Numerical methods for ordinary differential equations

Ulrik Skre Fjordholm

May 1, 2018

Chapter 1

Introduction

Consider the ordinary differential equation (ODE)

$$\dot{x}(t) = f(x(t), t), \qquad x(0) = x_0 \tag{1.1}$$

where $x_0 \in \mathbb{R}^d$ and $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$. Under certain conditions on f there exists a unique solution of (1.1), and for certain types of functions f (such as when (1.1) is separable) there are techniques available for computing this solution. However, for most "real-world" examples of f, we have no idea how the solution actually looks like. We are left with no choice but to *approximate* the solution $x(t)$.

Assume that we would like to compute the solution of (1.1) over a time interval $t \in [0, T]$ for some $T > 0$[1]. The most common approach to finding an approximation of the solution of (1.1) starts by partitioning the time interval into a set of points $t_0, t_1, \ldots, t_N$, where $t_n = nh$ is a *time step* and $h = \frac{T}{N}$ is the *step size*[2]. A numerical method then computes an approximation of the actual solution value $x(t_n)$ at time $t = t_n$. We will denote this approximation by y_n. The basis of most numerical methods is the following simple computation: Integrate (1.1) over the time interval $[t_n, t_{n+1}]$ to get

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} f(x(s), s)\, ds. \tag{1.2}$$

Although we could replace $x(t_n)$ and $x(t_{n+1})$ by their approximations y_n and y_{n+1}, we cannot use the formula (1.2) directly because the integrand depends on the exact solution $x(s)$. What we can do, however, is to approximate the integral in (1.2) by some *quadrature rule*, and then approximate x at the quadrature points. More specifically, we consider a quadrature rule

$$\int_{t_n}^{t_{n+1}} g(s)\, ds \approx (t_{n+1} - t_n) \sum_{k=1}^{K} b_k g(s^{(k)})$$

where g is any function, $b_1, \ldots, b_k \in \mathbb{R}$ are *quadrature weights* and $s^{(1)}, \ldots, s^{(K)} \in [t_n, t_{n+1}]$ are the *quadrature points*. Two basic requirements of this approximations is that if g is nonnegative then the quadrature approximation is also nonnegative, and that the rule correctly integrates constant functions. It is straightforward to see that these requirements translate to

$$b_1, \ldots, b_k \geqslant 0 \qquad \text{and} \qquad b_1 + \cdots + b_K = 1.$$

[1] In this note we will only consider a time interval $t \in [0, T]$ for some positive endtime $T > 0$, but is it fully possible to work with negative times or even the whole real line $t \in \mathbb{R}$.

[2] The more sophisticated *adaptive time stepping* methods use a step size h which can change from one time step to another. We will not consider such methods in this note.

Some popular methods include the midpoint rule ($K = 1$, $b_1 = 1$, $s^{(1)} = \frac{t_n + t_{n+1}}{2}$), the trapezoidal rule ($K = 2$, $b_1 = b_2 = \frac{1}{2}$, $s^{(1)} = t_n$, $s^{(2)} = t_{n+1}$) and Simpson's rule ($K = 3$, $b_1 = b_3 = \frac{1}{6}$, $b_2 = \frac{4}{6}$, $s^{(1)} = t_n$, $s^{(2)} = \frac{t_n + t_{n+1}}{2}$, $s^{(3)} = t_{n+1}$). As you might recall, these three methods have an error of $O(h^3)$, $O(h^3)$ and $O(h^4)$, respectively, where $h = t_{n+1} - t_n$ denotes the length of the interval[3].

Applying the quadrature rule to (1.2) yields

$$y_{n+1} = y_n + h \sum_{k=1}^{K} b_k f\left(y_n^{(k)}, s_n^{(k)}\right) \tag{1.3}$$

where $s_n^{(k)} \in [t_n, t_{n+1}]$ are the quadrature points and $y_n^{(k)} \approx x\left(s_n^{(k)}\right)$. The initial approximation y_0 is simply set to the initial data prescribed in (1.1), $y_0 = x_0$.

If the quadrature points $y_n^{(k)}, s_n^{(k)}$ are either y_n, t_n or y_{n+1}, t_{n+1} (or some combination of these) then we can solve (1.3) for y_{n+1} and get a viable numerical method for (1.1). Two such methods, the explicit and implicit Euler methods, are the topic of Chapter 2. However, if we want to construct more accurate numerical methods then we have to include quadrature points at times $s_n^{(k)}$ which lie strictly between t_n and t_{n+1}, and consequently we need some way of computing the intermediate values $y_n^{(k)} \approx x\left(s_n^{(k)}\right)$. A systematic way of computing these points is the so-called Runge–Kutta methods. These are methods which converge to the exact solution much faster than the Euler methods, and will be the topic of Chapter 4.

Some natural questions arise when deriving numerical methods for (1.1): How large is the approximation error for a fixed step size $h > 0$? Do the computed solutions converge to $x(t)$ as $h \to 0$? How fast do they converge, and can we increase the speed of convergence? Is the method stable? The purpose of these notes is to answer all of these questions for some of the most commonly used numerical methods for ODEs.

Basic assumptions

In this note we will assume f is continuously differentiable, is bounded and has bounded first derivatives—that is, we assume that $f \in C^1(\mathbb{R} \times \mathbb{R}^d, \mathbb{R}^d)$ and that there are constants $M > 0$ and $K > 0$ such that

$$\sup_{\substack{x \in \mathbb{R}^d \\ t \in \mathbb{R}}} |f(x,t)| \leqslant M, \tag{1.4a}$$

$$\sup_{(x,t)\mathbb{R}^d \times \mathbb{R}} \left\| D_x f(x,t) \right\| \leqslant K, \tag{1.4b}$$

$$\sup_{(x,t)\mathbb{R}^d \times \mathbb{R}} \left| \frac{\partial f}{\partial t}(x,t) \right| \leqslant K. \tag{1.4c}$$

(Here, $D_x f$ denotes the Jacobian matrix of f, $\left(D_x f(x,t)\right)_{i,j} = \frac{\partial f^i}{\partial x^j}(x,t)$, and $\|\cdot\|$ denotes the matrix norm.) In particular, f is Lipschitz in x with Lipschitz constant K, so these assumptions guarantee that Picard iterations converge, and we can conclude that there exists a solution for all times $t \in \mathbb{R}$. Uniqueness of the solution follows from Lipschitz boundedness of f together with Gronwall's lemma.

[3]Here we use the "Big O notation". If e_h is some quantity depending on a parameter $h > 0$ (such as the error in a numerical approximation), then $e_h = O(h^p)$ means that there exists some constant $C > 0$ such that for small values of h, we can bound the error as $|e_h| \leqslant Ch^p$.

The assumptions in (1.4) can be replaced by *local* bounds, although one then needs to be more careful in some of the computations in this note. In particular, the solution might not exist for all times. Without going into detail we state here that all the results in this note are true (with minor modifications) assuming only local versions of the bounds (1.4).

Chapter 2

Euler's methods

The absolutely simplest quadrature rule that we can use in (1.3) is the one-point method $K = 1$, $b_1 = 1$, $s_n^{(1)} = t_n$. This yields the *forward Euler* or *explicit Euler* method

$$y_{n+1} = y_n + hf\big(y_n, t_n\big). \tag{2.1}$$

As its name implies, this method is *explicit*, meaning that the approximation y_{n+1} at the next time step is given by a formula depending only on the approximation y_n at the current time step. Explicit methods are easy to use on a computer because we can type the formula more or less directly in the source code. Another popular one-point quadrature is $K = 1$, $b_1 = 1$, $s_n^{(1)} = t_{n+1}$, which gives the *backward Euler* or *implicit Euler* method

$$y_{n+1} = y_n + hf\big(y_{n+1}, t_{n+1}\big). \tag{2.2}$$

This method is *implicit* in the sense that one has to solve an algebraic relation in order to find y_{n+1} as a function of only y_n. Both of the Euler methods can be seen as first-order Taylor expansions of $x(t)$ around $t = t_n$ and $t = t_{n+1}$, respectively.

2.1 Example. Consider the ODE (1.1) with $n = 1$ and $f(x) = ax$ for some $a \in \mathbb{R}$. The forward Euler method is

$$y_{n+1} = y_n + hf(y_n) = y_n\big(1 + ah\big).$$

The implicit Euler method is

$$y_{n+1} = y_n + hf(y_{n+1}) = y_n + ahy_{n+1},$$

and solving for y_{n+1} yields the explicit expression

$$y_{n+1} = \frac{y_n}{1 - ah}.$$

In both cases the method can be written as the difference equation $y_{n+1} = by_n$, whose solution is $y_n = b^n x_0$. Thus, the explicit Euler method can be written as $y_n = (1 + ah)^n x_0$ and the implicit Euler method as $y_n = (1 - ah)^{-n} x_0$. Using the fact that $h = \frac{T}{N}$, it is a straightforward exercise in calculus to show that for either scheme, the endtime solution y_N converges to the exact value $x(T) = e^{aT} x_0$ as $N \to \infty$.

Figure 2.1 show a computation with both schemes using the parameter $a = 1$ and initial data $x_0 = 1$. While the error in both schemes increase over time, this error decreases as the resolution N is increased.

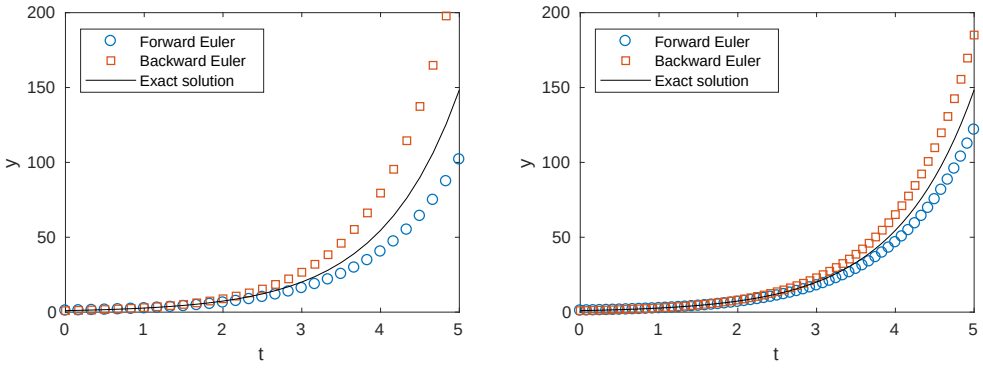


Figure 2.1: Forward and backward Euler computations for the linear, one-dimensional ODE in Example 2.1 up to time $T = 5$ using $N = 30$ (left) and $N = 60$ (right) time steps.

2.2 Example. Consider an object with mass m which is hanging from a spring secured to the ceiling. We let $q(t)$ denote the extension of the spring at time t, and fix the coordinate system so that $q = 0$ is the position at which the object is at rest, and the positive q-axis points vertically down. Hooke's law says that the force by the spring on the object is proportional to the extension of the spring: $F = -kq(t)$, where $k > 0$ is the *stiffness constant* of the spring. Applying Newton's second law $F = ma = m\ddot{q}$ gives us the second-order ODE

$$m\ddot{q} = -kq.$$

We convert this second-order equation to a system of first-order equations by introducing another unknown, the momentum $p(t) = m\dot{q}(t)$. This gives us the first-order ODE

$$\begin{cases} \dot{p} = -kq \\ \dot{q} = \frac{1}{m} p \end{cases} \tag{2.3}$$

We recognize this as a Hamiltonian system with Hamiltonian $H(p,q) = \frac{k}{2}q^2 + \frac{1}{2m}p^2$, which means that we can write

$$\begin{cases} \dot{p} = -\frac{\partial H}{\partial q}(p,q) \\ \dot{q} = \frac{\partial H}{\partial p}(p,q), \end{cases}$$

and the function H can be interpreted as the total energy of the system. The ODE (2.3) is often called the *harmonic oscillator*.

