Chapter 10

Initial value Ordinary Differential Equations

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

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

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

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

Example 10.1
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 α the function

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

satisfies the ODE.

If it is given, in addition, that $y(0) = 1$, then $1 = 0 - 1 + \alpha e^0$, hence $\alpha = 2$ and the unique solution is

$$y(t) = t - 1 + 2e^{-t}.$$

♦

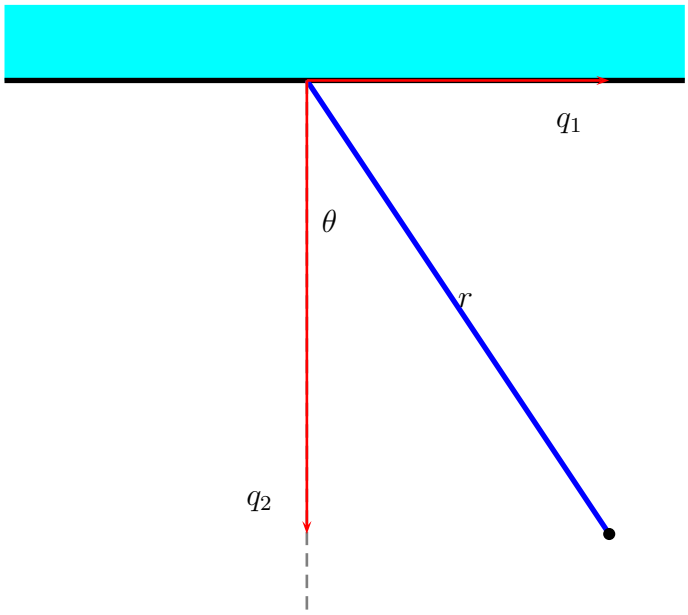

Figure 10.1: A simple pendulum.

It is convenient for presentation purposes to concentrate on just one (scalar) initial value ODE, because this makes the notation easier when introducing numerical methods. We should note though that scalar ODEs rarely appear in practice; they almost always arise as (sometimes large) systems. In the case of systems of ODEs we use vector notation and write our prototype ODE system as

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

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

In general, as in Example 10.1, there is a family of solutions depending on m parameters for this ODE. The solution becomes unique (under some mild assumptions) if m *initial conditions* are specified,

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

This is then an *initial value problem.*

Example 10.2
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 10.1.)

Denoting by θ 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 θ. The initial position and velocity configuration translate into values for $\theta(0)$ and $\theta'(0)$.

We can write this as a first order system: Let

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

Then $y_1' = y_2$ and $y_2' = -g \sin y_1$. The problem is then written as

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}), \quad t \geq 0, \\ \mathbf{y}(0) &= \mathbf{c}, \end{aligned}$$

where

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

♦

Whereas in the previous three chapters we denote the independent variable by x, here we denote it by t, because it is convenient to think of initial value problems as depending on *time*, as in Example 10.2. (However, t does not have to correspond to physical time in all applications.)

Going back to scalar ODE notation, we note that in principle we can write the initial value ODE in integral form,

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

This highlights both similarities to and differences from the problem of numerical integration considered in Sections 9.3–9.7. Specifically, on the one hand both prototype problems involve integration, but on the other hand, here we are recovering a function rather than a value (as in definite integration), and even more importantly, the integrand f depends on the unknown function.

10.1 Euler's method

Euler's method, which is also known as the **forward Euler** method (to distinguish it from its *backward Euler* counterpart, which we will discuss soon) is the simplest numerical method for approximately solving initial value ODEs. We first consider finding an approximate solution for the scalar initial value ODE at equidistant abscissae. Thus, define the points $t_0 = a$, $t_i = a +$

$ih,\ i = 0, 1, 2, \ldots, N$, where $h = \frac{b-a}{N}$ is the step size. Denote the approximate solution for $y(t_i)$ by y_i.

Recall from Section 9.1 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 *Euler method* which the approximate solution $\{y_i\}_{i=0}^N$ satisfies (and is defined 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 10.2. (We omit the details of the problem for which the graph was produced, since the points made in the figure are not specific to one particular 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
```

Thus we will have produced 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-h;
y = c;
for i=1:N
  t = t + h;
  y = y + h * f(t,y);
end
```

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

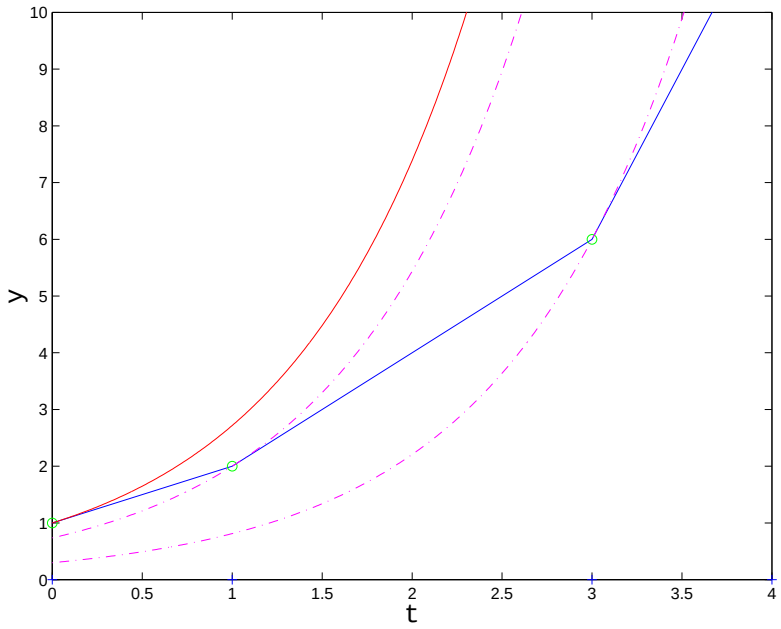

Figure 10.2: The forward Euler method. The exact solution is the curved solid line, and the numerical values obtained by the Euler scheme are circled and lie on a broken line that interpolates them. The broken line is tangential at the beginning of each step to the ODE trajectory passing through the corresponding point (dashed lines). Notice that the step size need not be uniform. In this example the second step size is double the first.

