothing of the initial value profile by the heat equation operator for $t > 0$. ∎

## The finite element method

If the domain on which an elliptic PDE is given is simply a rectangle (or a box in three dimensions), then often a discretization based on finite differences can be conveniently whipped up. The challenge then shifts to solving the resulting system of algebraic equations. But what if the domain

is not geometrically simple, as in Figure 11.13, for instance? Constructing reasonable difference approximations becomes complicated and cumbersome in such a situation.

The finite element method (FEM) is in fact an approach yielding a family of methods that provide an alternative to the finite difference family. It is particularly attractive for problems defined on domains of general shapes whose boundaries do not align with coordinate axes, as it remains general and elegant for such problems, too. Furthermore, the theory behind the FEM family is relatively solid and deep. In fact, in addition to the attention given to computational aspects, there is a large community of scientists who are interested in the mathematical theory behind FEM approximations.

For the prototype Poisson equation, reintroduced towards the end of Section 16.1, the domain $\Omega$ can be triangulated into *elements* as in Figure 11.13. An approximate solution is sought in the form of a piecewise polynomial function over the elements. On each triangle this solution reduces to a polynomial of a certain degree, and these polynomials are pieced together along interelement boundaries. Such an approximate solution belongs to a space of functions $V_h$, where $h$ is a typical element side length.

The functions in $V_h$ have bounded first derivatives, yet second derivatives may be generally unbounded, unlike those of the exact solution $u$. However, the space in which $u$ is sought can be correspondingly enlarged by considering the **variational form**. If $V$ is the space of all functions satisfying the homogeneous boundary conditions (see Example 7.1 and imagine a nonsquare domain) that are square integrable together with their first derivatives, then for each **test function** $w \in V$ we can integrate by parts to obtain

$$\iint_\Omega -\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) w \, dxdy = \iint_\Omega \left(\frac{\partial u}{\partial x}\frac{\partial w}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial w}{\partial y}\right) dxdy.$$

Writing the right-hand side as $b(u,w)$ and defining for the source term $g$ the corresponding expression

$$(g,w) = \iint_\Omega gw \, dxdy,$$

we obtain the variational formulation of the differential problem:

Find $u \in V$ such that for all $w \in V$

$$b(u,w) = (g,w).$$

Now, the **Galerkin** FEM discretizes the variational form above instead of the original PDE! Thus we seek $u_h \in V_h$ such that for any $w_h \in V_h$

$$b(u_h, w_h) = (g, w_h).$$

This FEM is called **conforming** since $V_h \subset V$.

How is the computation mandated by the Galerkin formulation carried out in practice? We describe the piecewise polynomial functions using a **basis**. Let the functions $\phi_i(x,y) \in V_h, i = 1,\ldots,J$, be such that any $w_h \in V_h$ can be uniquely written as

$$w_h(x,y) = \sum_{i=1}^{J} w_i \phi_i(x,y)$$

for a set of coefficients $w_i$. Then the coefficients $u_i$ that correspondingly define $u_h$ are determined by requiring

$$\sum_{i=1}^{J} b\left(\phi_i, \phi_j\right) u_i = b\left(\sum_{i=1}^{J} u_i \phi_i, \, \phi_j\right) = (g, \phi_j), \quad j = 1,\ldots,J.$$

This yields a set of $J$ linear equations for the $J$ coefficients $u_i$.

Assembling this linear system of equations, the resulting matrix $A$ is called the **stiffness matrix** for historical reasons. We have $a_{i,j} = b(\phi_i, \phi_j)$, $1 \le i, j \le J$. Of course, we seek a basis such that assembling $A$ and solving the resulting linear algebraic equations can be done as efficiently as possible. In one dimension we have seen such bases in Section 11.4. For a piecewise linear approximate solution on a triangulation as in Figure 11.13, this is achieved by associating with each vertex, or *node* $(x_i, y_i)$, a **roof function** (which also goes by the more tribal name *tent function*) $\phi_i \in V_h$ that vanishes at all nodes other than the $i$th one and satisfies $\phi_i(x_i, y_i) = 1$. This gives three interpolation conditions at the vertices of each triangular element, determining a linear polynomial in two dimensions, and the global solution pieced together from these triangle contributions is thus once differentiable. Then, unless vertex $j$ is an immediate neighbor of vertex $i$, we have $b(\phi_i, \phi_j) = 0$ because the regions on which these two functions do not vanish do not intersect. Moreover, the construction of nonzero entries of $A$ becomes relatively simple, although it still involves quadrature approximations for the integrals that arise.