We consider now a numerical method for (2.3). Let us first write the system as

$$\dot{x} = Ax, \qquad \text{where } x = \begin{pmatrix} p \\ q \end{pmatrix} \text{ and } A = \begin{pmatrix} 0 & -k \\ 1/m & 0 \end{pmatrix}.$$

The forward Euler method (2.1) is then

$$y_{n+1} = y_n + hAy_n = (I_2 + hA)y_n$$

where I_2 is the identity matrix in $\mathbb{R}^{2\times 2}$, and the implicit Euler method (2.2) is

$$y_{n+1} = y_n + hAy_{n+1} \qquad \Rightarrow \qquad y_{n+1} = (I_2 - hA)^{-1} y_n.$$

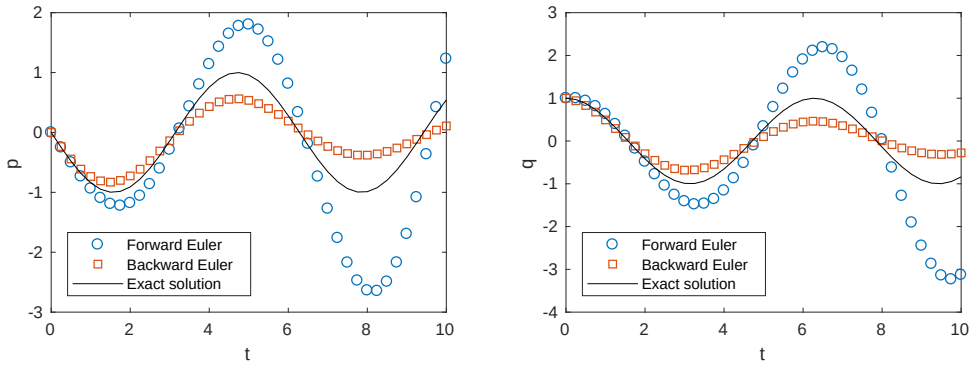


Figure 2.2: Forward and backward Euler computations for (2.3) using $T = 10$ and $N = 40$.

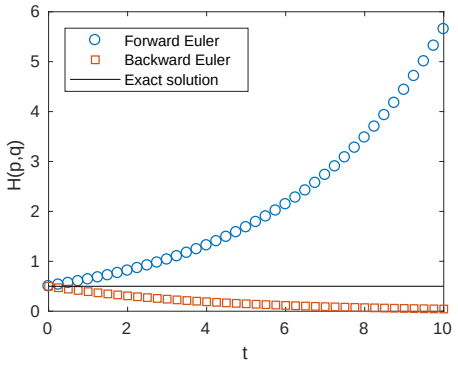


Figure 2.3: The total energy $H(p, q)$ for the computations in Example 2.2.

Figure 2.2 shows the numerically computed solutions for parameter values $m = k = 1$ and initial data $p = 0$, $q = 1$. The exact solution consists of sinusoidal waves (exercise for the reader!). As can be seen in the plots, the explicit Euler method overshoots the exact solutions, while the implicit Euler method undershoots. The same effect is seen in Figure 2.3, where we plot the Hamiltonian $H(p, q)$ as a function of time. Although the total energy $H(p, q)$ should be constant in time, we see that the explicit Euler method produces energy, while the implicit Euler dissipates energy over time. We will get back to the issue of Hamiltonian systems and energy preservation in Chapter 5.

2.3 Example. The harmonic oscillator in Example 2.2 is somewhat unrealistic, physically speaking, because there is no friction in the system. We can add friction to our model by modeling it as a force which is proportional to the speed and which acts opposite to the direction of travel:

$$F_{\text{friction}} = -c\dot{q},$$

where $c \geqslant 0$ is the *friction coefficient.* Applying Newton's second law again yields the ODE

$$m\ddot{q} = -kq - c\dot{q}.$$

Introducing the unknown $p = mq$, we can convert this second-order ODE to the first-order system

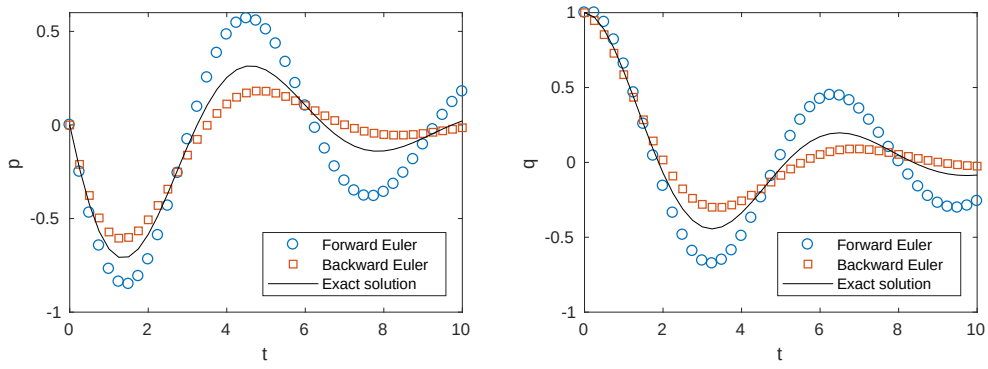


Figure 2.4: Forward and backward Euler computations for the damped harmonic oscillator (2.4) using $T = 10$ and $N = 40$.

of ODEs

$$\begin{cases} \dot{p} = -kq - \frac{c}{m}p \\ \dot{q} = \frac{1}{m}p. \end{cases} \tag{2.4}$$

Since friction removes energy from the system (converting it to heat, sound, etc.), the total energy H is no longer preserved, and so the above system is *not* Hamiltonian. (Exercise for the reader: Compute $\frac{d}{dt}H(p,q)$ and show that the energy $H(p,q)$ decreases over time!) The system (2.4) is called a *damped harmonic oscillator*. Writing (2.4) in the form (3.1) we get

$$A = \begin{pmatrix} -\frac{c}{m} & -k \\ \frac{1}{m} & 0 \end{pmatrix},$$

whose eigenvalues are

$$\lambda_- = -\frac{c}{2m} - \frac{\sqrt{c^2 - 4km}}{2m}, \qquad \lambda_+ = -\frac{c}{2m} + \frac{\sqrt{c^2 - 4km}}{2m}.$$

If $c < 2\sqrt{km}$ then the eigenvalues are complex and the solution will consist of sine waves, while if $c \geqslant 2\sqrt{km}$ then the eigenvalues are real and the solution will simply approach 0 at a speed $e^{\mathrm{Re}(\lambda_\pm t)} = e^{-ct/(2m)}$ without oscillating. The borderline case $c = 2\sqrt{km}$ gives equal eigenvalues $\lambda_\pm = -\frac{c}{2m}$, and corresponds to a so-called *critically damped* oscillator.

Figure 2.4 shows the same experiment as in Figure 2.2, but with friction coefficient $c = \frac{1}{2}$. We see that the friction acts to dampen the amplitude of the oscillations. Although this dampening effects reduces the oscillations in the explicit Euler method, it still overshoots the exact solution.

2.1 Error estimates for the forward Euler method

By approximating the exact solution of an ODE by a numerical method we will inevitably make some error. It is important to have an idea of how large this error can be, so that we can be sure that our approximation is not too far from the actual value. In this section we will estimate the error in terms of the step length $h > 0$, and we will see that the error decreases and goes to 0 as $h \to 0$.

Let us consider the forward Euler method (2.1). Even if the approximation y_n at time t_n for some reason were exactly equal to the exact value $x(t_n)$, the method would introduce a small error in the

next time step. The *local truncation error* of a method is precisely the error that the method makes from one time step to the next. More specifically, the local truncation error is the quantity

$$\tau_{n+1} := x(t_{n+1}) - \big(x(t_n) + hf(x(t_n), t_n)\big) \tag{2.5}$$

where $x(t)$ is the exact solution. More generally, if we have a numerical method of the form

$$y_{n+1} = L(y_n, t_n, h)$$

for some function L, then the truncation error is defined as $\tau_{n+1} = x(t_{n+1}) - L(x(t_n), t_n, h)$.

2.4 Lemma. *The local truncation error for the forward Euler method* (2.1) *can be bounded by*

$$|\tau_n| \leqslant \frac{K(1+M)}{2} h^2$$

for every $n = 1, \dots, N$ (where M is the constant in (1.4a)*).*

Proof. Taylor expand $x(t)$ around $t = t_n$ to get

$$x(t_{n+1}) = x(t_n) + (t_{n+1} - t_n)\frac{dx}{dt}(t_n) + \frac{(t_{n+1} - t_n)^2}{2}\frac{d^2x}{dt^2}(s)$$

for some point $s \in [t_n, t_{n+1}]$. Using the facts that $t_{n+1} - t_n = h$ and that $x(t)$ solves (1.1), we get

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + hf(x(t_n), t_n) + \frac{h^2}{2}\frac{df(x(t), t)}{dt}\Big|_{t=s} \\ &= x(t_n) + hf(x(t_n), t_n) + \frac{h^2}{2}\left(\frac{\partial f}{\partial x}(x(s), s)\frac{dx}{dt}(s) + \frac{\partial f}{\partial t}(x(s), s)\right) \\ &= x(t_n) + hf(x(t_n), t_n) + \frac{h^2}{2}\left(\frac{\partial f}{\partial x}(x(s), s) f(x(s), s) + \frac{\partial f}{\partial t}(x(s), s)\right). \end{aligned}$$

Comparing with the definition of τ_{n+1}, we see that

$$|\tau_{n+1}| = \frac{h^2}{2}\left|\frac{\partial f}{\partial x}(x(s), s) f(x(s), s) + \frac{\partial f}{\partial t}(x(s), s)\right| \leqslant \frac{h^2}{2}\big(KM + K\big). \qquad \square$$

We can use the local truncation error to estimate the *global error* of the method,

$$e_n := x(t_n) - y_n. \tag{2.6}$$

Using (2.1) and (2.5), the error at the next time step can be written as

$$\begin{aligned} e_{n+1} &= x(t_{n+1}) - y_{n+1} \\ &= \big(\tau_{n+1} + \big(x(t_n) + hf(x(t_n), t_n)\big)\big) - \big(y_n + hf(y_n, t_n)\big) \\ &= \tau_{n+1} + \big(x(t_n) - y_n\big) + h\big(f(x(t_n), t_n) - f(y_n, t_n)\big) \\ &= \tau_{n+1} + e_n + h\big(f(x(t_n), t_n) - f(y_n, t_n)\big). \end{aligned}$$

We take absolute values on both sides, apply the triangle inequality and then use the Lipschitz assumption (1.4b):

$$\begin{aligned} |e_{n+1}| &\leqslant |\tau_{n+1}| + |e_n| + hK|x(t_n) - y_n| \\ &= |\tau_{n+1}| + \big(1 + hK\big)|e_n| \end{aligned}$$

Applying this inequality to $|e_n|$, and then iteratively to $|e_{n-1}|, |e_{n-2}|, \dots, |e_1|$ gives

$$\begin{aligned}
|e_{n+1}| &\leqslant |\tau_{n+1}| + \big(1+hK\big)|e_n| \\
&\leqslant |\tau_{n+1}| + \big(1+hK\big)\big(|\tau_n| + \big(1+hK\big)|e_{n-1}|\big) \\
&\leqslant \dots \\
&\leqslant |\tau_{n+1}| + \big(1+hK\big)|\tau_n| + \dots + \big(1+hK\big)^n|\tau_1| + \big(1+hK\big)^{n+1}|e_0| \\
&= \sum_{m=0}^{n} \big(1+hK\big)^m |\tau_{n+1-m}|
\end{aligned}$$

where the last step follows from the fact that $e_0 = x(t_0) - y_0 = 0$. We can now bound the local truncation terms in the above sum using Lemma 2.4:

$$\begin{aligned}
|e_{n+1}| &\leqslant \sum_{m=0}^{n} \big(1+hK\big)^m \frac{K(1+M)}{2} h^2 = \frac{(1+hK)^{n+1}-1}{hK} \frac{K(1+M)}{2} h^2 \\
&= \frac{(1+M)\big((1+hK)^{n+1}-1\big)}{2} h
\end{aligned}$$

where we used the formula for a geometric series. Writing $n+1 = \frac{t_{n+1}}{h}$ and using the inequality $1 + a \leqslant e^a$ (which is valid for any $a \in \mathbb{R}$) gives us

$$|e_{n+1}| \leqslant \frac{(1+M)\big(e^{hKt_{n+1}/h}-1\big)}{2} h \leqslant \frac{(1+M)\big(e^{KT}-1\big)}{2} h$$

since $t_0, \dots, t_N \leqslant T$. We have thus proved that there is a constant $C > 0$ such that $|e_n| \leqslant Ch$ for every $n = 1, \dots, N$. We summarize these calculations as follows.

2.5 Lemma. *The global error for the forward Euler method* (2.1) *can be bounded by*

$$|e_n| \leqslant Ch$$

for all $n = 0, \dots, N$, where $C > 0$ is a constant that only depends on f and T.

