# Chapter 9

# Initial value problems

Differential equations are fundamental to model the laws of Nature, such as Newton's laws of motion, Einstein's general relativity, Schrödinger's equation of quantuum mechanics, and Maxwell's equations of electromagnetics.

The case of one single independent variable, we refer to as ordinary differential equations, whereas partial differential equations involve several independent variables. We first consider the initial value problem, an ordinary differential equation where the independent variable naturally represents time, for which we develop solution methods based on time-stepping algorithms.

## 9.1   The scalar initial value problem

We consider the following *ordinary differential equation* (ODE) for a scalar function $u : \mathbb{R}^+ \to \mathbb{R}$, with derivative $\dot{u} = du/dt$,

$$\begin{aligned} \dot{u}(t) &= f(u(t), t), \quad 0 < t \leq T, \\ u(0) &= u_0, \end{aligned} \tag{9.1}$$

which we refer to as a *scalar initial value problem*, defined on the interval $I = [0, T]$ by the function $f : \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$, and the *initial condition* $u(0) = u_0$.

Only in simple special cases can analytical solutions be found. Instead, in the general case, numerical methods must be used to compute approximate solutions to (9.1).

## 9.2   Time stepping methods

The variable $t \in [0, T]$ is often interpreted as time, and numerical methods to solve (9.1) can be based on the idea of *time-stepping*, where successive approximations $U(t_n)$ are computed on a partition $0 = t_0 < t_1 < \ldots < t_N = T$, starting from $U(t_0) = u_0$, with a suitable *interpolation* of $U(t)$ on each subinterval $I_n = (t_{n-1}, t_n)$ of length $k_n = t_n - t_{n-1}$.

$$I_n = (t_{n-1}, t_n)$$
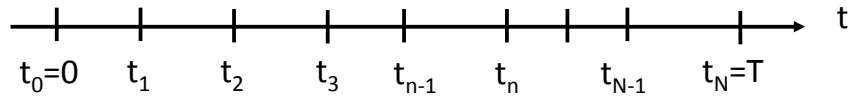
$$k_n = t_n - t_{n-1}$$



Figure 9.1: Partition of the interval $I = [0, T]$, $0 = t_0 < \ldots < t_N = T$.

### Forward Euler method

For node $t_n$, we may approximate the derivative $\dot{u}(t_n)$ by

$$\dot{u}(t_{n-1}) \approx \frac{u(t_n) - u(t_{n-1})}{k_n}, \tag{9.2}$$

so that

$$u(t_n) \approx u(t_{n-1}) + k_n \dot{u}(t_{n-1}) = u(t_{n-1}) + k_n f(u(t_{n-1}), t_{n-1}), \tag{9.3}$$

which motivates the *forward Euler method* for successive computational approximation of $U_n = U(t_n)$.

---

**Algorithm 14:** Forward Euler method

---

$U_0 = u_0$                             ▷ `initial approximation`
**for** $n = 1, 2, ..., N$ **do**
   |   $U_n = U_{n-1} + k_n f(U_{n-1}, t_{n-1})$        ▷ `explicit update`
**end**

---

We note that the forward Euler method is *explicit*, meaning that $U_{n+1}$ is directly computable from the previous solution $U_n$ in the time-stepping algorithm. The method is thus also referred to as the *explicit Euler method.*

## Backward Euler method

Alternatively, we may approximate the derivative $\dot{u}(t_{n+1})$ by

$$\dot{u}(t_n) \approx \frac{u(t_n) - u(t_{n-1})}{k_n}, \tag{9.4}$$

so that

$$u(t_n) \approx u(t_{n-1}) + k_n \dot{u}(t_n) = u(t_{n-1}) + k_n f(u(t_n), t_n), \tag{9.5}$$

which motivates the *backward Euler method* for successive computational approximation of $U_n = U(t_n)$.