Note that

$$u_i = u_h(x_i, y_i).$$

This is called a **nodal** method. The resulting scheme has many characteristics of a finite difference method, including sparsity of $A$, with the advantages that the mesh has a more flexible shape. Note also that $A$ is symmetric positive definite in this specific case.

The assembly of the FEM equations, especially in more complex situations than for our model Poisson problem, can be significantly more costly than simple finite differences. Moreover, for more complicated PDEs different (and often more involved) elements may be required. Nevertheless, for problems with complex geometries the FEM is the approach of choice, and most general packages for solving elliptic boundary value problems are based on it.

## Methods for hyperbolic problems

The numerical solution of hyperbolic PDEs involves considerable challenges, even in one space dimension where complex domain geometry does not arise. Thus, next we consider finite difference methods for such PDEs in time and one space variable. The essential source for numerical difficulties is the lack of smoothing or natural dissipation. Even if the solution is known to be smooth, there is no quick decay to steady state as in Example 16.17, so integration over a long time must be contemplated. Moreover, assuming an appropriate discretization in space, the eigenvalues of the resulting ODE system in time tend to be purely imaginary, which leads to only marginal stability in terms of the absolute stability concept discussed in Section 16.5. Furthermore, discontinuities in initial or boundary value functions propagate into the domain in $(t, x)$ where the PDE is defined, as we have seen in Example 16.24. In fact, it turns out that for nonlinear problems of this type solution discontinuities may form inside the spacetime domain even if the initial and boundary value data are smooth! Who said life can't be exciting?

Consider again the simple advection equation of Example 16.24. We have seen that the exact solution propagates from $x = 0$ at the initial time line with **wave speed** $\frac{dx}{dt} = -1$ along the **characteristic curve**

$$x + t = 0.$$

Another simple instance of a hyperbolic equation is the **classical wave equation**

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \quad t \ge 0,\ x_0 < x < x_{J+1},$$

where $c > 0$ is the speed of sound. This equation is subject to two initial conditions

$$u(0, x) = u_0(x), \quad \frac{\partial u}{\partial t}(0, x) = u_1(x),$$

and to boundary conditions, which we will take to be

$$u(t,x_0) = u(t,x_{J+1}) = 0 \quad \forall t.$$

Here in fact there are two characteristic curves given by

$$\frac{dx}{dt} = \pm c,$$

so there are waves propagating both leftward and rightward at the speed of sound. In fact, it can be verified that the exact solution before boundary conditions get in the way is

$$u(t,x) = \frac{1}{2}[u_0(x-ct) + u_0(x+ct)] + \frac{1}{2c}\int_{x-ct}^{x+ct} u_1(s)ds,$$

which can be used as a sanity check on proposed numerical methods.

The classical wave equation resembles the homogeneous Poisson equation in appearance (if we could only take $c = \imath$), even though its essential properties are rather different (because we can't: the speed of sound is real). Applying a similar centered discretization as in (7.1) we obtain here the **leapfrog** scheme

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = c^2 \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}, \quad j = 1,2,\ldots,J.$$

This is an explicit, two-step scheme in time: we use the initial conditions to determine $u_j^0 = u_0(x_j)$ and $u_j^1 \approx u(\Delta t, x_j)$ for all $j = 1,\ldots,J$, and then march forward in time by calculating $u_j^{n+1}$, for all $j$, for $n = 1,2,\ldots$, using the boundary conditions at each such step to close the system.