Example 10.3
Consider the simple, linear initial value ODE

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

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

Here, $f(t, y) = y$. Thus, 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 10.1. The (absolute) errors clearly reduce by a factor of roughly $1/2$ when h is cut from 0.2 to 0.1. We also note that the absolute error increases as t increases in this particular example. Even the relative error increases with t, although not as fast. ♦

The forward difference formula that leads to the (forward) Euler method

		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	0.050

Table 10.1: Absolute errors using the forward Euler method for the ODE $y' = y$.

can be replaced by a *backward* formula,

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

which leads to the **backward Euler** method:

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

One might be tempted to think of the backward Euler formula as a minor variation not much different from the forward scheme. But there is a substantial difference here: In the backward Euler formula, 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 point y_{i+1} itself, then the method is implicit and requires a solution of 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 obtained in previous iterations, such as y_i), 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. This is what has given the forward Euler method its great popularity in numerous areas of applications. The backward Euler, in contrast, while easy to understand conceptually, requires a deeper understanding of numerical issues, and cannot be programmed as easily and seamlessly as forward Euler can. However, as we shall see later, backward Euler and in general

implicit methods have numerical properties that in certain cases, and for certain applications, make them superior to explicit methods.

Local trunction error and global error

The **local truncation error** is the amount by which the exact solution fails to satisfy the difference equation written in divided difference form.

The concept is general; let us demonstrate it for the forward Euler method:

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

Note that $d_i = \frac{h}{2}y''(\xi_i) = \mathcal{O}(h)$. In general, if nothing goes wrong we expect that the **global error**

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

be of the same order, namely, we expect

$$e_i = \mathcal{O}(h).$$

We have seen such behaviour in Example 10.3. Let us show that this is true in general under very mild conditions on $f(t, y)$.

Since the local truncation error satisfies

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

we subtract the two expressions and obtain for the error $e_k = y(t_k) - y_k$ the difference formula

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

Assume that f satisfies a Lipschitz condition, i.e., there is a constant L such that

$$|f(t, v) - f(t, w)| \le L|v - w| \quad \text{for all } a \le t \le b,\ v, w \in \mathcal{D}$$

for some appropriate domain $\mathcal{D}$. Then we can write the error difference equation as

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

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

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

then set

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

It follows that

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

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

This provides a proof that the global error in Euler's method is indeed first order in h, as observed in Example 10.3. Note also that the bound may indeed 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.

Note:

Since deriving discretization formulae for differential equations involves numerical differentiation, there is an unavoidable roundoff error that behaves like $\mathcal{O}(h^{-1})$; recall the discussion in Section 9.2. However, roundoff error is typically a secondary concern here, especially when using double precision (which is the default in MATLAB). 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.

Example 10.4

A more challenging problem 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 = .005; t = 0:h:322;
y = y0 * ones(1,length(t));
for i = 2:length(t)
   y(:,i) = y(:,i-1) + h*hires(t(i-1),y(:,i-1));
end
plot(t,y(6,:))
```

(plus labelling) produces Figure 10.3. To gauge accuracy of the Euler method we repeat the solution process with a more accurate method (from Section 10.2) and regard the difference between these approximate solutions as the error for the less accurate forward Euler. Based on this the maximum absolute error occurs after 447 steps at $t_* = 2.235$, where $y(t_*) = .4483$ and $|y(t_*) - y_{447}| = 4.67 \times 10^{-4}$.

♦

10.2 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 obtain higher order methods there are extensions of Euler's method in several directions. The two most common directions lead to

- *Runge-Kutta methods.* A special class of these, not discussed here further, are *Extrapolation methods.*
- *Multistep methods.* These are discussed in Section 10.3.

Consider our prototype ODE 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 (never mind how we got it) and consider ways to obtain an approximation y_{i+1} to

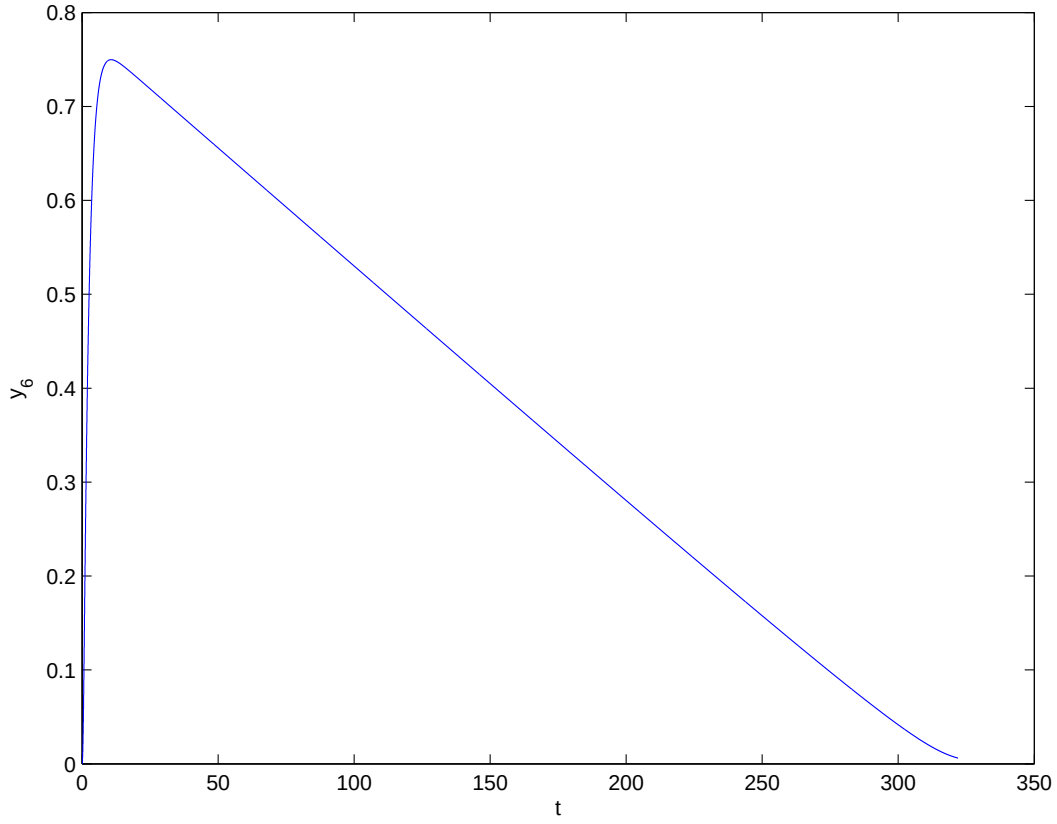

Figure 10.3: The 6th solution component of the HIRES model.

$y(t_{i+1})$ at the next point t_{i+1}. 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 a basic quadrature rule (Section 9.3). For instance, the *trapezoidal rule* gives

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

and the *midpoint rule* gives

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

where we denote

$$t_{i+1/2} = \frac{t_i + t_{i+1}}{2} = t_i + h/2.$$

The problem is that both these last two schemes (which are $\mathcal{O}(h^2)$-accurate) are *implicit*, recall the definition in Section 10.1. Thus, the evaluation of y_{i+1} requires solving a generally nonlinear equation for y_{i+1}. This is

a major difference between numerical quadrature and numerical integration of differential equations. For an ODE system of size m we get a nonlinear system of m equations to solve for a vector $\mathbf{y}_{i+1}$ at each step i. One would like to avoid this expense and complication, if possible (although not at any cost, as becomes clear in Section 10.4).

Thus, apply Euler's method to evaluate an approximation for y at $t_{i+1/2}$ first:

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

This is the **explicit midpoint** scheme, and it is an instance of an explicit Runge-Kutta method. It can be shown to have the local truncation error $d_i = \mathcal{O}(h^2)$. [1] 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 Euler's method per step.

The **classical Runge-Kutta** method is based on the Simpson quadrature rule, and uses four stages (four function evaluations per step) to achieve $\mathcal{O}(h^4)$ accuracy:

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

Showing that this formula is actually 4th order accurate is not a simple matter, unlike for the composite Simpson quadrature, and will not be discussed further.

It is common to abbreviate Runge-Kutta to RK. 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)
```