The technique in the proof of Lemma 2.5 is called *Lady Windermere's Fan*, after a play by Oscar Wilde. The term was most likely coined by Gerhard Wanner in the 1980's.

Although the constant C in Lemma 2.5 is potentially very large, it is *independent of h*. Hence, if we would like to know the exact solution up to an error of at most $\varepsilon > 0$, then we simply choose $h \leqslant \frac{\varepsilon}{C}$. (The number ε is called the *error tolerance*.)

2.2 Convergence of the forward Euler method

We are now close to having proved that the forward Euler method converges to the exact solution as $h \to 0$, but to state this precisely we need to specify what we mean by "converges". What we have at hand is a sequence of numbers $y_0, y_1, \dots, y_N$, one for every value of $h = \frac{T}{N}$. We need to interpret these sequences as functions which in some way approximate the function $x(t)$. To this end we define the *piecewise linear interpolant*[1]

$$y(t) = y_n + \frac{t-t_n}{h}\big(y_{n+1} - y_n\big) \qquad \text{when } t \in [t_n, t_{n+1}]. \tag{2.7}$$

[1]There is no special reason to choose a *linear* interpolant—we might as well have chosen a piecewise constant or cubic interpolant. We consider the linear interpolant here because it makes some of the computations easier.

This interpolant is continuous, satisfies $y(t_n) = y_n$ for every $n = 0, 1, \ldots, N$, and in-between the time steps t_n, t_{n+1} it is linear.

Both the exact solution $x(t)$ and the numerical approximation $y(t)$ lie in $C^0([0,T], \mathbb{R}^d)$, the space consisting of continuous and bounded functions from $[0,T]$ into $\mathbb{R}^d$. In particular, their supremum norm

$$\|x\|_\infty = \sup_{t\in[0,T]} |x(t)|$$

is bounded. As is straightforward to check, this is indeed a norm on $C^0([0,T], \mathbb{R}^d)$. We will measure convergence of numerical methods in this norm: If $y^h(t)$ is a sequence of functions in $C^0([0,T], \mathbb{R}^d)$ indexed by $h > 0$, then we say that y^h converges to y, or $\lim_{h\to 0} y^h = y$, if

$$\lim_{h\to 0} \|y^h - y\|_\infty = 0.$$

2.6 Theorem. *Let $y^h(t)$ be the piecewise linear interpolant of the forward Euler method using a time step h, and let $x(t)$ be the exact solution of* (1.1)*. Then there is a constant $C > 0$, not depending on h, such that*

$$\|x - y^h\|_\infty \leqslant Ch.$$

In particular, the forward Euler method converges to the exact solution $x(t)$ as $h \to 0$.

Proof. First, note that both x and y^h are Lipschitz continuous with Lipschitz constant M. Indeed, the derivative of x is $\frac{dx}{dt} = f(x,t)$, so $|\frac{dx}{dt}| = |f(x,t)| \leqslant M$. For y^h we have for every $t \in [t_n, t_{n+1}]$ that $\frac{dy^h}{dt}(t) = \frac{y_{n+1}-y_n}{h} = f(y_n, t_n)$ (the last step following from (2.1)), so again $|\frac{dy^h}{dt}(t)| \leqslant M$.

Define the error function $e^h(t) = x(t) - y^h(t)$. Then $e^h(t_n) = e_n$ for every $n = 0, \ldots, N$, and e^h is Lipschitz continuous with Lipschitz constant at most $2M$. Hence, if, say, $t \in (t_n, t_{n+1})$ then

$$\begin{aligned} |e^h(t)| &= |e^h(t) - e^h(t_n) + e^h(t_n)| \leqslant |e^h(t) - e^h(t_n)| + |e^h(t_n)| \\ &\leqslant 2M(t - t_n) + Ch \leqslant (2M + C)h. \end{aligned}$$

The result now follows from the fact that $\|x - y^h\|_\infty = \|e^h\|_\infty = \sup_{t\in[0,T]} |e^h(t)|$. □

2.3 Convergence of the implicit Euler method

The error estimates and the proof of convergence of the implicit Euler method are quite similar to those of the explicit Euler method, although we have to be more careful since the method is implicit. As before the global error is $e_n := x(t_n) - y_n$, and the local truncation error is the error that is made after a single step in the method. Thus, the local truncation error is

$$\tau_{n+1} := x(t_{n+1}) - z_{n+1}, \qquad \text{where} \qquad z_{n+1} = x(t_n) + hf(z_{n+1}, t_{n+1}).$$

2.7 Lemma. *The local truncation error for the implicit Euler method* (2.2) *can be bounded by*

$$|\tau_{n+1}| \leqslant \frac{K(M+1)}{2(1-hK)} h^2 \tag{2.8}$$

as long as $h < \frac{1}{K}$.

Proof. As in the proof of Lemma 2.4 we Taylor expand $x(t)$, but this time around $t = t_{n+1}$:

$$\begin{aligned} x(t_n) &= x(t_{n+1}) + (t_n - t_{n+1})\frac{dx}{dt}(t_{n+1}) + \frac{(t_n - t_{n+1})^2}{2}\frac{d^2x}{dt^2}(s) \\ &= x(t_{n+1}) - hf(x(t_{n+1}), t_{n+1}) + \frac{h^2}{2}\frac{df(x(t),t)}{dt}\Big|_{t=s} \end{aligned}$$

for some $s \in [t_n, t_{n+1}]$. Using this in τ_{n+1} gives

$$\begin{aligned} |\tau_{n+1}| &= \left| x(t_n) + hf(x(t_{n+1}), t_{n+1}) - \frac{h^2}{2}\frac{df(x(t),t)}{dt}\Big|_{t=s} - x(t_n) - hf(z_{n+1}, t_{n+1}) \right| \\ &\leqslant h\big|f(x(t_{n+1}), t_{n+1}) - f(z_{n+1}, t_{n+1})\big| + \frac{h^2}{2}\left|\frac{df(x(t),t)}{dt}\Big|_{t=s}\right| \\ &\leqslant hK|x(t_{n+1}) - z_{n+1}| + \frac{h^2}{2}\big(KM + K\big) \\ &\leqslant hK|\tau_{n+1}| + \frac{h^2}{2}\big(KM + K\big). \end{aligned}$$

(Note that, unlike the proof for the forward Euler scheme, the truncation error τ_{n+1} appears in the final expression above. This happens because the scheme is implicit.) Solving for $|\tau_{n+1}|$ gives (2.8). □

Once the estimate of the local truncation error (2.8) is in place, the proof of convergence for the implicit Euler method follows in the same way as for the explicit Euler method. We state the results below and leave the proofs as exercises for the reader.

2.8 Lemma. *Assume that $h \leqslant \frac{1}{2K}$. Then the global error for the backward Euler method* (2.2) *can be bounded by*

$$|e_n| \leqslant Ch$$

for all $n = 0, \ldots, N$, where $C = 3(M+1)(e^{KT} - 1)$ is a constant that only depends on f and T.

2.9 Theorem. *Let $y^h(t)$ be the piecewise linear interpolant of the backward Euler method using a time step h, and let $x(t)$ be the exact solution of* (1.1)*. Then there is a constant $C > 0$, only depending on f and T, such that*

$$\|x - y^h\|_\infty \leqslant Ch.$$

In particular, the backward Euler method converges to the exact solution $x(t)$ as $h \to 0$.

Chapter 3

Stiff problems and linear stability

As shown in the previous chapter, both the explicit and implicit Euler methods converge to the exact solution as $h \to 0$, and the error at any time $t \in [0, T]$ can be bounded by Ch for some constant $C > 0$. Hence, to guarantee that the error is smaller than some *error tolerance* $\varepsilon > 0$, we simply have to choose $h \leqslant \frac{\varepsilon}{C}$. There is a catch, though: The constant C is proportional to e^{KT}, where K is the Lipschitz constant of f, and in many real-world applications this constant can be very large. Such problems are often characterized by a sensitivity to changes in the initial data, or the existence of features in the solution which change at very different time scales (such as two sine waves with very different frequency). Problems with some or all of these characteristics are called *stiff*.

In this chapter we will study stiff *linear* ODEs

$$\dot{x} = Ax, \qquad x(0) = x_0 \tag{3.1}$$

(for $A \in \mathbb{R}^{d\times d}$ and $x_0 \in \mathbb{R}^d$). Recall that the system (3.1) is *linearly stable* if the solution is bounded for all $t \in \mathbb{R}$, regardless of the choice of $x_0 \in \mathbb{R}^d$. Our aim will be to find out how different numerical methods perform on stiff linear problems, and under what conditions the numerical method is also linearly stable.

3.1 One-dimensional problems

To begin with we consider the linear, one-dimensional ODE

$$\dot{x} = \lambda x, \qquad x(0) = x_0 \tag{3.2}$$

for some $\lambda \in \mathbb{C}$ and $x_0 \in \mathbb{R}$. (The reason for allowing complex λ will soon be apparent.) Writing $\lambda = a + ib$ for $a, b \in \mathbb{R}$, the solution of this problem is

$$x(t) = e^{\lambda t}x_0 = e^{at}\big(\cos(bt) + i\,\sin(bt)\big)x_0.$$

Since $|x(t)| = e^{at}|x_0|$, we see that the problem (3.2) is linearly stable *if and only if* $a \leqslant 0$:

$$\lambda \in \mathbb{C}_- = \big\{z \in \mathbb{C} \,:\, \mathrm{Re}(z) \leqslant 0\big\} \tag{3.3}$$

(for otherwise e^{at} will grow towards $+\infty$ as t increases); see Figure 3.1. If $|\mathrm{Re}(\lambda)|$ is large then the corresponding solution $x(t)$ goes very quickly towards 0 or $\pm\infty$, while if $|\mathrm{Im}(\lambda)|$ is large then the solution will oscillate rapidly. We say that the ODE (3.2) is *stiff* if either of these is the case—that is, if $|\lambda|$ is large[1].

[1] Of course, "large" depends on your point of view. Here, we mean that $T|\lambda| \gg 1$. Since (3.2) states that the rate of change of the solution is proportional to $|\lambda|$, the number $T|\lambda|$ gives an indication of the net change in the solution over the time interval $[0, T]$.

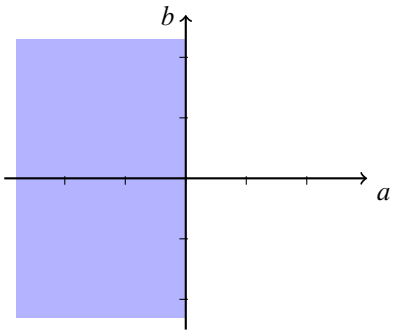
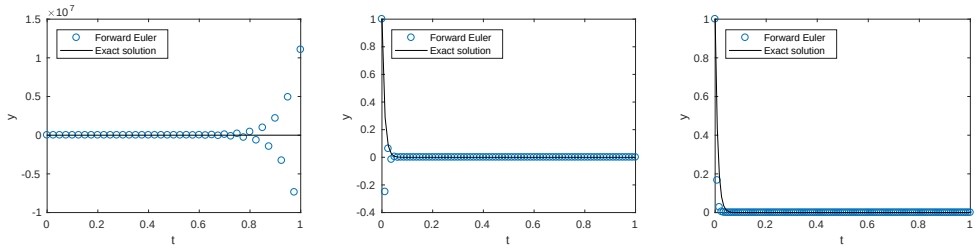


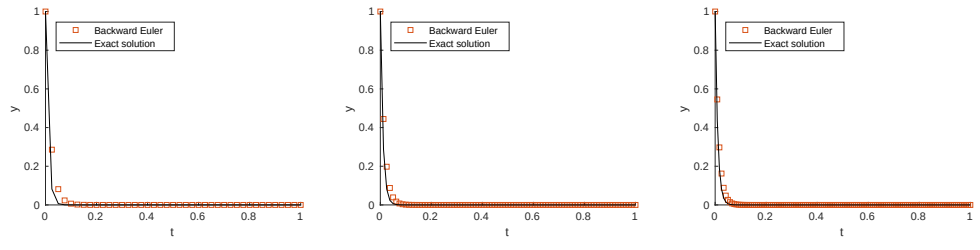
Figure 3.1: The set $\mathbb{C}_-$ of all $\lambda = a + ib$ such that (3.2) is linearly stable (in blue).