Is there a restriction on the time step size $\Delta t$ that we may take? A look at the exact solution formula indicates that at some point $(n\Delta t, j\Delta x)$, the solution depends on the initial data at $t = 0$ only in the dependence interval $[j\Delta x - cn\Delta t, j\Delta x + cn\Delta t]$. For the numerical solution by the leapfrog scheme the corresponding dependence interval is $[j\Delta x - n\Delta x, j\Delta x + n\Delta x]$. Now, if $\Delta x < c\Delta t$, then the latter interval is strictly contained in the former, so an arbitrary change of $u_0(x)$ in the "skin area" $j\Delta x + n\Delta x < x \le j\Delta x + cn\Delta t$ will affect the exact solution without having any effect on the numerical one! This is unacceptable, of course, and we therefore must obey the time step restriction

$$c\Delta t \le \Delta x.$$

This is the famous CFL condition derived by Courant, Friedrichs, and Lewy in 1928 without the use of computing power. As it turns out it agrees with the numerical stability restriction corresponding to Section 16.5 for this method.

**Example 16.25.** Let us use the leapfrog scheme to calculate solutions to the classical wave equation with $c = 1$, subject to initial conditions $u_0(x) = \exp(-\alpha x^2)$, $u_1(x) = 0$, on the interval $-10 \le x \le 10$. We set $\Delta x = .04$ and $\Delta t = .02$, a combination that satisfies the CFL condition.

A "waterfall" plot depicting the progress of the initial pulse in time appears in Figure 16.16. The original pulse splits into two pulses which travel at speeds $\pm 1$, change sign at the boundaries, and return, change sign again, and reunite to form the original shape at $t = 40$. This pattern repeats periodically for $t > 40$. For $\alpha = 1$ the numerical reconstruction is very good.

However, for $\alpha = 20$ the pulse profile is too sharp for this mesh resolution. Additional ripples form in the numerical solution as a result, which is unfortunate since the error looks "physical," so it is possible in more complex situations that a researcher may not be able to tell from the numerical solution shape that it is wrong. ∎

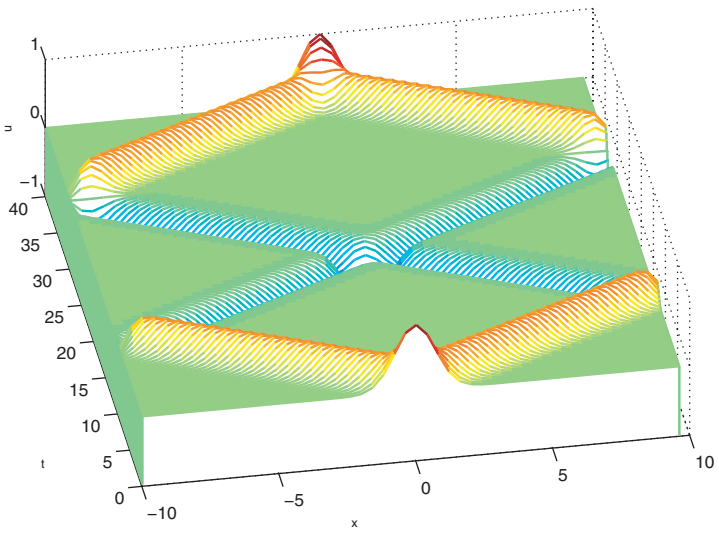



**Figure 16.16.** *"Waterfall" solutions of the classical wave equation using the leapfrog scheme; see Example* 16.25. *For* $\alpha = 1$ *the solution profile is resolved well by the discretization.*

The leapfrog scheme is simple and second order accurate in both time and space, provided that the approximate solution is smooth. This follows from the derivation of difference approximations in Section 14.1 combined with a classical convergence-stability theorem that is beyond the scope of our quick PDE-world tour. But if the solution is not smooth, then no such result holds for a simple centered scheme. Indeed, the derivations in Chapter 14 all use one form or another of local polynomial approximation. Thus, numerical differentiation across a mesh element in spacetime where the solution is discontinuous could prove to be a very bad idea.