[1] The local truncation error is defined in general in the same way as it is defined for the forward Euler method in Section 10.1: It is the resulting residual when we insert the exact solution into the difference equation which the numerical solution satisfies.

```
%
% [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 clascical 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 = shiftdim(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
```

The appearance of methods of different orders in this section motivates us to explore the question how we can test that a particular method comes close to reflecting its order. A way to go about this is the following: If the error at fixed t is $e(h) \approx ch^q$ then with step size $2h$, $e(2h) \approx c(2h)^q \approx 2^q e(h)$. Thus, calculate

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

This *convergence rate*, or *observed order* is compared to the predicted order q of the given method. See Exercise 3.

Example 10.5
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 10.2.

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-3	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

Table 10.2: Errors and calculated convergence rates for the forward Euler, the explicit midpoint (RK2) and the classical Runge–Kutta (RK4) methods.

By using the approach described above for computing "rates", 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.

But bear in mind that for rougher accuracy (and for rougher ODEs) lower order methods may become more competitive.

♦

Example 10.6
Next, we unleash our function `rk4` on the following problem of size $m = 2$:

$$\begin{aligned} y_1' &= .25y_1 - .01y_1y_2, \quad y_1(0) = 80, \\ y_2' &= -y_2 + .01y_1y_2, \quad y_2(0) = 30. \end{aligned}$$

Integrating from $a = 0$ to $b = 100$ with step size $h = 0.01$, we plot the solution in Figure 10.4.

This is a simple *predator-prey* model, originally due to 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

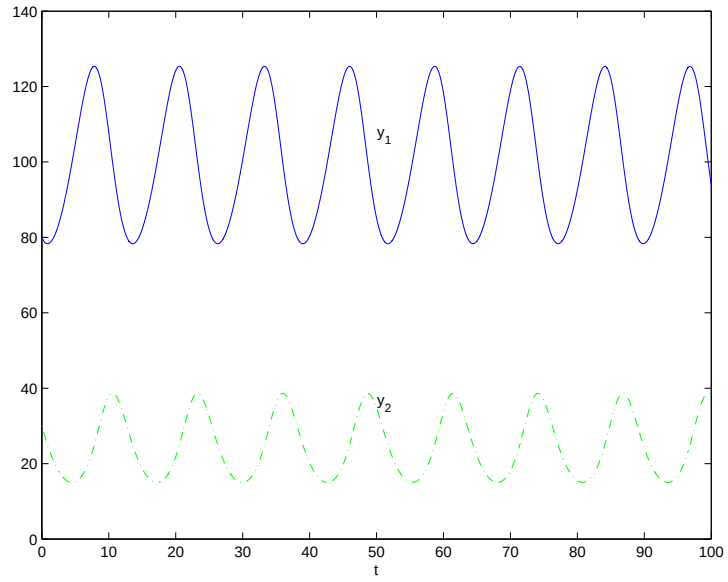

Figure 10.4: Predator-prey model: $y_1(t)$ is number of prey, $y_2(t)$ is number of perdator.

one predator species whose number at a any given time is $y_2(t)$. The number of predators whould 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 $y_2(t)$.

In Figure 10.5 we plot y_1 vs y_2. A *limit cycle* is obtained, suggesting that at least according to this simple model neither species will become extinct or grow unboundedly at any future time.

♦

In the 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 the **explicit, s-stage RK** method looks like

$$\begin{aligned} y_{i+1} &= y_i + h\sum_{j=1}^{s} b_j f(Y_j), \quad \text{where} \\ Y_j &= y_i + h\sum_{k=1}^{j-1} a_{jk} f(Y_k). \end{aligned}$$

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 $b_2 = 1$.