3.1 Example. Consider the stiff ODE (3.2) with $\lambda = -100$ and $x_0 = 1$, whose solution is $x(t) = e^{-100t}$. Figure 3.2(a)–(c) shows computations with the explicit Euler method for three different choices of h. On the coarsest mesh (Figure 3.2(a)) the method jumps erratically between positive and negative values and reaches values of 10^7. Increasing the resolution (Figure 3.2(b)) improves the solution, but still shows some initial oscillations. Only on the finest resolution (Figure 3.2(c)) is the numerical solution qualitatively correct.

Figure 3.2(d)–(f) shows the same computation but with the implicit Euler method. Unlike the explicit method, the implicit method shows qualitatively correct behavior on all the three resolutions. The error estimate for the implicit method, Lemma 2.8, bounds the error by $(M+1)(e^{KT}-1)h$, and since $K = 100$ and $T = 1$, the term $e^{KT} \approx 10^{43}$ is huge. Thus, the implicit method provides a reasonable, qualitatively correct solution for step sizes h *much* larger than what the error estimate seems to require.

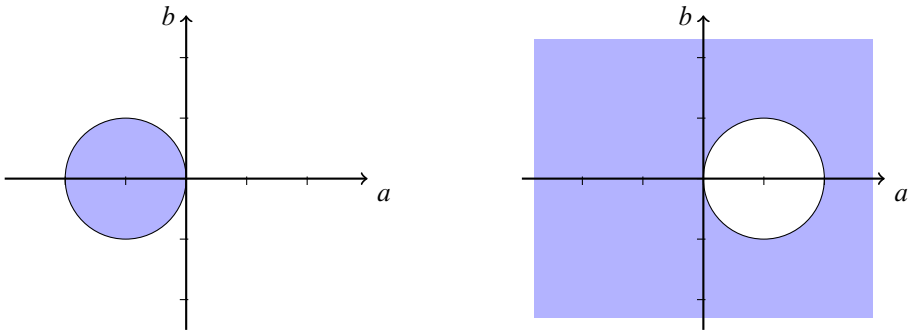


(a) $h = \frac{1}{40}$ (b) $h = \frac{1}{80}$ (c) $h = \frac{1}{120}$



(d) $h = \frac{1}{40}$ (e) $h = \frac{1}{80}$ (f) $h = \frac{1}{120}$

Figure 3.2: Forward Euler (top row) and backward Euler (bottom row) for the stiff problem in Example 3.1 up to time $T = 1$.



(a) Explicit Euler (b) Implicit Euler

Figure 3.3: Stability regions for the Euler methods (in blue).

To understand why the explicit Euler method fails and the implicit method succeeds in the above example, we write first both equations in the form

$$y_{n+1} = \varphi(\mu) y_n, \qquad y_0 = x_0, \tag{3.4}$$

where $\mu = h\lambda$, and φ is the *stability function* of the method. From Example 2.1 we see that the stability function for the explicit Euler method is given by $\varphi(\mu) = 1 + \mu$ and for the implicit Euler method it is $\varphi(\mu) = (1 - \mu)^{-1}$.

3.2 Definition. We say that the difference equation (3.4) is *linearly stable* if y_n is bounded for any initial data—that is, for every $x_0 \in \mathbb{R}$ there exists some $M > 0$ such that $|y_n| \leqslant M$ for all $n \in \mathbb{N}$.

The solution of the difference equation (3.4) is $y_n = \varphi(\mu)^n x_0$, so $|y_n| = |\varphi(\mu)|^n |x_0|$. Thus, the method is linearly stable *if and only if* $|\varphi(\mu)| \leqslant 1$.

3.3 Definition. The *region of absolute stability*, or simply the *stability region*, of the method (3.4) is the set

$$\mathcal{D} = \{\mu \in \mathbb{C} \,:\, |\varphi(\mu)| \leqslant 1\}.$$

The method (3.4) is *A-stable* if $\mathbb{C}_- \subset \mathcal{D}$.

Recalling that the ODE (3.2) is linearly stable *if and only if* $\lambda \in \mathbb{C}_-$, it is clear that a numerical method is A-stable *if and only if* the method is linearly stable for any $h > 0$ whenever $\lambda \in \mathbb{C}_-$. A method which is stable for any choice of step size $h > 0$ is often called *unconditionally stable.*

We can now answer the question of linear stability of the explicit and implicit Euler methods. Consider a linearly stable ODE (1.1), so that $\lambda \in \mathbb{C}_-$. For the explicit Euler method we have $\varphi(\mu) = 1 + \mu$, so the stability region for the explicit Euler method is the set

$$\mathcal{D}_{\text{FE}} = \{\mu \in \mathbb{C} \,:\, |\mu + 1| \leqslant 1\}.$$

For the implicit Euler method we have $\varphi(\mu) = (1 - \mu)^{-1}$, and since $|\varphi(\mu)| = |1 - \mu|^{-1}$ we get the stability region

$$\mathcal{D}_{\text{BE}} = \{\mu \in \mathbb{C} \,:\, |\mu - 1| \geqslant 1\}.$$

(The colored regions in Figure 3.3 show the stability regions for both Euler methods.) Recalling that the number μ in (3.4) is given by $\mu = h\lambda$, we see that the implicit Euler method is linearly stable

for *any* $h > 0$, while the explicit Euler method is linearly stable only for small values of h, when $|h\lambda + 1| \leqslant 1$. Hence, the implicit Euler method is A-stable, while the explicit Euler method is only *conditionally stable*—it is only stable for certain (small) values of the step size h.

If λ is a real, negative number then the explicit Euler method is linearly stable for $h\lambda \in [-2, 0]$, or $h \leqslant \frac{2}{|\lambda|}$. In Example 3.1 we had $\lambda = -100$ so the value $h = \frac{1}{40}$ in Figure 3.2(a) lies outside the region of linear stability.

3.2 Multi-dimensional problems

We consider next a general d-dimensional linear system (3.1). For the sake of simplicity we assume that A is diagonalizable and we let $\lambda_1, \ldots, \lambda_d \in \mathbb{C}$ be the eigenvalues and $v_1, \ldots, v_d \in \mathbb{C}^d$ the corresponding eigenvectors of A. The exact solution of (3.1) is then

$$x(t) = \alpha_1 e^{\lambda_1 t} v_1 + \cdots + \alpha_d e^{\lambda_d t} v_d \tag{3.5}$$

where $\alpha_1, \ldots, \alpha_d \in \mathbb{C}$ are such that $\alpha_1 v_1 + \cdots + \alpha_d v_d = x_0$. If we write $\lambda_k = a_k + i b_k$ for $k = 1, \ldots, d$ then the contribution $\alpha_k e^{\lambda_k t} v_k$ from the kth eigenpair consists of a *scaling* $e^{a_k t}$ and a *rotation* with frequency b_k. Thus, if the numbers $a_1, \ldots, a_d$ and $b_1, \ldots, b_d$ are of very different magnitude, then the formula (3.5) is a sum of terms which "live" on several different time scales. We say that the linear system (3.1) is *stiff* if the real and imaginary parts of the eigenvalues of A differ by several orders of magnitude, or if they are much larger in magnitude than $\frac{1}{T}$. Note that in the scalar case $d = 1$, this definition is equivalent to the one given in Section 3.1.

Recall that the linear system (3.1) is *spectrally stable* if $\mathrm{Re}(\lambda_k) \in \mathbb{C}_-$ for all eigenvalues $\lambda_1, \ldots, \lambda_d$, and is *linearly stable* if all solutions are bounded for all t. It can be shown that when A is diagonalizable, these two notions of stability are equivalent.

3.4 Example. Consider the damped harmonic oscillator from Example 2.3, but assume that the spring is relatively stiff[2], say, $k = 10$. From the computation in Example 2.3 we get eigenvalues $\lambda_\pm = -\frac{1}{4} \pm \sqrt{-10 + \frac{1}{16}} \approx -\frac{1}{4} \pm 3.15i$, so the solution will contain a dampening term $e^{-t/4}$ as well as sine waves with a period of approximately $\frac{2\pi}{3.15} \approx 2$. In particular, the real parts $\mathrm{Re}(\lambda_\pm) = -\frac{1}{4}$ are negative so the ODE is linearly stable.

Figure 3.4 shows the solutions computed by the explicit and implicit Euler methods using the same parameters as in Example 2.3, but with $k = 10$. While the explicit method is completely wrong, the implicit method is qualitatively correct, although it fails to capture the fine-scaled features of the solution. To obtain reasonable results with the explicit method we have to increase the resolution to $N = 400$ time steps, shown in Figure 3.5.

3.5 Example. Consider a stiff spring with a large friction coefficient, say, $k = 1000$ and $c = 1001$. If $m = 1$ as before then

$$A = \begin{pmatrix} -1001 & -1000 \\ 1 & 0 \end{pmatrix} \qquad \Rightarrow \qquad \lambda_- = -1000,\ \lambda_+ = -1,\ v_- = \begin{pmatrix} -1 \\ 1000 \end{pmatrix},\ v_+ = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

The exact solution is

$$x(t) = \alpha_- e^{-1000t} v_- + \alpha_+ e^{-t} v_+$$

where, as before, $\alpha_\pm$ are such that $x_0 = \alpha_- v_- + \alpha_+ v_+$. We see that if α_-, α_+ are both nonzero then the solution "lives" on two different time scales: the rapid time scale $1000t$ and the slow time scale

[2]This example is (probably) the origin of the term *stiff ODE*.

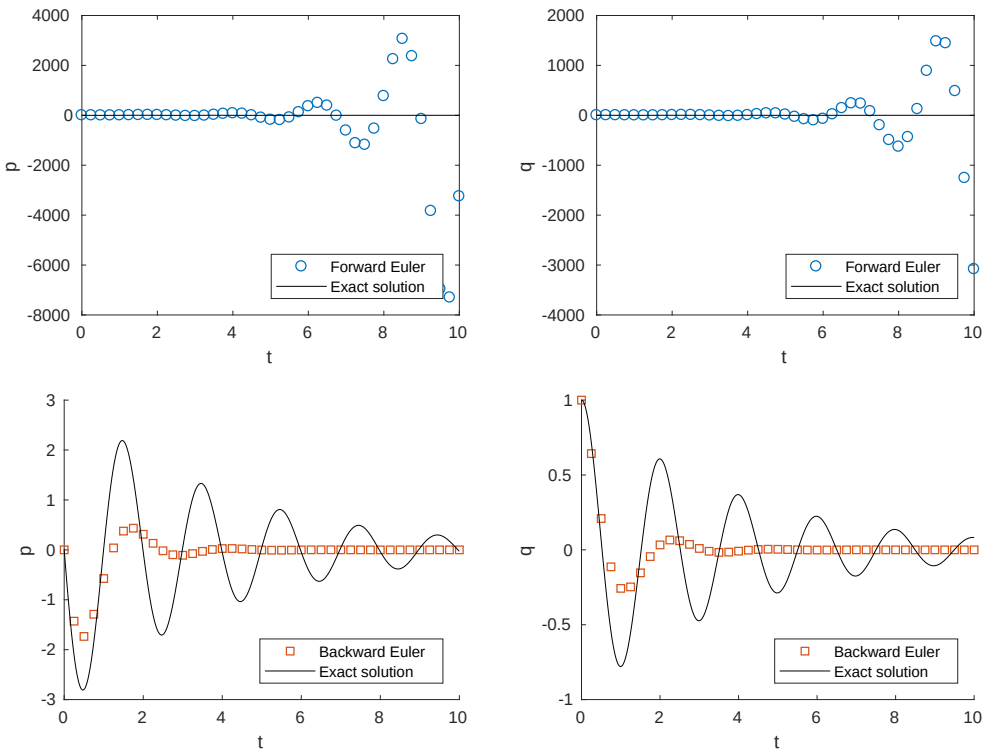


Figure 3.4: Forward (top row) and backward (bottom row) Euler computations for the damped harmonic oscillator (2.4) using $T = 10$, $N = 40$ and $k = 10$.