Consider discretizing the advection equation in Example 16.24. A numerical solution is sought at the point $(t_{n+1}, x_j)$ based on knowledge of current numerical solution values at $(t_n, x_i)$, for all $i$, where $t_n = n\Delta t$ and $x_i = i\Delta x$. But the exact solution follows the characteristic curve, and hence $u(t_{n+1}, x_j) = u(t_n, x_j + \Delta t)$. The CFL condition here indicates that we must select $\Delta t \le \Delta x$, so $x_j \le x_j + \Delta t \le x_{j+1}$. We can therefore interpolate the two mesh values $u_j^n$ and $u_{j+1}^n$ for the value at the foot of the characteristic that leads to the point of interest at the next time level. This gives the explicit one-sided method

$$u_j^{j+1} = u_j^n + \frac{\Delta t}{\Delta x}(u_{j+1}^n - u_j^n).$$

More generally, for the advection equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0,$$

we construct the **upwind scheme** by

$$u_j^{j+1} = u_j^n - \frac{\Delta t}{\Delta x}\begin{cases} a(u_{j+1}^n - u_j^n), & a < 0, \\ a(u_j^n - u_{j-1}^n), & a > 0. \end{cases}$$

This scheme goes with the flow, and not just in a manner of speech.

The correct version of the upwind scheme for the nonlinear **Burgers equation**

$$\frac{\partial u}{\partial t} + .5\frac{\partial u^2}{\partial x} = 0$$

reads

$$u_j^{j+1} = u_j^n - \frac{\Delta t}{2\Delta x} \begin{cases} (u_{j+1}^n)^2 - (u_j^n)^2, & u_j^n < 0, \\ (u_j^n)^2 - (u_{j-1}^n)^2, & u_j^n \geq 0. \end{cases}$$

(Note that the PDE can be written as $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$, so $u$ plays the role of $a$ in the advection equation.) This yields reasonable first order approximations for a problem whose solution may develop discontinuities even when starting from smooth initial data. But justifying all this is well beyond the scope of our text, and here is a good spot to stop.

## 16.9 Exercises

0. **Review questions**

   (a) For a given ODE system, what are an initial value problem, a terminal value problem, and a boundary value problem?

   (b) What are the basic differences between the numerical solution of ODEs and numerical integration?

   (c) Distinguish between the forward Euler and backward Euler methods.

   (d) Define local truncation error and order of accuracy and show that both the forward Euler and the backward Euler methods are first order accurate.

   (e) How does the global error relate to the local truncation error?

   (f) What is an explicit RK method? Write down its general form.

   (g) Define convergence rate, or observed order.

   (h) Name three advantages that RK methods have over multistep methods and three advantages that multistep methods have over RK methods.

   (i) Why is it difficult to apply error control and step size selection using the global error?

   (j) In what sense are linear multistep methods linear?

   (k) What are the two Adams families of methods? What is the main distinguishing point between them?

   (l) Write down the PECE method for the two-step formula pair consisting of the two-step Adams–Bashforth method and the one-step, second order Adams–Moulton method.

   (m) Define region of absolute stability and explain its importance.

   (n) What is a stiff ODE problem? Why is this concept important in the numerical solution of ODEs?

   (o) Define A-stability and L-stability and explain the difference between these concepts.

1. The ODE that leads to Figure 16.2 is $y' = 10y$. The exact solution satisfies $y(0) = 1$, and Euler's method is applied with step size $h = 0.1$.

   What are the initial values $y(0)$ for the other two trajectories, depicted in Figure 16.2 in dashed lines?

2. (a) Use the Taylor expansion

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{6}y'''(t_i) + \frac{h^4}{24}y^{(iv)}(t_i) + \frac{h^5}{120}y^{(v)}(t_i) + \mathcal{O}(h^6)$$

to derive a corresponding series expansion for the local truncation error of the forward Euler method.

(b) Manipulating the forward Euler method written for the step sizes $h$ and $h/2$, apply extrapolation (Section 14.2) to obtain a second order one-step method.

(c) Manipulating the forward Euler method written for the step sizes $h$, $h/2$, and $h/3$, apply extrapolation to obtain a third order one-step method.

3. Show that the backward Euler method obeys the same convergence theorem as the one given on page 489 for the forward Euler method.

4. Derive convergence results (i.e., a bound on the global error in terms of a bound on an appropriate derivative of the exact solution $y(t)$ and $h^q$) for the extrapolation methods of orders $q = 2$ and $q = 3$ derived in Exercise 2.