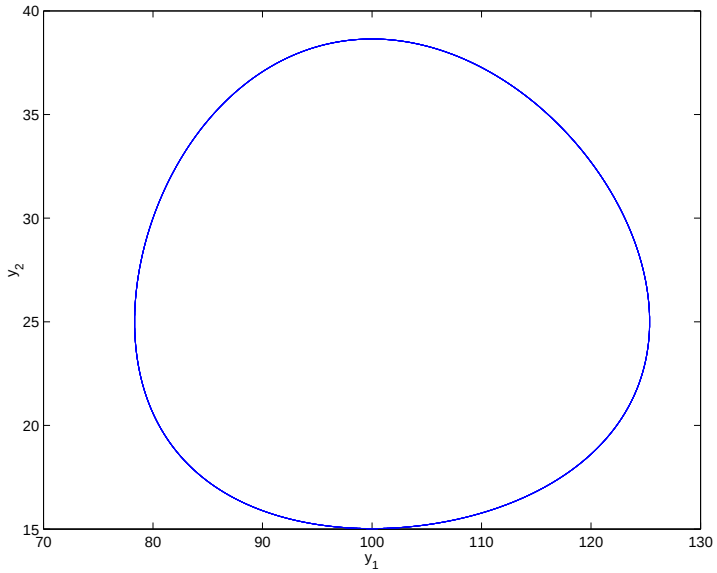

Figure 10.5: Predator-prey solution in phase plane: plotting $y_1(t)$ vs $y_2(t)$ yields a limit cycle.

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

$$\begin{aligned} y_{i+1} &= y_i + h\sum_{j=1}^{s} b_j f(Y_j), \quad \text{where} \\ Y_j &= y_i + h\sum_{k=1}^{s} a_{jk} f(Y_k). \end{aligned}$$

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 10.4).

The definitions of local truncation error and order, as well as the proof of convergence and global error bound given in Section 10.1, readily extend for any Runge-Kutta (RK) method.

Some of the advantages and disadvantages of Runge-Kutta methods are evident by now. Advantages are:

- 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 Runge-Kutta methods include:

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

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

Indeed, there are many problem instances such as those in Examples 10.2, 10.4 and 10.6 where there is no explicit dependence on t in $\mathbf{f}$. But 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.

- The number of function evaluations required – our measure of work in Sections 9.3–9.7 – is relatively high, compared to multistep methods.
- 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 below.)
- More involved and possibly more costly procedures are required for stiff problems, the topic of Section 10.4, than upon using multistep methods.

10.2.1 Error control and estimation

As in the case with quadrature and `quads` (Section 9.6), we wish to write a mathematical software routine where a user would specify $f(t, y)$, a, b, c and a tolerance Tol, and the routine would calculate $\{y_i\}_{i=0}^N$ accurate to within Tol. However, here the situation is significantly more complex than for numerical integration:

- The global error $e_i = y(t_i) - y_i$ 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 indicates, the global error may grow exponentially in time, which means that the contribution of each local error may also grow in time.

- If we use a method such as RK4 for a given step from t_i to t_{i+1} and then subdivide and apply RK4 on each subinterval to obtain an error estimate (analogously to what is described in Section 9.6 using the Simpson rule) then the same function values cannot be used and essentially everything must be recalculated afresh. Hence the process of local error estimation may be very expensive if done simplistically.

The first of these concerns is the more traumatic one. There are methods to estimate the global error by recalculating the entire solution on $[a, b]$, but we want a local error control, i.e. we want to adapt the step size locally! Relating this to the global error is hard, 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 Figure 10.6.

Thus, we consider the ith 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 schemes, one of order q and the other of order $q + 1$, to calculate two approximations y_{i+1} and $\hat{y}_{i+1}$ at t_{i+1}, both starting from y_i at t_i. Then we can estimate

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

So, at the ith step we calculate these two approximations and compare: If

$$|\hat{y}_{i+1} - y_{i+1}| \leq Tol$$

then the step is accepted: set $y_{i+1} \leftarrow \hat{y}_{i+1}$ (the more accurate of the two values calculated) 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 $l_{i+1} \approx ch^{q+1}$, upon decreasing h to a satisfactory $\tilde{h}$ the local error will become $c\tilde{h}^{q+1} \approx 0.9\ Tol$, where the factor 0.9 is for safety. Dividing, we get

$$\frac{\tilde{h}^{q+1}}{h^{q+1}} \approx \frac{0.9\ Tol}{|\hat{y}_{i+1} - y_{i+1}|}.$$

Thus, set

$$\tilde{h} = h \left(\frac{0.9\ Tol}{|\hat{y}_{i+1} - y_{i+1}|} \right)^{\frac{1}{q+1}}.$$