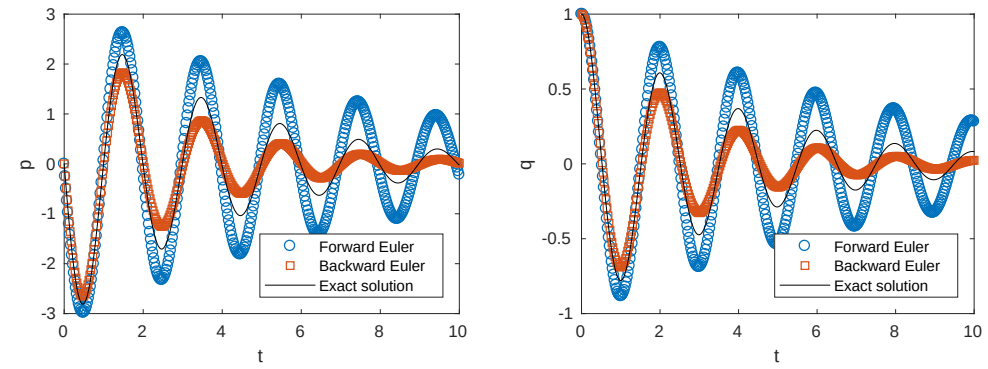


Figure 3.5: Forward and backward Euler computations for the damped harmonic oscillator (2.4) using $T = 10$, $N = 400$ and $k = 10$.

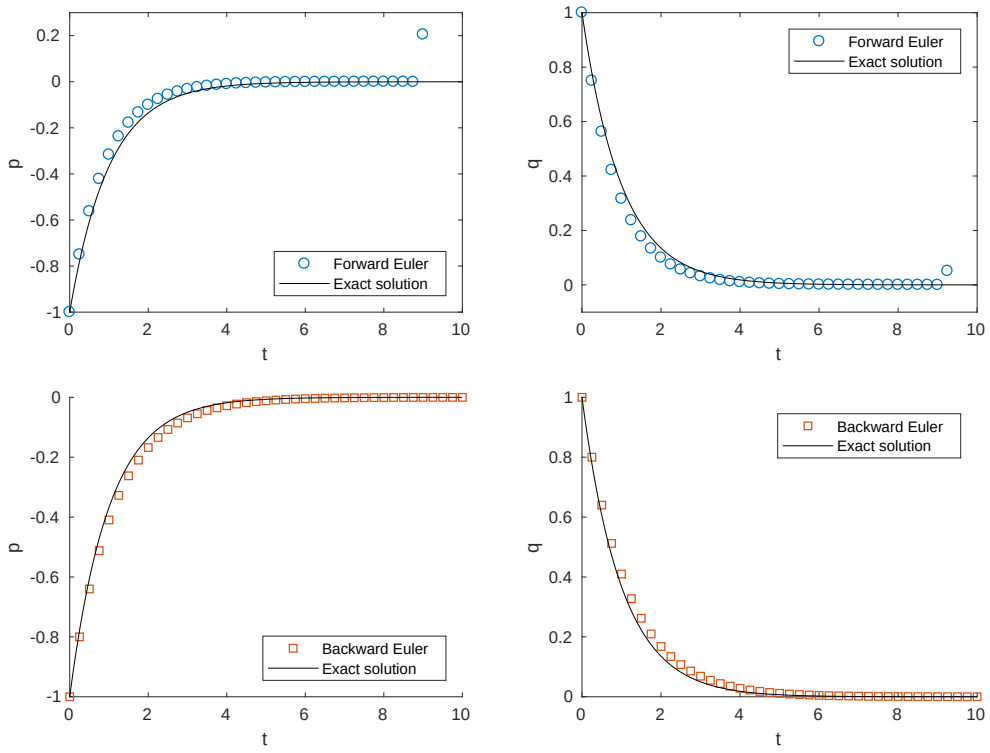


Figure 3.6: Forward (top row) and backward (bottom row) Euler computations for the damped harmonic oscillator (2.4) using $T = 10$, $N = 40$, $k = 1000$ and $c = 1001$.

t. If $\alpha_- = 0$, however, the solution is simply $x(t) = \alpha_+ e^{-t} v_+$, and it might seem that the problem is non-stiff. Figure 3.6 shows the computed solution when $x_0 = v_+$. The solution should converge to zero at the moderate pace e^{-t}, but while the implicit method computes the correct solution, the explicit method suddenly jumps at around $t \approx 9$, and quickly reaches values of around 10^8.

From the two examples above we can deduce two things. First, the presence of moderately large eigenvalues puts strict restrictions on the step size for the explicit (but not the implicit) Euler method. Second, even if x_0 is such that the solution does not "see" the large eigenvalues (that is, the solution lies entirely in the eigenspace corresponding to small eigenvalues), small round-off errors might cause instabilities in the explicit (but not the implicit) Euler method.

To study the linear stability of the explicit and implicit Euler methods we first perform a change of variables for the equation (3.1): If $u(t) = P^{-1}x(t)$ (where P is the matrix of eigenvectors) then u solves the ordinary differential equation

$$\dot{u} = \Lambda u, \qquad u(0) = P^{-1}x_0,$$

whose solution is $u(t) = e^{t\Lambda}P^{-1}x_0$. Letting y_n be the forward Euler approximation of (3.1), the sequence $w_n = P^{-1}y_n$ solves the difference equation

$$w_{n+1} = w_n + h\Lambda w_n = (I_d + h\Lambda)w_n, \qquad w_0 = P^{-1}x_0.$$

Just as for the differential equation, we have decoupled the components of y_n, yielding d scalar

difference equations

$$w_{n+1}^k = (1 + h\lambda_k) w_n^k. \tag{3.6}$$

Since y_n stays bounded *if and only if* $w_n^1, \ldots, w_n^d$ stay bounded, we see that the forward Euler method for (3.1) is linearly stable *if and only if* the forward Euler method (3.6) for each of the unknowns $u^1(t), \ldots, u^d(t)$ is linearly stable. In particular, the forward Euler method for (3.1) is linearly stable *if and only if*

$$|1 + h\lambda_k| \leqslant 1 \qquad \forall\, k = 1, \ldots, d.$$

In a similar fashion we find that the implicit Euler method is linearly stable *if and only if*

$$|1 - h\lambda_k| \geqslant 1 \qquad \forall\, k = 1, \ldots, d.$$

In particular, if (3.1) is linearly stable, that is, $\lambda_1, \ldots, \lambda_d \in \mathbb{C}_-$, then the implicit Euler method is linearly stable for *any* $h > 0$.

Chapter 4

Runge–Kutta methods

While the explicit and implicit Euler methods work just fine in a large number of applications, they only converge at a rate of $O(h)$. This is not an issue for the relatively simple problems considered in the previous chapters, but the amount of work required to get below a given error tolerance $\varepsilon > 0$ could become insurmountable if, for instance:

- the number of dimensions d is large
- the end time T is large
- the ODE is stiff
- the error tolerance ε is small
- the computational resources available are limited.

(While the last point is less of a problem today, it was a big factor when many of the methods discussed here were developed several decades ago.) This motivates developing numerical methods which converge at a speed $O(h^p)$ for $p > 1$.

Many of the ideas developed in Chapter 2 will carry over to the methods developed in this chapter. Most importantly, the principle of Lady Windermere's Fan—the conversion of a truncation error estimate to a global error estimate—still applies. Let us consider a (possibly implicit) numerical method for (1.1) of the form

$$y_{n+1} = y_n + h\Phi\big(y_n, y_{n+1}, t_n, h\big). \tag{4.1}$$

Then the *local truncation error* of the method is defined as

$$\tau_{n+1} := x(t_{n+1}) - z_{n+1}, \qquad \text{where } z_{n+1} = x(t_n) + h\Phi\big(x(t_n), z_{n+1}, t_n, h\big),$$

which can be interpreted as the error that the method makes from one time step to the next. The *global error* of the method is $e_n := x(t_n) - y_n$.

4.1 Lemma (Lady Windermere's Fan)**.** *Assume that the function Φ in* (4.1) *is Lipschitz continuous in the two first variables. Assume moreover that $|\tau_n| \leqslant C h^{p+1}$ for some $p \in \mathbb{N}$ and some $C > 0$ which is independent of h. Then $|e_n| \leqslant \tilde{C} h^p$.*

4.1 Second-order methods

As explained in Chapter 1 we can write

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} f(x(s), s)\, ds. \tag{4.2}$$

The above integral depends on the exact solution and hence must be approximated by some *quadrature rule,* that is, some function Φ such that

$$x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} f(x(s), s)\, ds \approx h\Phi\big(x(t_n), x(t_{n+1}), t_n, h\big).$$

Assume that the quadrature rule Φ is pth order accurate, so that

$$h\Phi\big(x(t_n), x(t_{n+1}), t_n, h\big) = \int_{t_n}^{t_{n+1}} f(x(s), s)\, ds + O(h^{p+1}) \tag{4.3}$$

for some $p \in \mathbb{N}$, and let us assume that Φ is Lipschitz continuous,

$$\big|\Phi\big(x_1, x_2, t, h\big) - \Phi\big(y_1, y_2, t, h\big)\big| \leqslant L_\Phi\big(|y_1 - x_1| + |y_2 - x_2|\big) \qquad \forall\ x_1, x_2, y_1, y_2 \tag{4.4}$$

for some $L_\Phi > 0$. If we use Φ in the numerical method (4.1) then the truncation error will be

$$\begin{aligned}
|\tau_{n+1}| &= \big|x(t_{n+1}) - z_{n+1}\big| \\
&= \big|x(t_{n+1}) - \big(x(t_n) + h\Phi\big(x(t_n), z_{n+1}, t_n, h\big)\big)\big| \\
&= \left|\int_{t_n}^{t_{n+1}} f(x(s), s)\, ds - h\Phi\big(x(t_n), z_{n+1}, t_n, h\big)\right| && \text{(by (4.2))} \\
&\leqslant \left|\int_{t_n}^{t_{n+1}} f(x(s), s)\, ds - h\Phi\big(x(t_n), x(t_{n+1}), t_n, h\big)\right| \\
&\quad + h\,\big|\Phi\big(x(t_n), x(t_{n+1}), t_n, h\big) - \Phi\big(x(t_n), z_{n+1}, t_n, h\big)\big| && \text{(triangle inequality)} \\
&\leqslant Ch^{p+1} + hL_\Phi\big|x(t_{n+1}) - z_{n+1}\big| && \text{(by (4.3) and (4.4))} \\
&= Ch^{p+1} + h|\tau_{n+1}|
\end{aligned}$$

Solving for τ_{n+1} gives

$$|\tau_{n+1}| \leqslant \frac{C}{1 - hL_\Phi} h^{p+1}.$$

If, say, $h \leqslant \frac{1}{2L_\Phi}$ then the term in front of h^{p+1} is less than $2C$. Hence, applying Lemma 4.1 we get a global error of $e_n = O(h^p)$.

4.1.1 The Crank–Nicholson method

The simplest quadrature rule which is more than first-order accurate is perhaps the trapezoidal rule,

$$\int_{t_n}^{t_{n+1}} g(s)\, ds = \frac{h}{2}\big(g(t_n) + g(t_{n+1})\big) + O(h^3).$$

Inserting $g(s) = f(x(s), s)$, we obtain the approximation

$$x(t_{n+1}) = x(t_n) + \frac{h}{2}\big(f(x(t_n), t_n) + f(x(t_{n+1}), t_{n+1})\big) + O(h^3). \tag{4.5}$$

The resulting numerical method can be written in the following form:

$$\begin{cases} k_1 = f(y_n, t_n) \\ k_2 = f(y_{n+1}, t_{n+1}) \\ y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2). \end{cases} \tag{4.6}$$

This is the *Crank–Nicholson* method. Note that this method is implicit, since k_2 depends on y_{n+1} and *vice versa.*

4.1.2 Heun's method

The Crank–Nicholson method (4.6) is implicit and can therefore be difficult to compute with if f is nonlinear. The troublesome term is $f(x(t_{n+1}), t_{n+1})$ in the trapezoidal rule (4.5). A common trick is to replace the exact value $x(t_{n+1})$ by an approximate value, such as the forward Euler computation

$$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n) + O(h^2).$$

Using this in the right-hand side of (4.5) gives (denoting $x_n = x(t_n)$ and $x_{n+1} = x(t_{n+1})$)

$$\begin{aligned} x_{n+1} &= x_n + \frac{h}{2}\big(f(x_n, t_n) + f(x_{n+1},\, t_{n+1})\big) + O(h^3) \\ &= x_n + \frac{h}{2}\Big(f(x_n, t_n) + f\Big(x_n + hf(x_n, t_n) + O(h^2),\, t_{n+1}\Big)\Big) + O(h^3) \\ &= x_n + \frac{h}{2}\Big(f(x_n, t_n) + f\Big(x_n + hf(x_n, t_n),\, t_{n+1}\Big) + O(h^2)\Big) + O(h^3) \\ &= x_n + \frac{h}{2}\Big(f(x_n, t_n) + f\Big(x_n + hf(x_n, t_n),\, t_{n+1}\Big)\Big) + O(h^3). \end{aligned}$$