5. Consider the ODE

$$\frac{dy}{dt} = f(t, y), \qquad 0 \le t \le b,$$

where $b \gg 1$.

(a) Apply the *stretching* transformation $t = \tau b$ to obtain the equivalent ODE

$$\frac{dy}{d\tau} = b\ f(\tau b, y), \qquad 0 \le \tau \le 1.$$

(Strictly speaking, $y$ in these two ODEs is not quite the same function. Rather, it stands in each case for the unknown function.)

(b) Show that applying the forward Euler method[65] to the ODE in $t$ with step size $h = \Delta t$ is equivalent to applying the same method to the ODE in $\tau$ with step size $\Delta\tau$ satisfying $\Delta t = b\Delta\tau$. In other words, the same stretching transformation can be equivalently applied to the discretized problem.

6. (a) Show that the explicit trapezoidal method for $y' = f(y)$ is second order accurate.

(b) Show that the explicit midpoint method for $y' = f(y)$ is second order accurate.

[Hint: You need to compare derivatives of $y$ with their expressions in terms of $f$ and show that terms that are not $\mathcal{O}(h^2)$ cancel in the expression for $d_i$. Use the identities $y' = f$ and $y'' = \frac{\partial f}{\partial y} y' = f_y f$.]

7. Show that the methods obtained by extrapolation in Exercise 2 are special cases of explicit RK methods, and argue that in general there is nothing particular to recommend them, except for the ease of ascertaining their order of accuracy.

8. To draw a circle of radius $r$ on a graphics screen, one may proceed to evaluate pairs of values $x = r\cos(\theta)$, $y = r\sin(\theta)$ for a succession of values $\theta$. But this is computationally expensive. A cheaper method may be obtained by considering the ODE

$$\begin{aligned} \dot{x} &= -y, \qquad x(0) = r, \\ \dot{y} &= x, \qquad y(0) = 0, \end{aligned}$$

where $\dot{x} = \frac{dx}{d\theta}$, and approximating this using a simple discretization method. However, care must be taken to ensure that the obtained approximate solution looks right, i.e., that the approximate curve closes rather than spirals.

[65] The same holds for any of the discretization methods in this chapter.

Carry out this integration using a uniform step size $h = .02$ for $0 \le \theta \le 120$, applying forward Euler, backward Euler, and the implicit trapezoidal method. Determine if the solution spirals in, spirals out, or forms an approximate circle as desired. Explain the observed results.

[Hint: This has to do with a certain invariant function of $x$ and $y$, rather than with the accuracy order of the methods.]

9. The observed order (rate) defined before Example 16.7 is based on the assumption that the calculation is carried out once with $h_1 = h$ and once with $h_2 = 2h$. Show that, more generally,

$$\text{Rate}(h) = \log_2\left(\frac{e(h_2)}{e(h_1)}\right) / \log_2\left(\frac{h_2}{h_1}\right).$$

10. Write the explicit and implicit trapezoidal methods, as well as the classical RK method of order 4, in tableau notation.

11. The three-stage RK method given by the tableau

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

is based, like the classical RK method, on the Simpson quadrature rule.

Implement this method and add a corresponding column to Table 16.2. What is the observed order of the method?

12. Derive the two-step Adams–Moulton formula

$$y_{i+1} = y_i + \frac{h}{12}(5f_{i+1} + 8f_i - f_{i-1}).$$

13. Propose a way to initialize (i.e., specifying $y_1, \ldots, y_{s-1}$) the Adams–Bashforth methods of orders 1, 2, 4, and 5.

14. Show that the local truncation error of the four-step Adams–Bashforth method is $d_i = \frac{251}{720}h^4 y^{(v)}(t_i)$, that of the five-step Adams–Bashforth method is $d_i = \frac{95}{288}h^5 y^{(vi)}(t_i)$, and that of the four-step Adams–Moulton method is $d_i = \frac{-3}{160}h^5 y^{(vi)}(t_i)$.