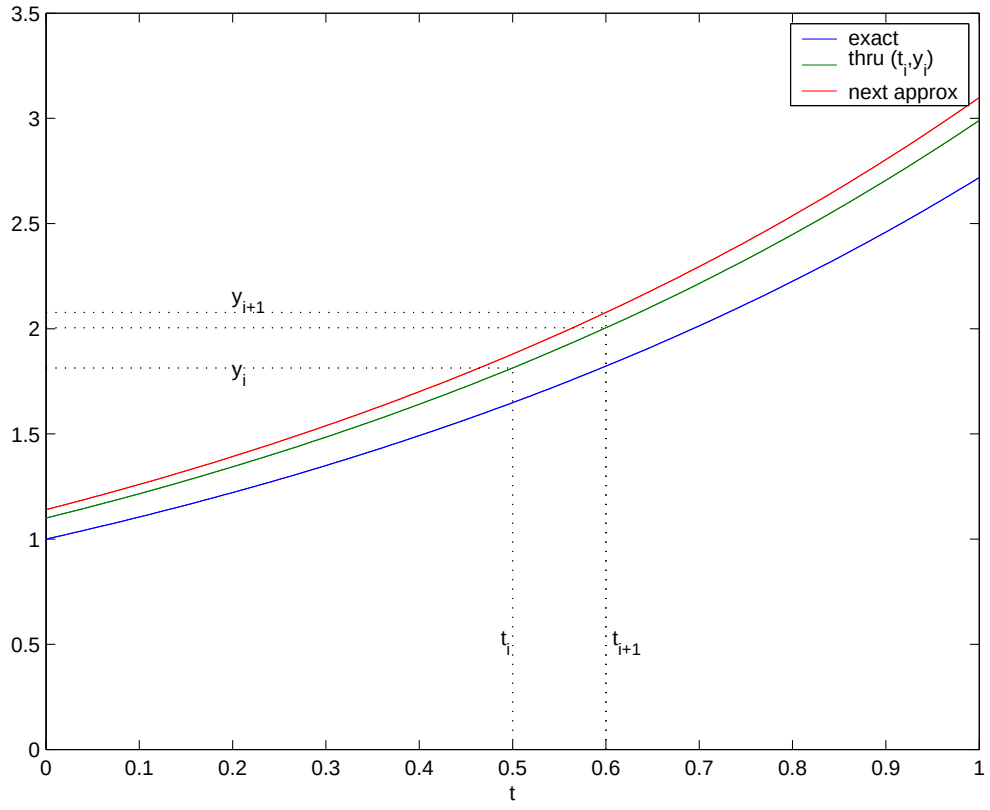

Figure 10.6: The exact solution $y(t_i)$ which lies on the lowest of the three curves is approximated by y_i which lies on the middle curve. If we integrate the next step exactly, starting from (t_i, y_i), then we obtain $(t_{i+1}, \bar{y}(t_{i+1}))$, which also lies on the middle curve. But we don't: 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.

We caution again that the meaning of Tol here is different from that in the adaptive quadrature section 9.6. Here we attmept to control the local, not the global error.

How is the value of h selected upon starting the ith 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 step size. For instance, if in the past two time steps no decrease was needed we can hazard a larger initial step size for the current step.

The only question left is how to choose the pair of formulae of orders q and $q+1$ wisely. This is achieved by searching for a pair of formulae 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 formulae are implemented in MATLAB's `ODE45`. See Exercise 8 for a detailed example demonstrating the power of a variable step code.

Example 10.7 (The Lorenz equations)
The famous Lorenz equations provide a simple example of a chaotic system. They are given by

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}) = \begin{pmatrix} \sigma(y_2 - y_1) \\ ry_1 - y_2 - y_1 y_3 \\ y_1 y_2 - by_3 \end{pmatrix},$$

where σ, r, b are positive parameters. Following Lorenz we set $\sigma = 10$, $b = 8/3$, $r = 28$. We then integrate, starting from $\mathbf{y}(0) = (0, 1, 0)^T$, using ODE45. Plotting $y_3(t)$ vs. $y_1(t)$ we obtain the famous "butterfly" depicted in Figure 10.7. It is interesting to note that upon changing the error tolerance and rerunning the integration software ODE45, say for $0 \le t \le 100$, the solution at $t = 100$ is quite different, as befits a chaotic system. And yet, the delicate features of the phase plane plot in Figure 10.7 are reproduced, even without pointwise accuracy of the numerical solution. Sometimes, there is free lunch!

♦

10.3 Multistep methods

The basic idea of multistep methods is very simple: once we are in the midst of integrating the ODE we really have at the start of step i knowledge not only of y_i (the approximate solution at t_i which is all that Runge-Kutta

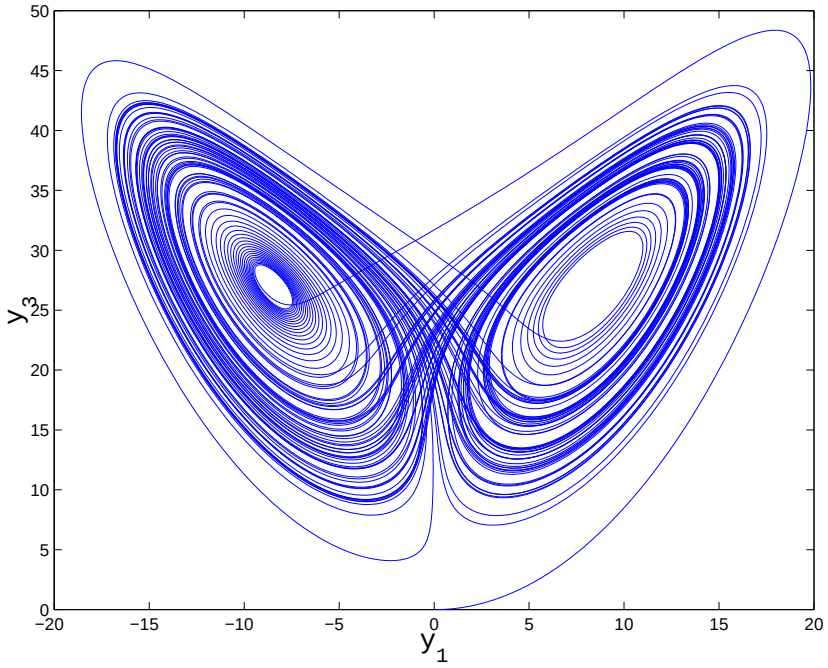

Figure 10.7: Lorenz "butterfly" in the $y_1 \times y_3$ plane.

Note: It is possible to read almost all of Section 10.4 without going first in detail through Section 10.3.

methods use) 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 in order to obtain cheap yet accurate approximations for the next unknown, y_{i+1}.

Assuming (for simplicity) a uniform grid, $t_i = ih$, $i = 0, 1, \ldots$, where h is the step size in time t, these methods use solution values at previous time steps to generate high accuracy approximations of the given ODE. 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 α_j, β_j are coefficients; 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 β_0: The method is *explicit* if $\beta_0 = 0$ and *implicit* otherwise. It is called *linear* because, unlike general Runge–Kutta, the expression in the multistep formula is linear in f. (Of course f itself may still be nonlinear in y.)

Let us define the *local truncation error* for the general multistep method

as

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

This is the amount by which the exact solution fails to satisfy the difference equations, divided by h. Thus, the method has *accuracy order* q if for all problems with sufficiently smooth exact solutions $y(t)$,

$$d_i = O(h^q).$$

Example 10.8
The forward Euler method is a particular instance of a linear multistep method, with $s = 1$, $\alpha_1 = -1$, $\beta_1 = 1$, and $\beta_0 = 0$ as an explicit method ought to have.

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

The other RK methods that we have seen in Section 10.2 are not linear multistep methods.

♦

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

The Adams methods are derived by considering integrating

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

and 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,

$\alpha_0 = 1$, $\alpha_1 = -1$, and $\alpha_j = 0$, $j > 1$.