(In the third line we have used the Lipschitz continuity of f to take the $O(h^2)$ term out of f.) The resulting second-order, explicit method can be written as

$$\begin{cases} k_1 = f(y_n, t_n) \\ k_2 = f(y_n + hk_1, t_n + h) \\ y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2). \end{cases} \tag{4.7}$$

This is *Heun's method* or the *improved Euler method.*

4.1.3 The modified Euler method

Another third-order accurate quadrature is the midpoint rule,

$$\int_{t_n}^{t_{n+1}} g(s)\, ds = hg(t_{n+1/2}) + O(h^3),$$

where $t_{n+1/2} = \frac{t_n + t_{n+1}}{2}$. Inserting $g(s) = f(x(s), s)$ gives the approximation

$$x(t_{n+1}) - x(t_n) = f(x(t_{n+1/2}), t_{n+1/2}) + O(h^3).$$

We can use the same trick as for Heun's method to approximate the midpoint value $x(t_{n+1/2})$ with the forward Euler method:

$$x(t_{n+1/2}) = x(t_n) + \frac{h}{2} f(x(t_n), t_n) + O(h^2).$$

We find that (denoting $x_n = x(t_n)$ and $x_{n+1/2} = x(t_{n+1/2})$)

$$\begin{aligned}\int_{t_n}^{t_{n+1}} f(x(s),s)\,ds &= hf\left(x_{n+1/2}, t_{n+1/2}\right) + O(h^3)\\ &= hf\left(x_n + \tfrac{h}{2} f(x_n,t_n) + O(h^2), t_{n+1/2}\right) + O(h^3)\\ &= hf\left(x_n + \tfrac{h}{2} f(x_n,t_n), t_{n+1/2}\right) + hO(h^2) + O(h^3)\\ &= hf\left(x_n + \tfrac{h}{2} f(x_n,t_n), t_{n+1/2}\right) + O(h^3).\end{aligned}$$

We can conclude that if we define

$$\Phi(y_n,t_n,h) = f\left(y_n + \tfrac{h}{2} f(y_n,t_n), t_{n+1/2}\right)$$

then the method (4.1) has a global error of $O(h^2)$. We can write out the resulting method as follows:

$$\begin{cases} k_1 = f(y_n,t_n)\\ k_2 = f\left(y_n + \frac{h}{2}k_1, t_n + \frac{h}{2}\right)\\ y_{n+1} = y_n + hk_2. \end{cases} \tag{4.8}$$

This method is often called the *modified Euler method.*

4.2 Butcher tableaus

The observant reader will have noticed that we have written all of the second-order accurate methods (4.6), (4.7), (4.8) in a similar manner. In fact, all three methods can be written in the following form:

$$\begin{cases} k_1 = f\left(y_n + h\sum_{j=1}^{2} a_{1j}k_j,\ t_n + c_1 h\right)\\ k_2 = f\left(y_n + h\sum_{j=1}^{2} a_{2j}k_j,\ t_n + c_2 h\right)\\ y_{n+1} = y_n + h\sum_{i=1}^{2} b_i k_i \end{cases} \tag{4.9}$$

for some coefficients a_{ij}, b_j and c_j. These coefficients can be neatly collected in what is called a *Butcher tableau*[1]

c_1	a_{11}	a_{12}
c_2	a_{21}	a_{22}
	b_1	b_2

The Butcher tableaus of the second-order methods in Section 4.1 are as follows:

0	0	0
1	1/2	1/2
	1/2	1/2

Crank–Nicholson

0	0	0
1	1	0
	1/2	1/2

Heun's method

0	0	0
1/2	1/2	0
	0	1

Modified Euler

[1] After the New Zealand mathematician John Charles Butcher (1933–).

These methods are examples of *two-stage Runge–Kutta methods*[2]. More generally, an s-stage Runge–Kutta method is a method of the form

$$\begin{cases} k_1 = f\left(y_n + h\sum_{j=1}^{s} a_{1j}k_j,\ t_n + c_1 h\right) \\ \quad\vdots \\ k_s = f\left(y_n + h\sum_{j=1}^{s} a_{sj}k_j,\ t_n + c_s h\right) \\ y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \end{cases} \tag{4.10}$$

where a_{ij}, b_j, c_j are coefficients that can be collected in a Butcher tableau:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

or even more compactly,

$$\begin{array}{c|c} c & A \\ \hline & b^{\top} \end{array}$$

where $A = \left(a_{ij}\right)_{i,j=1}^{s}$ is the *Runge–Kutta matrix*, $b = \left(b_i\right)_{i=1}^{s}$ is the vector of *Runge–Kutta weights* and $c = \left(c_i\right)_{i=1}^{s}$ is the vector of *Runge–Kutta nodes*.

From (4.10) we see that if $a_{ij} = 0$ for every $j \geqslant i$ then k_i only depends on preceding values $k_1, \ldots, k_{i-1}$, which implies that the method is explicit. Conversely, if $a_{ij} \neq 0$ for some $j \geqslant i$ then the expression for k_i depends either on k_i itself or some unknown, yet to be computed value k_j, and hence the method is implicit. Hence, by checking whether the diagonal and upper-diagonal entries of the Runge–Kutta matrix are zero, we quickly determine if the method is explicit or implicit. In this way we easily see that the Crank–Nicholson method is implicit, while Heun's and the modified Euler methods are explicit.

4.2 Example. The explicit and implicit Euler methods are one-stage Runge–Kutta methods with Butcher tableaus

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \qquad \text{and} \qquad \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

respectively.

4.3 Example. Runge and Kutta's original method (often simply referred to as *the Runge–Kutta method*) is the fourth-order accurate, four-stage method with Butcher tableau

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

(here a missing entry means 0).

[2] After the German mathematicians Carl David Tolmé Runge (1856–1927) and Martin Wilhelm Kutta (1867–1944).

4.3 Consistency and order of accuracy

Given a Runge–Kutta method, how do we determine its order? How do we construct a method which is, say, pth order accurate? How many stages s are required for a pth order explicit Runge–Kutta method? The first question is actually not so easy to answer; the second question requires some advanced graph theory; and the third question is an unsolved problem! In this section we will only develop some rudimentary ideas and basic consistency requirements, and we refer to more thorough treatises such as [3, Chapter 3] or [1] for further details.

The basic idea to deriving a pth order Runge–Kutta method is the same as for the second-order methods in Section 4.1. Begin by applying a quadrature rule (such as Gauss quadrature) to approximate

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} f\big(x(s), s\big)\, ds \\ &\approx x(t_n) + h \sum_{i=1}^{s} b_i f\Big(x\big(t_n + hc_i\big), t_n + hc_i\Big) \end{aligned}$$

where $t_n + hc_i \in [t_n, t_{n+1}]$ are the quadrature points and b_i the quadrature weights. A basic requirement of the quadrature weights is that

$$\sum_{i=1}^{s} b_i = 1. \tag{4.11}$$

This guarantees that the quadrature rule is exact whenever f is constant, which is a necessary requirement for the method to be at least first-order accurate.

Let y_n and y_{n+1} be approximations of $x(t_n)$ and $x(t_{n+1})$, respectively, and let $y_n^{(i)}$ approximate $x\big(t_n + hc_i\big)$. Then

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i f\big(y_n^{(i)}, t_n + hc_i\big) \tag{4.12a}$$

gives an approximation of $x(t_{n+1})$. Using the fact that

$$x\big(t_n + hc_i\big) = x(t_n) + \int_{t_n}^{t_n + hc_i} f\big(x(s), s\big)\, ds$$

we can express $y_n^{(i)}$ by approximating the above integral using another quadrature rule. Since using quadrature points other than $t_n + hc_1, \ldots, t_n + hc_s$ would introduce new, unknown values of x, we stick to these quadrature points:

$$y_n^{(i)} = y_n + hc_i \sum_{j=1}^{s} \tilde{a}_{ij} f\big(y_n^{(j)}, t_n + hc_j\big) \tag{4.12b}$$

(the factor hc_i is the length of the integration domain), where $\tilde{a}_{i1}, \ldots, \tilde{a}_{is}$ are the quadrature weights to compute $y_n^{(i)}$. Just as in (4.11), a basic consistency requirement for the quadrature weights $\tilde{a}_{ij}$ is $\sum_{j=1}^{s} \tilde{a}_{ij} = 1$. Defining $a_{ij} = c_i \tilde{a}_{ij}$, this requirement translates to

$$\sum_{j=1}^{s} a_{ij} = c_i \qquad \text{for every } i = 1, \ldots, s. \tag{4.13}$$

Putting together (4.12a) and (4.12b) gives exactly the Runge–Kutta method (4.10). Note that all of the Butcher tableaus given in Section 4.2 satisfy the consistency requirements (4.11) and (4.13).

So what is the order of accuracy of a given Runge–Kutta method? Unfortunately there is no straightforward method to answer this question. One approach consist of Taylor expanding f and x around each quadrature point, but this quickly becomes too messy for any order p greater than 2. For a systematic (but quite technical) method of using graphs to express Taylor expansions, see [1].

For explicit Runge–Kutta methods it is known that one needs at least $s = p$ stages to obtain pth order accuracy. In fact, if $p \geqslant 5$ then one needs $s \geqslant p + 1$ stages, and this trend continues for larger values of p: The number of stages s grows faster than p. For implicit Runge–Kutta methods, on the other hand, it is known that the order of accuracy of an s-stage method is at most $p = 2s$, and for every $s \in \mathbb{N}$ there is in fact a unique $2s$-order accurate, s-stage Runge–Kutta method. (This is analogous to the uniqueness of the s-point, $(2s + 1)$th order accurate Gauss quadrature rule.) Thus, there is a tradeoff here: An implicit, s-stage method is potentially much more accurate than an explicit s-stage method, but on the other hand, implicit methods require solving implicit equations and are hence much more computationally demanding.

4.4 Linear stability of Runge–Kutta methods

In this section we investigate the linear stability of Runge–Kutta methods. We will see that for these methods, the general rule of thumb of stability still holds: implicit methods are stable regardless of the timestep h, while explicit methods are only stable for sufficiently small h.

Recall from Chapter 3 that a numerical method is *linearly stable* if the computed solution stays bounded for all times when solving the linear ODE

$$\dot{x} = \lambda x$$

for some $\lambda \in \mathbb{C}$. We insert the choice $f(x) = \lambda x$ into the general form of a Runge–Kutta method (4.10) to get

$$\begin{cases} k_1 = \lambda y_n + \lambda h \sum_{j=1}^{s} a_{1j} k_j \\ \quad \vdots \\ k_s = \lambda y_n + \lambda h \sum_{j=1}^{s} a_{sj} k_j \\ y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i. \end{cases} \tag{4.14}$$

We can write the above in matrix notation:

$$k = \mathbb{1}\lambda y_n + \lambda h A k, \qquad y_{n+1} = y_n + h b^\top k$$

where $\mathbb{1}$ and k are the vectors $\mathbb{1} = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}^\top$ and $k = \begin{pmatrix} k_1 & k_2 & \cdots & k_s \end{pmatrix}^\top$, and b is the vector of Runge–Kutta weights. Solving the above for k gives $k = (I - \mu A)^{-1} \mathbb{1} \lambda y_n$, where we denote $\mu = \lambda h$, as before. This gives $y_{n+1} = y_n \varphi(\mu)$, where $\varphi(\mu) = 1 + \mu b^\top (I - \mu A)^{-1} \mathbb{1}$ is the stability function for the Runge–Kutta method (4.14) (see Section 3.1). We can simplify this expression even further:

4.4 Lemma. *The stability function for the s-step Runge–Kutta method* (4.14) *is*

$$\varphi(\mu) = \frac{\det\left(I - \mu A + \mu \mathbb{1} b^\top\right)}{\det\left(I - \mu A\right)}. \tag{4.15}$$

The proof of the above lemma is outside the scope of these notes, but the idea is to use Cramer's rule to invert the matrix $I - \mu A$ (see [2, Section IV.3]). As an exercise, try to use the formula (4.15)

to compute the stability functions for the explicit and implicit Euler methods (see Example 4.2) and see if you get the same results as in Chapter 3.