15. The **leapfrog** method for the ODE $y' = f(t, y)$ is derived like the forward and backward Euler methods, except that centered differencing is used. This yields the formula

$$\frac{y_{i+1} - y_{i-1}}{2h} = f(t_i, y_i).$$

Show that this is an explicit linear two-step method that is second order accurate and does not belong to the Adams or BDF families.

16. Verify that the implicit midpoint and trapezoidal methods are both A-stable.

[Some of that suppressed knowledge you have about complex arithmetic may come in handy here!]

17. (a) Show that the application of an explicit $s$-stage RK method to the test equation can be written as $y_{i+1} = R(z)y_i$, where $z = \lambda h$ and $R(z)$ is a polynomial of degree at most $s$ in $z$.

    (b) Further, if the order of the method is $q = s$, then any such method for a fixed $s$ has the same amplification factor $R(z)$, given by

$$R(z) = \sum_{j=0}^{s} \frac{z^j}{j!}.$$

    [Hint: Consider the Taylor expansion of the exact solution of the test equation.]

18. (a) Show that no explicit RK method can be A-stable or L-stable.

    (b) Show that the implicit midpoint and trapezoidal methods are not L-stable.

19. Suppose we have at mesh points $a = t_0 < t_1 < \cdots < t_{N-1} < t_N = b$ the result of a calculation of a solution for a given smooth initial value ODE using a one-step method of order $q$.

    (a) By repeating the calculation using the same method at the mesh points $t_0, \frac{t_0+t_1}{2}, t_1, \frac{t_1+t_2}{2}, t_2, \ldots t_{N-1}, \frac{t_{N-1}+t_N}{2}, t_N$, i.e., with each of the original step sizes $h_i = t_{i+1} - t_i$ halved, show how an estimate for the *global* error may be obtained.

    (b) Estimate global errors for the results of Example 16.7 without using the known exact solution. Compare your estimates with the exact errors given in Table 16.2.

20. The ODE $u'' = \frac{1}{t}u' - 4t^2u$ has the solution $u(t) = \sin(t^2) + \cos(t^2)$.

    (a) Plot the exact solution over the interval $[1, 20]$.

    (b) Integrate this problem on the interval $[1, 20]$ using `ode45` with default tolerances (specify initial values $u(1)$ and $u'(1)$ based on the given exact solution), and record the absolute error in $u$ at $t = 20$ and the number of steps required to get there.

    Then use our `rk4` with a uniform step size $h = .01$ for the same purpose. Compare errors and step counts, and comment.

    (c) Now use MATLAB's `odeset` to change the relative error tolerance in `ode45` to 1.e-6. Repeat the run as above. Repeat also the `rk4` run, this time with a uniform step size $h = .002$. Compare errors and step counts, and comment further. Do not become cynical.

21. The ODE system given by

$$y_1' = \alpha - y_1 - \frac{4y_1y_2}{1+y_1^2},$$
$$y_2' = \beta y_1 \left(1 - \frac{y_2}{1+y_1^2}\right),$$

    where $\alpha$ and $\beta$ are parameters, represents a simplified approximation to a chemical reaction. There is a parameter value $\beta_c = \frac{3\alpha}{5} - \frac{25}{\alpha}$ such that for $\beta > \beta_c$ solution trajectories decay in amplitude and spiral in phase space into a stable fixed point, whereas for $\beta < \beta_c$ trajectories oscillate without damping and are attracted to a stable limit cycle. (This is called a *Hopf bifurcation*.)

(a) Set $\alpha = 10$ and use any of the discretization methods introduced in this chapter with a fixed step size $h = 0.01$ to approximate the solution starting at $y_1(0) = 0$, $y_2(0) = 2$, for $0 \le t \le 20$. Do this for the parameter values $\beta = 2$ and $\beta = 4$. For each case plot $y_1$ vs. $t$ and $y_2$ vs. $y_1$. Describe your observations.

(b) Investigate the situation closer to the critical value $\beta_c = 3.5$. (You may have to increase the length of the integration interval $b$ to get a better look.)