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

Example 10.9
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:

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

The formula is, therefore,

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

♦

Table 10.3 gives the coefficients of the Adams–Bashforth methods for s up to 6. For $s = 1$ we obtain the forward Euler method.

q	s	$j \rightarrow$	1	2	3	4	5	6
1	1	β_j	1					
2	2	$2\beta_j$	3	-1				
3	3	$12\beta_j$	23	-16	5			
4	4	$24\beta_j$	55	-59	37	-9		
5	5	$720\beta_j$	1901	-2774	2616	-1274	251	
6	6	$1440\beta_j$	4277	-7923	9982	-7298	2877	-475

Table 10.3: Coefficients of Adams–Bashforth methods up to order 6.

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 $O(h)$ apart we expect (and obtain) order of accuracy $q = s + 1$. For instance, Exercise 5 shows that, upon passing a quadratic through the points (t_{i+1}, f_{i+1}), (t_i, f_i) and (t_{i-1}, f_{i-1}), the following formula is obtained:

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

Table 10.4 gives the coefficients of the Adams–Moulton methods for s up to 5. Note that there are two one-step methods here: backward Euler

q	s	$j \rightarrow$	0	1	2	3	4	5
1	1	β_j	1					
2	1	$2\beta_j$	1	1				
3	2	$12\beta_j$	5	8	-1			
4	3	$24\beta_j$	9	19	-5	1		
5	4	$720\beta_j$	251	646	-264	106	-19	
6	5	$1440\beta_j$	475	1427	-798	482	-173	27

Table 10.4: Coefficients of Adams–Moulton methods up to order 6.

$(s = q = 1)$,

$$y_{i+1} = y_i + hf_{i+1},$$

and trapezoidal $(s = 1, q = 2)$,

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

Example 10.10
For 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 Tables 10.5 and 10.6 which provide a comparison with Table 10.2 of Example 10.5. 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 behaviour.

We can see that the observed order of the methods is as advertised. The error constant (i.e. c in $e(h) = ch^q$) is smaller for the Adams-Moulton method of order $q > 1$ than for the corresponding Adams-Bashforth 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 trapezoidal method in Table 10.6. The error constant in RK4 is smaller than that in the corresponding 4th order methods here. ♦

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 10.5: Example 10.10: Errors and calculated convergence rates for Adams–Bashforth methods; (k, q) denotes the k-step method of order q.

Predictor-corrector methods

The great advantage of multistep methods is their cost in terms of function evaluation: 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 formulae there are some serious limitations in terms of absolute stability – a concept made clear in the next section. For now, suffice it to say that Adams-Bashforth methods are usually not used as stand-alone discretizations with $s > 2$.

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 equations for y_{i+1}. Worse, for ODE systems we generally have a nonlinear system of algebraic equations at each step. Fortunately, everything implicit and nonlinear is multiplied by h in the formula (because only f, and not y', may be nonlinear in the ODE), so for h sufficiently small a simple fixed point iteration converges under fairly mild conditions (unless the ODE system is *stiff* – see Section 10.4).

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 formula based on Adams-Moulton.

But next, note that the fixed point iteration need not be carried to convergence: alll it yields in the end is an approximation for $y(t_{i+1})$ anyway. The most popular **predictor-corrector** variant, denoted PECE, reads as

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

Table 10.6: Example 10.10: Errors and calculated convergence rates for Adams–Moulton methods; (k, q) denotes the k-step method of order q.

follows:

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 is carried out in preparation for the next time step and in order to maintain an acceptable measure of absolute stability.

Example 10.11
Combining the two-step Adams–Bashforth formula with the second order one-step Adams–Moulton formula (i.e., the trapezoidal method), we obtain the following method for advancing one time step.

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

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

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

This method really looks like an "explicit-trapezoidal" variant of a Runge-Kutta method. The power of the predictor-corrector approach really shines when more steps are involved, because the same simplicity – and cost of two function evaluations per step! – remains for higher order methods, whereas higher order Runge-Kutta methods get significantly more complex and expensive per step. ♦

In the common situation where the orders of the predictor formula and of the corrector formula are the same, 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. Error control in the spirit of Subsection 10.2.1 is then facilitated. In fact, it is also possible to vary the order (by varying s) of the PECE pair adaptively. Varying the step size, however, is overall more complicated (even though certainly possible) than in the case of one-step, RK methods.

Comparing multistep methods to Runge-Kutta methods, the comments we have made just before Subsection 10.2.1 are relevant (except that now what was worse in RK is better here and what was better in RK is relatively worse here). Briefly, the important advantages are the cheap high order PECE pairs with the local error estimation that comes for free. The important disadvantages are the need for additional initial values (all s initial values must be $O(h^q)$-accurate for a method of order q - they are obtained using another method which requires fewer steps), 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 more recently taken the popularity lead. This is true except for stiff problems, for which BDF methods are still the bee's knee.

Backward differentiation formulae (BDF)

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 previous points plus the next one, $t = t_{i+1}, t_i, t_{i-1}, \ldots, t_{i+1-s}$, and differentiating it. This gives an implicit method of accuracy order $q = s$. Table 10.7 gives the coefficients of the BDF methods for s up to 6. For $s = 1$ we obtain again the backward Euler method.

q	s	β_0	α_0	α_1	α_2	α_3	α_4	α_5	α_6
1	1	1	1	-1					
2	2	$\frac{2}{3}$	1	$-\frac{4}{3}$	$\frac{1}{3}$				
3	3	$\frac{6}{11}$	1	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$			
4	4	$\frac{12}{25}$	1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$		
5	5	$\frac{60}{137}$	1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	
6	6	$\frac{60}{147}$	1	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$

Table 10.7: Coefficients of BDF methods up to order 6.

Example 10.12
Continuing with Example 10.10, we now compute errors for the same ODE problem, namely, $y' = -y^2,\ y(1) = 1$, using the BDF methods. The results in Table 10.8 below are to be compared against those in Tables 10.5 and 10.6.

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

Table 10.8: Example 10.12: Errors and calculated convergence rates for BDF methods; (k, q) denotes the k-step method of order q.

The results are clearly comparable, although the error constants for methods of similar order are worse here. Again, the advantage of BDF methods comes to life in the context of stiff problems, which is our next topic.