Let us first assume that the method in question is explicit, so that $a_{ij} = 0$ for all $j \geqslant i$. Then $I - \mu A$ is a lower-triangular matrix with ones along the diagonal, so $\det\big(I - \mu A\big) = 1$. The numerator in (4.15) is the determinant of an $s \times s$ matrix, so it follows that the stability function for an explicit Runge–Kutta method is an sth order polynomial,

$$\varphi(\mu) = 1 + d_1\mu + d_2\mu^2 + \cdots + d_s\mu^s$$

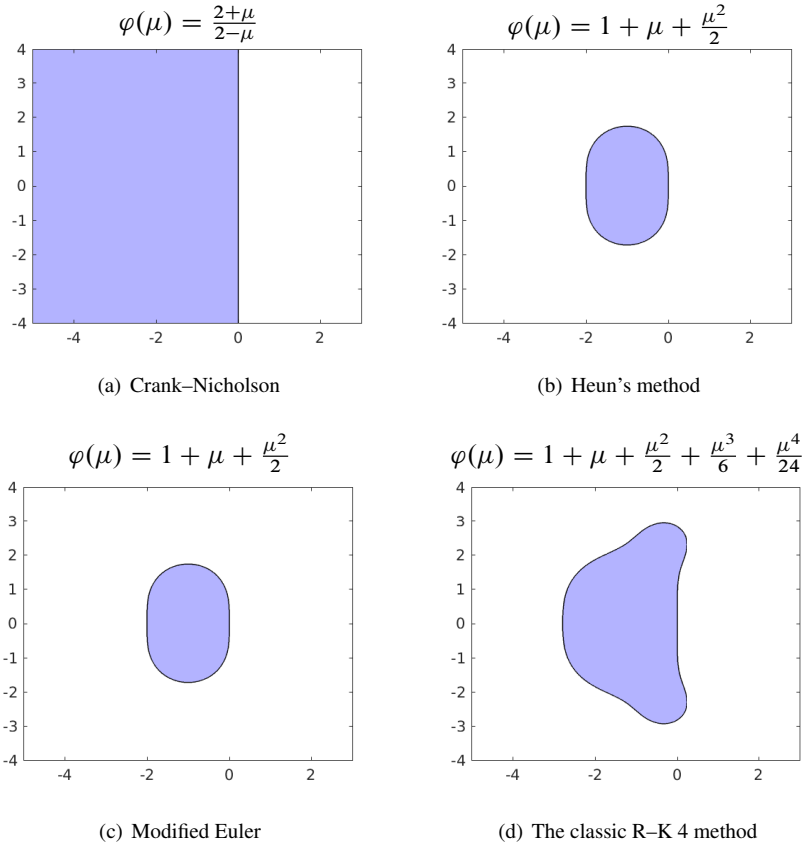
for some coefficients $d_1, \ldots, d_s \in \mathbb{R}$. The classical result of Liouville (see any textbook on complex analysis) states that every non-constant polynomial is unbounded. Hence, the stability region

$$\mathcal{D} = \big\{\mu \in \mathbb{C} \,:\, |\varphi(\mu)| \leqslant 1\big\}$$

of any explicit Runge–Kutta method is a bounded set, and in particular, the method is only stable for certain values of h.

If the Runge–Kutta method is in implicit then it is generally *not* the case that $\det\big(I - \mu A\big)$ equals 1; instead it will be some sth order polynomial of μ. Consequently, the stability function (4.15) is a rational polynomial (that is, one polynomial divided by another) which in many cases is bounded for any value of μ.

Figure 4.1 shows the stability functions and stability regions for the four different Runge–Kutta methods encountered in this chapter. We see that the stability functions of the explicit methods (Figures (b)–(d)) are polynomials, and that their stability regions are bounded sets. Hence, if the problem is stiff—that is, if $|\lambda| \gg 1$—then the step size h must be very small in order to get a stable method. On the other hand, the stability function of the implicit Crank–Nicholson method (Figure (a)) is a rational polynomial, and the stability region is $\mathcal{D} = \mathbb{C}_-$. It follows that the Crank–Nicholson method is A-stable.



(a) Crank–Nicholson

(b) Heun's method

(c) Modified Euler

(d) The classic R–K 4 method

Figure 4.1: Stability functions and stability regions (in blue) for different Runge–Kutta methods.

Chapter 5

Symplectic methods

Recall that a Hamiltonian system is an ordinary differential equation which can be written in the form

$$\begin{cases} \dot{p} = -\frac{\partial H}{\partial q}(p,q) \\ \dot{q} = \frac{\partial H}{\partial p}(p,q) \end{cases} \tag{5.1}$$

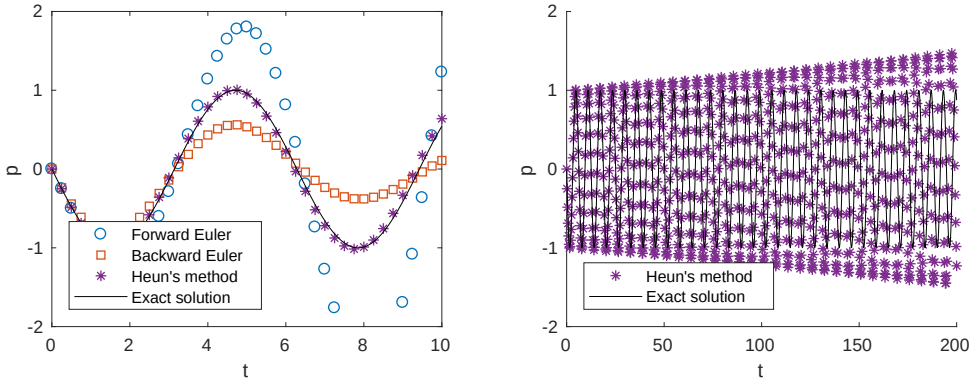
for some function $H = H(p,q)$. For simplicity we will assume in this chapter that the unknown functions p, q are scalar-valued, but emphasize that everything in this chapter can be generalized to vector-valued unknowns.

We have already seen one example of a Hamiltonian system in Example 2.2, the harmonic oscillator. Other examples of Hamiltonian systems include the equations of motion of a planetary system or of particles in a particle accelerator. In many applications it is important to be able to solve these systems very accurately for a long time. For instance, an astrophysicist could ask whether an asteroid in our solar system will collide with Earth sometime during the next fifty years; any small error in the computation might give a false negative. Or a particle physicist at CERN could ask whether a beam of particles will make it through the Large Hadron Collider particle accelerator over the next few seconds; the particles pass through each of the 1232 magnets in the the 27 km long tunnel more than 11.000 times per second, so any inaccuracies will quickly multiply and give an incorrect answer.

An easy solution would be to use a high-order accurate numerical method, such as a Runge–Kutta method. But very high-dimensional or long-time problems might still be too computationally expensive for the desired accuracy, even with a high-order accurate method.

5.1 Example. As we saw in Example 2.2, the explicit and implicit Euler methods were quite inaccurate when simulating the harmonic oscillator (cf. Figure 2.2). Figure 5.1(a) shows the same problem computed with Heun's method. It's clear that the second-order method gives a *much* better approximation. But what if we want to compute for a longer time, say, until $T = 200$? As shown in Figure 5.1(b), the method slowly, but surely starts to overestimate the amplitude of the oscillator, and at the end of the simulation the error is of the order of 1.

By simply using a higher-order accurate method we are treating the ODE (5.1) just like any other ODE. As we will see in this chapter, we will gain much more in accuracy if we take advantage of the very special Hamiltonian structure (5.1).



(a) $T = 10$ (b) $T = 200$

Figure 5.1: The p component in the harmonic oscillator (2.3), computed with Heun's method.

5.1 Symplectic maps

Let $x = (p\ q)^\top$ be the vector of unknowns and let φ be the flow of the ODE (5.1), so that $\varphi_t(x)$ solves (5.1) with initial data $x \in \mathbb{R}^2$. In this section we will see that, in addition to preserving the Hamiltonian $H(\varphi_t(x))$ over time, the flow φ has the property of being *area preserving*.

If $x_1 = (p_1\ q_1)^\top$ and $x_2 = (p_2\ q_2)^\top$ are given vectors then the area of the parallelogram spanned by x_1 and x_2 is given by

$$\omega(x_1, x_2) = p_1 q_2 - p_2 q_1 = x_1^\top J x_2, \qquad \text{where } J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

Let now $g : \mathbb{R}^2 \to \mathbb{R}^2$ be some given function, and assume for the moment that g is linear, so that $g(x) = Ax$ for some $A \in \mathbb{R}^{2\times 2}$. We say that g (or A) is *symplectic* if it preserves the area of all such paralellograms, that is, if

$$\omega\big(g(x_1), g(x_2)\big) = \omega(x_1, x_2) \qquad \forall\ x_1, x_2 \in \mathbb{R}.$$

Inserting $g(x) = Ax$, it is not hard to see that g is symplectic *if and only if*

$$A^\top J A = J. \tag{5.2}$$

5.2 Exercise. Let

$$A = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 0 \\ 0 & 1/2 \end{pmatrix}, \qquad C = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

for some $\theta \in [0, 2\pi)$. Show that A and B are symplectic, and that C is not.

Let now $g : \mathbb{R}^2 \to \mathbb{R}^2$ be an arbitrary, possibly nonlinear function. Noting that the matrix A above is the Jacobian of g, we say that g is *symplectic* if the identity (5.2) is true for the matrix $A = \nabla g(x)$, for every $x \in \mathbb{R}^2$. Although we will not go into details here, it is possible to use tools from differential geometry to show that g being symplectic is the same as saying that g preserves

area—that is, if $\Omega \subset \mathbb{R}^2$ is some bounded set, then Ω and $g(\Omega) = \{g(x) : x \in \Omega\}$ have the same area.

We now investigate the flow $\varphi_t(x)$ of the Hamiltonian system (5.1). First, observe that we can write the system (5.1) as

$$\dot{x} = -J\nabla H(x) \tag{5.3}$$

where $\nabla H(p,q) = \left(\frac{\partial H}{\partial p}(p,q) \;\; \frac{\partial H}{\partial q}(p,q)\right)^\top$.

5.3 Theorem (Poincaré, 1899)**.** *Let φ be the flow of the Hamiltonian system* (5.1)*. Then for every $t \in \mathbb{R}$, the function $\varphi_t(x)$ is symplectic.*

Proof. We need to check that for every fixed time $t \in \mathbb{R}$ and $x \in \mathbb{R}^2$, the matrix $A = \nabla\varphi_t(x)$ satisfies (5.2). If $t = 0$ then $\varphi_t(x) = x$, so $A = I$, which clearly satisfies (5.2). For a general $t \in \mathbb{R}$ we see from (5.3) that

$$\frac{d}{dt}\nabla\varphi_t(x) = \nabla\left(\tfrac{d}{dt}\varphi_t(x)\right) = -\nabla\left(J\nabla H(\varphi_t(x))\right) = -J\nabla^2 H(\varphi_t(x))\nabla\varphi_t(x).$$

(Here, $\nabla^2 H$ is the Hessian matrix of H, consisting of all second-order partial derivatives of H.) Hence, if $A = \nabla\varphi_t(x)$ then

$$\begin{aligned}\frac{d}{dt}\left(A^\top JA\right) &= \frac{d}{dt}\left(\nabla\varphi_t(x)^\top J\nabla\varphi_t(x)\right) = \left(\tfrac{d}{dt}\nabla\varphi_t(x)\right)^\top J\nabla\varphi_t(x) + \nabla\varphi_t(x)^\top J\left(\tfrac{d}{dt}\nabla\varphi_t(x)\right)\\ &= -A^\top\nabla^2 H(\varphi_t(x))^\top J^\top JA - A^\top JJ\nabla^2 H(\varphi_t(x))A\\ &= 0\end{aligned}$$

because $J^{-1} = J^\top = -J$. Since $A^\top JA = J$ when $t = 0$, we conclude that $A^\top JA = J$ for every $t \in \mathbb{R}$. □

Perhaps even more surprising is the fact that Poincaré's theorem holds in the converse direction:

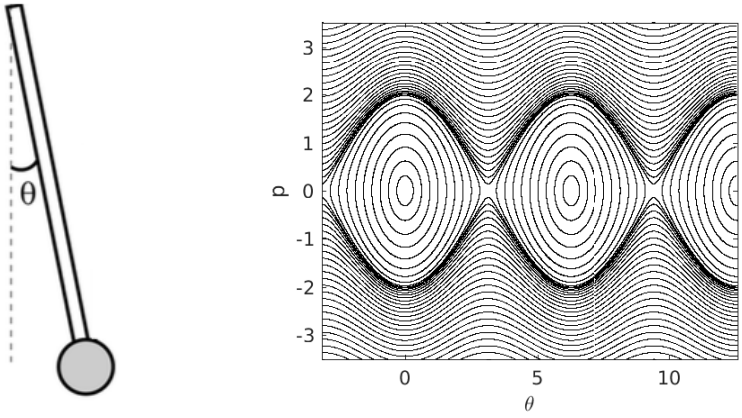
5.4 Theorem. *Let $\varphi_t : \mathbb{R}^2 \to \mathbb{R}^2$ be the flow of some ODE and assume that for each $t \in \mathbb{R}$, the map $\varphi_t(x)$ is symplectic. Then for every point $x_0 \in \mathbb{R}^2$ there is a neighborhood U of x_0 and a function $\tilde{H}(x)$ such that for $x \in U$, $\varphi_t(x)$ is the flow of the ODE with Hamiltonian $\tilde{H}$.*

Thus, in a certain sense, the flow map φ is symplectic *if and only if* the corresponding ODE is Hamiltonian. The proof of the above theorem is outside the scope of these notes.