22. In molecular dynamics simulations using classical mechanics modeling, one is often faced with a large nonlinear ODE system of the form

$$M\mathbf{q}'' = \mathbf{f}(\mathbf{q}), \quad \text{where } \mathbf{f}(\mathbf{q}) = -\nabla U(\mathbf{q}).$$

Here $\mathbf{q}$ are generalized positions of atoms, $M$ is a constant, diagonal, positive mass matrix, and $U(\mathbf{q})$ is a scalar potential function. Also, $\nabla U(\mathbf{q}) = (\frac{\partial U}{\partial q_1}, \ldots, \frac{\partial U}{\partial q_m})^T$. A small (and somewhat nasty) instance of this is given by the Morse potential where $\mathbf{q} = q(t)$ is scalar, $U(q) = D(1 - e^{-S(q-q_0)})^2$, and we use the constants $D = 90.5 \cdot 0.4814\text{e-}3$, $S = 1.814$, $q_0 = 1.41$, and $M = 0.9953$.

(a) Defining the velocities $\mathbf{v} = \mathbf{q}'$ and momenta $\mathbf{p} = M\mathbf{v}$, the corresponding first-order ODE system for $\mathbf{q}$ and $\mathbf{v}$ is given by

$$\begin{aligned} \mathbf{q}' &= \mathbf{v}, \\ M\mathbf{v}' &= \mathbf{f}(\mathbf{q}). \end{aligned}$$

Show that the Hamiltonian function

$$H(\mathbf{q}, \mathbf{p}) = \mathbf{p}^T M^{-1} \mathbf{p}/2 + U(\mathbf{q})$$

is constant for all $t > 0$.

(b) Use a library nonstiff RK code based on a 4(5) embedded pair such as MATLAB's `ode45` to integrate this problem for the Morse potential on the interval $0 \le t \le 2000$, starting from $q(0) = 1.4155$, $p(0) = \frac{1.545}{48.888} M$. Using a tolerance `tol` = 1.e-4, the code should require a little more than 1000 times steps. Plot the obtained values for $H(q(t), p(t)) - H(q(0), p(0))$. Describe your observations.

23. The first order ODE system introduced in the previous exercise for $\mathbf{q}$ and $\mathbf{v}$ is in *partitioned form*. It is also a Hamiltonian system with a separable Hamiltonian; i.e., the ODE for $\mathbf{q}$ depends only on $\mathbf{v}$ and the ODE for $\mathbf{v}$ depends only on $\mathbf{q}$. This can be used to design special discretizations. Consider a constant step size $h$.

(a) The *symplectic Euler* method applies backward Euler to the ODE $\mathbf{q}' = \mathbf{v}$ and forward Euler to the other ODE. Show that the resulting method is explicit and first order accurate.

(b) The *leapfrog*, or *Verlet,* method can be viewed as a staggered midpoint discretization and is given by

$$\begin{aligned} \mathbf{q}_{i+1/2} - \mathbf{q}_{i-1/2} &= h\, \mathbf{v}_i, \\ M(\mathbf{q}_{i+1/2})(\mathbf{v}_{i+1} - \mathbf{v}_i) &= h\, \mathbf{f}(\mathbf{q}_{i+1/2}). \end{aligned}$$

Thus, the mesh on which the $\mathbf{q}$-approximations "live" is staggered by half a step compared to the $\mathbf{v}$-mesh. The method can be kick-started by

$$\mathbf{q}_{1/2} = \mathbf{q}_0 + h/2\mathbf{v}_0.$$

To evaluate $\mathbf{q}_i$ at any mesh point, the expression

$$\mathbf{q}_i = \frac{1}{2}(\mathbf{q}_{i-1/2} + \mathbf{q}_{i+1/2})$$

can be used.

Show that this method is explicit and second order accurate.

24. Consider a linearized version of Example 16.22, given by

$$v'' + a(t)v = q(t), \quad v(0) = v(1) = 0.$$

(a) Converting the linear ODE to first order form as in Section 16.7, show that

$$A(t) = \begin{pmatrix} 0 & 1 \\ -a(t) & 0 \end{pmatrix}, \ \mathbf{q}(t) = \begin{pmatrix} 0 \\ q(t) \end{pmatrix},$$
$$B_a = B_b = \begin{pmatrix} 1 & 0 \end{pmatrix}, \ c_a = c_b = 0.$$

(b) Write down explicitly the linear system of algebraic equations resulting from the application of the midpoint method. Show that the obtained matrix is banded with five diagonals.