♦

10.4 Absolute stability and stiffness

The methods described in Sections 10.2 and 10.3 are routinely used in many applications and they often give satisfactory results in practice. An exception is the case of stiff problems, where explicit Runge-Kutta methods are forced to use a very small step size and thus become unreasonably expensive. (Adams multistep methods meet a similar fate.) But before we define stiffness, we should introduce a few important notions related to stability.

Absolute stability

To understand the phenomenon involved, consider the *test equation*

$$y' = \lambda y$$

where λ is a 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

$$\begin{aligned} y_{i+1} &= y_i + h\lambda y_i = (1 + h\lambda) y_i \\ &= \ldots = (1 + h\lambda)^{i+1} y(0). \end{aligned}$$

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:

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

For the (forward) Euler method this corresponds to the requirement

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

This is a requirement of **absolute stability**.

Of course the test equation is very simple. In general we consider a nonlinear ODE system

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

Here, absolute stability properties are determined by the Jacobian matrix,

$$J(\mathbf{y}) = \frac{\partial \mathbf{f}(\mathbf{y})}{\partial \mathbf{y}}.$$

A surprisingly good estimate of the stability properties of a numerical method for the nonlinear system is obtained by considering the test equation for each of the eigenvalues of the Jacobian matrix. One important generalization to the test equation that this implies, though, is that we must consider a

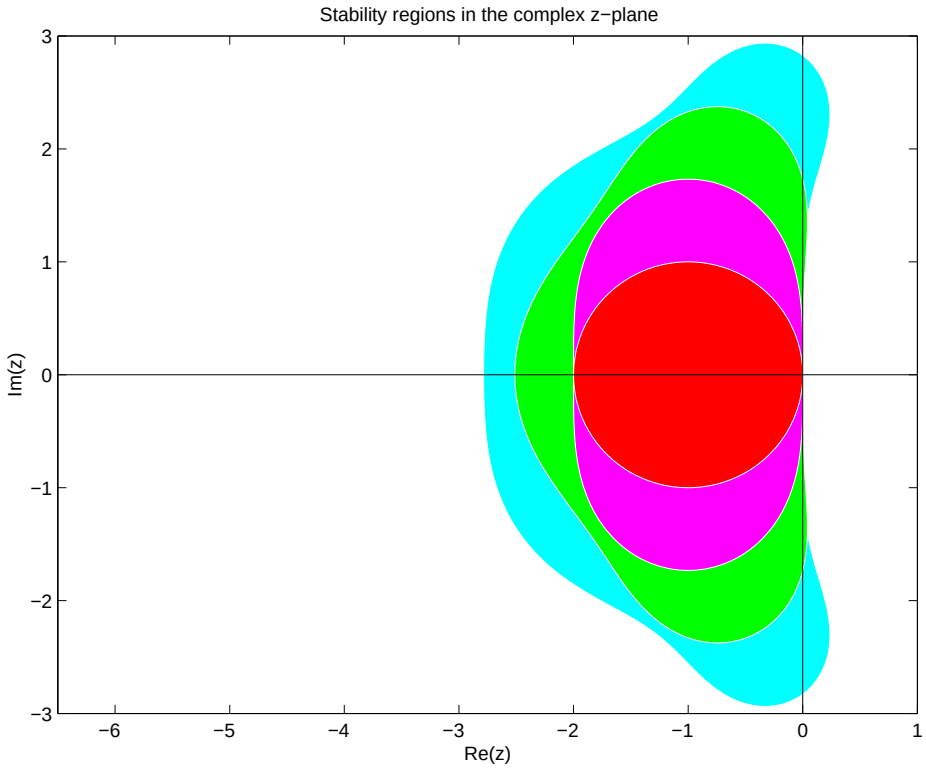

Figure 10.8: Stability regions for q-stage explicit Runge–Kutta methods of order q, $q = 1, 2, 3, 4$. The inner circle corresponds to forward Euler, $q = 1$. The larger q is, the larger the stability region. Note the "ear lobes" of the fourth-order method protruding into the right half-plane.

complex scalar λ, because eigenvalues are in general complex scalars. See Example 10.13 below. Thus, instead of an absolute stability bound we are really looking at an **absolute stability region** in the complex plane.

For instance, the forward Euler restriction

$$|1 + h\lambda| \leq 1$$

is written as

$$|1 + z| \leq 1$$

for the variable $z = h\lambda$. Clearly, the points z satisfying this constraint are all inside (and on the boundary of) the circle of radius 1 centered at the point $(-1, 0)$ in the complex z-plane. Figure 10.8 depicts stability regions for explicit Runge-Kutta methods of orders up to 4.

Example 10.13
Let us return to the mildly nonlinear problem of Example 10.4. The 8×8

Jacobian matrix is

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

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 are (up to the first few leading digits)

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

There is a conjugate pair of eigenvalues, while the rest are real. To get them all into the circle 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. The good results in Example 10.4 are obtained by carrying out the integration process with the much smaller $h = .005$.

Indeed it turns out that at $t \approx 10.7$ the eigenvalues are approximately

$$-211.7697, -10.4841, -8.2780, -2.3923, -2.1400, -0.4907, -3.e{-}5, -3.e{-}12.$$

The stability bound that the most negative eigenvalue yields is

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

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

♦

Stiffness

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

The problem of Examples 10.4 and 10.13 is stiff. The simple problem of the following example is even stiffer.

Example 10.14
For the ODE

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

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

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

for *any* reasonable accuracy! ♦

From Figure 10.8 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 trapezoidal or implicit midpoint become more attractive then, because their region of absolute stability is the entire left half z-plane. (Please verify this.)

So does the backward Euler method

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

which we briefly introduced in Section 10.1. For the test equation we obtain $y_{i+1} = y_i + h\lambda y_{i+1}$, 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. Even more importantly, if the real part of z is negative ($\Re(z) < 0$) then, as $\Re(z) \to -\infty$, $\frac{1}{1-h\lambda} \to 0$. Therefore $y_{i+1} \to 0$, as does the exact solution $y(h)$ under these circumstances. This is called *L-stability*.

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 a 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 Chapter 6. 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.