5.5 Example. Consider a pendulum with mass m. We can describe its position by the angle θ that the pendulum makes with the downward vertical, see Figure 5.2(a). Using Newton's second law, one can show that the angle behaves according to the ODE

$$m\ddot{\theta} = -g\sin\theta,$$

where $g > 0$ is the gravitational constant. Letting $p = m\dot{\theta}$, we can write the above ODE as a Hamiltonian system with unknown (p,θ) and Hamiltonian function $H(p,\theta) = \frac{1}{2m}p^2 - g\cos(\theta)$. A phase portrait of this Hamiltonian system can be seen in Figure 5.2(b). Figure 5.3 shows the evolution of the flow over time, along with a superimposed image where each pixel follows the flow. Although the image is severely distorted over time, its area (and the area of any section of the image) is preserved.



(a) The pendulum (b) Phase portrait

Figure 5.2: The pendulum in Example 5.5.

5.2 Symplectic numerical methods

Let us consider some numerical method for the Hamiltonian system (5.1), and assume that we are able to write down the method explicitly,

$$y_{n+1} = \Phi_h(y_n) \tag{5.4}$$

where $\Phi_h : \mathbb{R}^2 \to \mathbb{R}^2$ is a "discrete flow"—a function which returns the numerical approximation at time $t = h$. We say that a numerical method is *symplectic* (also called a *symplectic integrator*) if it can be written as (5.4) for some function Φ_h which is symplectic for all (sufficiently small) step sizes h. For instance, the exact flow $\Phi_h = \varphi_h$ is a symplectic integrator. As we saw in the previous section, a flow Φ_h is symplectic *if and only if* its corresponding flow is Hamiltonian. The Hamiltonian function for Φ_h, which we denote by $\tilde{H}$, is not necessarily the same as H, but it is usually very close. Thus, the energy of the computed solution is

$$H(y_n) \approx \tilde{H}(y_n) = \tilde{H}(y_{n-1}) = \cdots = \tilde{H}(y_0) \approx H(y_0),$$

and therefore we can expect a symplectic method to *almost* preserve the energy $H(y_n)$ over time.

5.6 Proposition. *The* symplectic Euler method

$$\begin{cases} p_{n+1} = p_n - h\frac{\partial H}{\partial q}(p_{n+1}, q_n) \\ q_{n+1} = q_n + h\frac{\partial H}{\partial p}(p_{n+1}, q_n) \end{cases} \tag{5.5}$$

is a first-order accurate symplectic method.

Proof. Let us for the sake of simplicity assume that H is separable, meaning that it can be written as

$$H(p, q) = T(p) + V(q), \tag{5.6}$$

where we can interpret $V(q)$ as the potential energy and $T(p)$ as the kinetic energy of the solution. Then the symplectic Euler method is in fact explicit:

$$\begin{cases} p_{n+1} = p_n - hV'(q_n) \\ q_{n+1} = q_n + hT'(p_{n+1}) = q_n + hT'\big(p_n - hV'(q_n)\big). \end{cases}$$

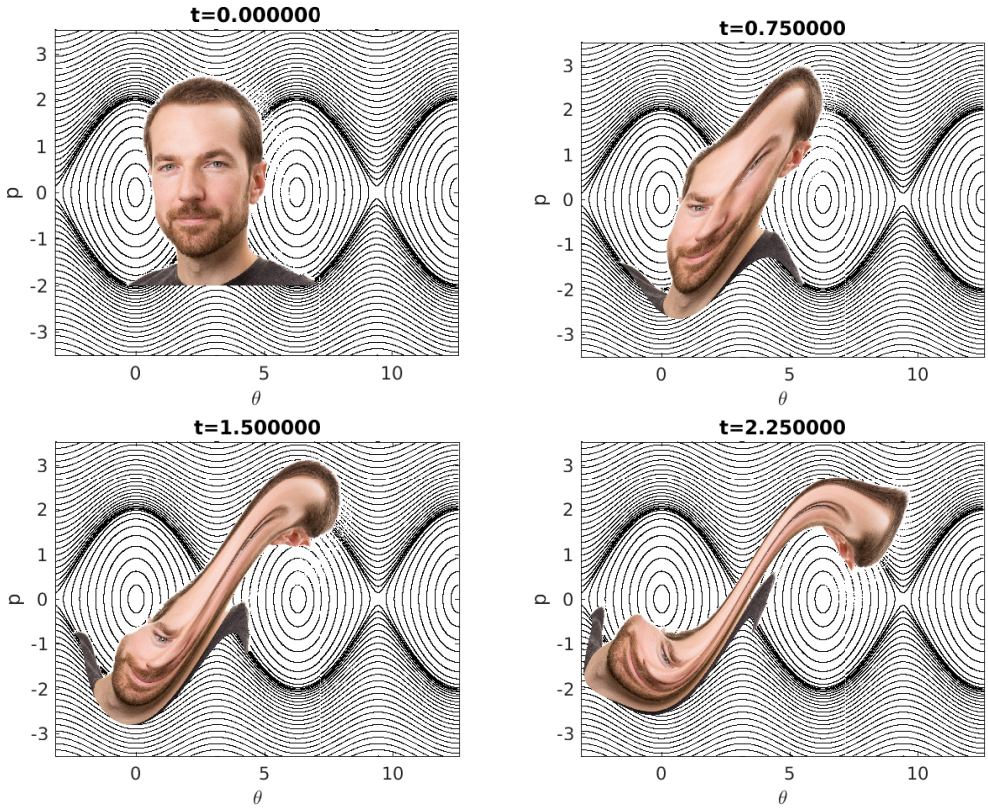


Figure 5.3: The author is transported with the flow φ_t, but his area is preserved over time. The black curves are orbits of the flow.

Letting $\Phi_h(p_n, q_n)$ denote the right-hand side, we can compute

$$A := \nabla\Phi_h(p_n, q_n) = \begin{pmatrix} 1 & -hV'' \\ hT'' & 1 - h^2T''V'' \end{pmatrix}$$

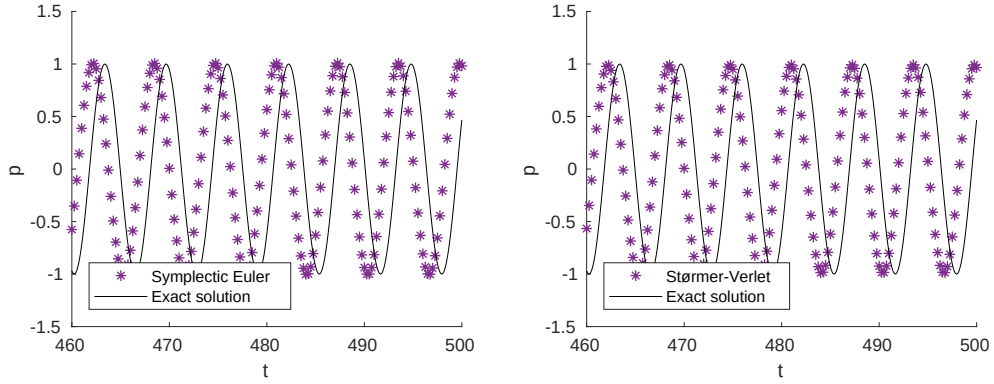
where $V'' = V''(q_n)$ and $T'' = T''\big(p_n - hV'(q_n)\big)$. It is now straightforward to check that A satisfies (5.2), regardless of p_n and q_n. □

5.7 Proposition. *The Størmer–Verlet method*[1]

$$\begin{cases} p_{n+1/2} = p_n - \frac{h}{2}\frac{\partial H}{\partial q}(p_{n+1/2}, q_n) \\ q_{n+1} = q_n + \frac{h}{2}\Big(\frac{\partial H}{\partial p}(p_{n+1/2}, q_n) + \frac{\partial H}{\partial p}(p_{n+1/2}, q_{n+1})\Big) \\ p_{n+1} = p_{n+1/2} - \frac{h}{2}\frac{\partial H}{\partial q}(p_{n+1/2}, q_{n+1}) \end{cases} \tag{5.7}$$

is a second-order accurate symplectic method.

[1]Named after Carl Størmer (1874–1957) and Loup Verlet (1931–), often (incorrectly) called the Störmer–Verlet method. A professor in mathematics at the University of Oslo, Størmer did groundbreaking work in the modeling of aurora borealis using differential equations. Størmer is also known for his photographs of people in Karl Johans gate, Oslo using a camera hidden under his jacket, in particular his photograph of Henrik Ibsen.



(a) Symplectic Euler (b) Størmer–Verlet

Figure 5.4: The p component in the harmonic oscillator (2.3). Note the values along the t-axis.

Proof. Let us again for the sake of simplicity assume that the Hamiltonian can be written in the separable form (5.6). Then the Størmer–Verlet method is also explicit:

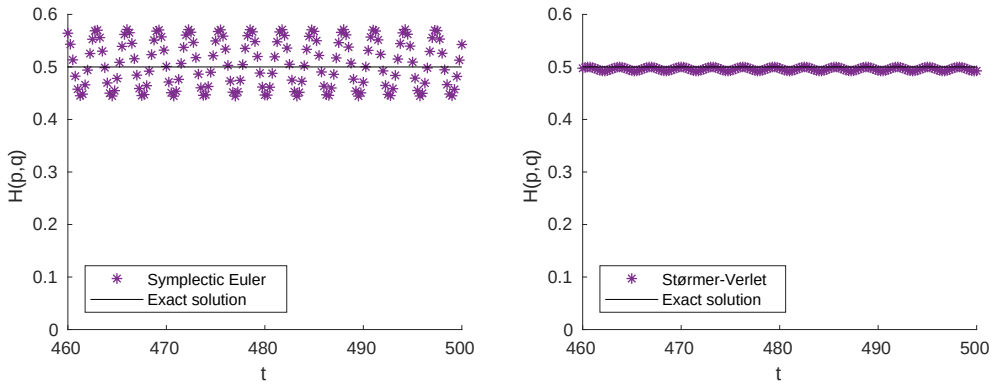
$$\begin{cases} p_{n+1/2} = p_n - \frac{h}{2} V'(q_n) \\ q_{n+1} = q_n + hT'(p_{n+1/2}) \\ p_{n+1} = p_{n+1/2} - \frac{h}{2} V'(q_{n+1}). \end{cases}$$

Writing the method in the form (5.4), we get

$$A := \nabla \Phi_h(p_n, q_n) = \begin{pmatrix} 1 & -\frac{h}{2} V'' \\ hT'' & 1 - \frac{h^2}{2} T'' V'' \end{pmatrix}$$

where $V'' = V''(q_n)$ and $T'' = T''\big(p_n - \frac{h}{2} V'(q_n)\big)$. Again, it is an easy exercise to check that A satisfies (5.2). □

5.8 Example. We repeat the example with the harmonic oscillator, Example 5.1. As shown in Figure 5.4, the first-order accurate symplectic Euler method and the second-order accurate Størmer–Verlet method both compute the solution quite accurately up to time $T = 500$, with only a small error in the phase of the oscillations. The total energy of the solution, shown in Figure 5.5, oscillates around the correct value $H \equiv 0.5$ and, most importantly, the error in the does not increase over time.



(a) Symplectic Euler

(b) Størmer–Verlet

Figure 5.5: The total energy $H(p,q)$ for the harmonic oscillator (2.3). Note the values along the t-axis.

Bibliography

[1] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 3rd edition, 2016.

[2] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II*. Springer-Verlag Berlin Heidelberg, 2nd edition, 1996.

[3] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2nd edition, 2008.