25. Consider the problem

$$v'' + \tau e^v = 0, \quad v(0) = v(1) = 0,$$

where $\tau > 0$ is a parameter. For $\tau = 1$ we saw in Examples 16.22–16.23 that there are two distinct solutions. These can be distinguished by their function norm defined by

$$\|v\|^2 = \int_0^1 v^2(s)ds.$$

It turns out that as $\tau$ is increased the two solutions approach one another: for a critical value $\tau^*$ they coincide, and for $\tau > \tau^*$ there is no solution for this problem anymore.

Solving this problem numerically for values of $\tau$ near $\tau^*$ can be challenging. Instead, we embed it first in the larger problem

$$\begin{aligned} v'' + \tau e^v &= 0, \quad v(0) = v(1) = 0, \\ \tau' &= 0, \\ w' &= v^2, \quad w(0) = 0, \ w(1) = \mu. \end{aligned}$$

This nonlinear system has a unique solution for each $\mu = \|v\|^2$.

Solve this enlarged boundary value ODE by a method of your choice for a sequence of values of $\mu \in [.1, 10]$, using the obtained solution for the previous $\mu$ as an initial guess for the next boundary value problem in the sequence. This is called a **simple continuation**. Plot $\mu$ vs. $\tau$. You should clearly see the turning point where the number of solutions for a given $\tau$ changes.

## 16.10 Additional notes

There are many books devoted to the numerical solution of initial value ODEs. The material in Sections 16.2–16.7 is covered more thoroughly in Ascher and Petzold [5]. Deeper, more encyclopedic books are Hairer, Norsett, and Wanner [36] and Hairer and Wanner [37].

Many iterative methods in optimization and linear algebra, including most of those described in Chapters 3, 7, and 9, can be written as

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}(\mathbf{y}_i), \quad i = 0, 1, \ldots,$$

where $h_i$ is a scalar step size. This reminds one of Euler's method for the ODE system

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}).$$

The independent variable $t$ is an "artificial time" variable. Much has been made of such connections recently, and this simple observation does prove important in some instances. But caution should be exercised here: always ask yourself if the "discovery" of the artificial ODE actually adds something in your quest for better algorithms for your given problem.

Much research on methods for stiff problems was carried out in the 1970s and 1980s. Despite the simplicity of the test equation there is significant general complication in stiff problems, essentially because some fast scales that are present in the given ODE are not approximated well in cases where these scales don't show up as a fast variation in the sought solution. This is fundamentally different from the usual nonstiff scenario, where the discretization typically approximates all scales well. Pioneering work was done by Gear [28]. The most exhaustive reference known to us remains [37].

The problem described in Example 16.4 is one in a set of initial value ODE applications used for testing research codes and maintained by F. Mazzia and F. Iavernaro in http://pitagora.dm.uniba.it/∼testset/.

A lot of attention has been devoted to numerical methods for *dynamical systems*; see Stuart and Humphries [66] and Strogatz [65].

Significant recent research work has been carried out in the context of *geometric integration*, and we refer the reader to Hairer, Lubich, and Wanner [35] and Leimkuhler and Reich [49] for comprehensive accounts on this topic. There is a lighter version in Ascher [3].

A relatively readable coverage of numerical methods for boundary value ODEs is [5]. An earlier, pioneering work is Keller [46]. A deeper, more encyclopedic treatment can be found in Ascher, Mattheij, and Russell [4].

There is vast literature on numerical methods for PDEs. Two recent textbooks are LeVeque [50] and [3]. We mention further only Trefethen [69] for spectral methods, Elman, Silvester, and Wathen [23] for an emphasis on linear iterative solvers, and Trottenberg, Oosterlee, and Schuller [71] for multigrid methods. Let us repeat that our present treatment of this topic in Section 16.8 is meant to give just a taste, and other texts (and more advanced courses) are required to cover it properly.