For instance, applying the backward Euler method to our ODE system we are facing at the ith step the following system of algebraic equations,

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

Recalling Newton's method from Section 6.1 and our current notation for the Jacobian matrix of $\mathbf{f}(\mathbf{y})$ we have an iterative method for $\mathbf{y}_{i+1}$. For a good starting iterate we can take $\mathbf{y}_{i+1}^{(0)} = \mathbf{y}_i$. Then the Newton iteration is:

for $k = 0, 1, \ldots,$ until convergence:
solve the linear system

$$(I - hJ(\mathbf{y}_{i+1}^{(k)}))\mathbf{p}_k = -\mathbf{g}(\mathbf{y}_{i+1}^{(k)})$$

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

All this is to be done at every integration step! The saving grace is that the initial guess $\mathbf{y}_i$ 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 Newton iteration due to the quadratic convergence rate of Newton's method. Therefore, one such iteration is usually sufficient. (Indeed, in sophisticated codes less than one Jacobian evaluation per step is applied on average.) This then yields the **semi-implicit backward Euler** method

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

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 10.15
The problem of Examples 10.4 and 10.13 is stiff, to recall. With a uniform step size the forward Euler method requires $322/.005 = 64,400$ integration steps in Example 10.4.

A solution which is qualitatively similar to the one displayed in Figure 10.3 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 10.13.

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, BDF are the methods of choice, although there are also some popular implicit Runge-Kutta methods. Their implementation 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. Moreover, their behaviour (especially the two- and three-step methods) is generally similar to that of backward Euler as well.

10.5 Exercises

1. The following ODE system:

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

   where α and β 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.]

2. 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 θ. 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 so as to ensure that the obtained approximate solution looks right, i.e., that the approximate curve closes rather than spirals.

Carry out this integration using a uniform step size $h = .02$ for $0 \le \theta \le 120$, applying forward Euler, backward Euler, and the 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 order of the methods.]

3. The convergence rate (observed order) defined prior to Example 10.3 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,

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

4. 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 any of the discretization methods in this chapter to the ODE in t with step size $h = \Delta t$ is equivalent to applying the same method to the ODE in τ 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.

5. Derive the 2-step Adams-Moulton formula

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

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

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

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, $\boldsymbol{\nabla} U(\mathbf{q}) = (\frac{\partial U}{\partial q_1}, \ldots, \frac{\partial U}{\partial q_m})^T$. 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

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

is constant for all $t > 0$.

(b) Use a library nonstiff Runge–Kutta code based on a 4(5) embedded pair 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 $= 1.\text{e}-4$, 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.

7. 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:

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

i.e., 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.

8. The following classical example from astronomy gives a strong motivation to integrate initial value ODEs with error 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

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

Starting with the initial conditions

$$\begin{aligned} &u_1(0) = 0.994,\ u_2(0) = 0,\ u_1'(0) = 0, \\ &u_2'(0) = -2.00158510637908252240537862224, \end{aligned}$$

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.

The orbit is depicted in Figure 10.9. It was obtained using `ODE45` with a local error tolerance $1.e-6$. This necessitated 204 time steps.

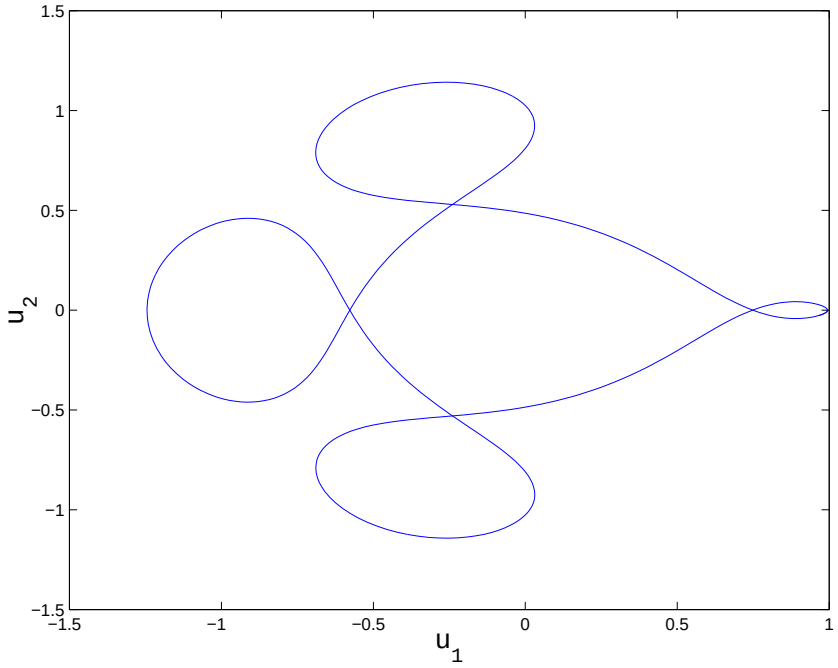

Figure 10.9: Astronomical orbit using ODE45.

Using the classical Runge–Kutta method of order 4, integrate this problem on $[0, 17.1]$ with a *uniform* step size, using 100, 1000, 10,000, and 20,000 steps. Plot the orbit for each case. How many uniform steps are needed before the orbit appears to be *qualitatively* correct?

10.6 Additional notes and references

The material covered in this brief chapter is a distillation of a few topics from the first five chapters of Ascher & Petzold [2]. More is covered in the books by Hairer, Norsett and Wanner [20] and by Hairer & Wanner [21].

Many iterative methods in optimization and linear algebra, including most of those described in Chapters 2, 5 and 6, can be written as

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

where α_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.

The problem described in Example 10.4 is one in a set of initial value ODE applications used for testing research codes and maintained in

`http://pitagora.dm.uniba.it/~testset/`

by F. Mazzia and F. Iavernaro.

This chapter discusses only methods for initial value ODEs. For a comprehensive (as distinct from user friendly) treatise of numerical methods for boundary value ODEs see Ascher, Mattheij & Russell [1].

A lot of recent attention has been devoted to numerical methods for dynamical systems, see Stuart & Humphries [34] and Strogatz [33]. A lot of recent research work has been carried out in the context of *Geometric integration*, and we refer to Hairer, Lubich & Wanner [19] for a comprehensive account on this topic.