---

**Algorithm 15:** Backward Euler method

---

$U_0 = u_0$                             ▷ `initial approximation`
**for** $n = 1, 2, ..., N$ **do**
   |   $U_n = U_{n-1} + k_n f(U_n, t_n)$        ▷ `solve algebraic equation`
**end**

---

Contrary to the forward Euler method, the backward Euler method is *implicit*, thus also referred to as the *implicit Euler method*, meaning that $U_{n+1}$ is not directly computable from $U_n$, but is obtained from the solution of an algebraic (possibly nonlinear) equation,

$$x = U_{n-1} + k_n f(x, t_n), \tag{9.6}$$

for example, by the fixed point iteration,

$$x^{(k+1)} = U_{n-1} + k_n f(x^{(k)}, t_n). \tag{9.7}$$

We note that the fixed point iteration (9.7) converges if $k_n L_f < 1$, with $L_f$ the Lipschitz constant of the function $f(\cdot, t_n)$, thus if the time step $k_n$ is small enough.

## Time stepping as quadrature

There is a strong connection between time-stepping methods and numerical approximation of integrals, referred to as *quadrature*. For example, assume that the initial condition is zero and that the function $f$ in (9.1) does not depend on the solution $u$, but the time $t$ only, that is $f = f(t)$. The solution $u(t)$ is then the primitive function of $f(t)$ that satisfies $u(0) = 0$, corresponding to the area under the graph of the function $f(t)$ over the interval $[0, t]$.

We can approximate this primitive function by left and right *rectangular rule* quadrature, or *Riemann sums*, which we illustrate in Figure 9.2. The two approximations of the area under the graph then corresponds to the forward and backward Euler approximations to the initial value problem (9.1) with $u_0 = 0$ and $f = f(t)$.
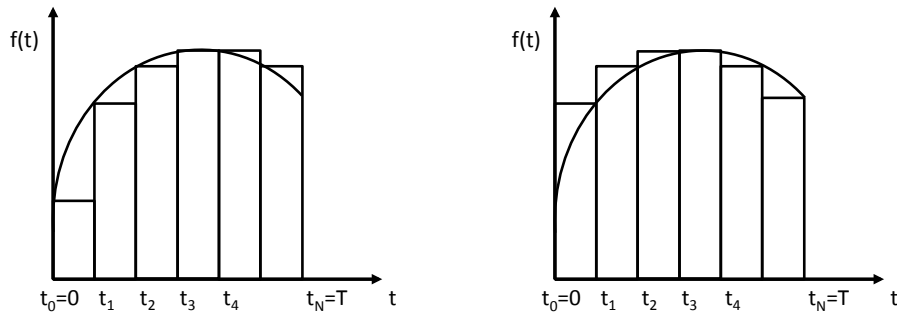


Figure 9.2: Left (left) and right (right) rectangular rule quadrature, or *Riemann sums*, approximating the primitive function of $f(t)$, corresponding to the area under the graph.

More generally, by the *Fundamental Theorem of Calculus* we have, for each subinterval $I_n = (t_{n-1}, t_n)$, that

$$u(t_n) = u(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(u(t), t) \, dt, \tag{9.8}$$

from which we can derive suitable time stepping methods corresponding to different quadrature rules used to evaluate the integral in (9.8).

## Quadrature as interpolation

Quadrature as an approximate integral of an exact function, can alternatively be expressed as exact integration of an approximate function. For example, the rectangular quadrature rules in Figure 9.2, corresponds to exact integration of a piecewise constant (over each subinterval $I_n$) approximation of the function $f(t)$, with its value determined by the function value at the left or right endpoint of the interval, $f(t_{n-1})$ or $f(t_n)$.

From this perspective, one may ask if such a piecewise constant approximation can be chosen in a more clever way to reduce the error in the approximation of the integral, which naturally leads to the *midpoint rule* where the piecewise constant function is chosen based on the function value at the midpoint of the subinterval, that is $(f(t_{n-1}) + f_n)/2$.

Further, we may seek to approximate the function by a higher order polynomial. By linear *interpolation* over the partition $\mathcal{T}_k$, corresponding approximation of the function by a continuous piecewise linear polynomial which is exact at each node $t_n$, exact integration corresponds to the *trapezoidal rule*.
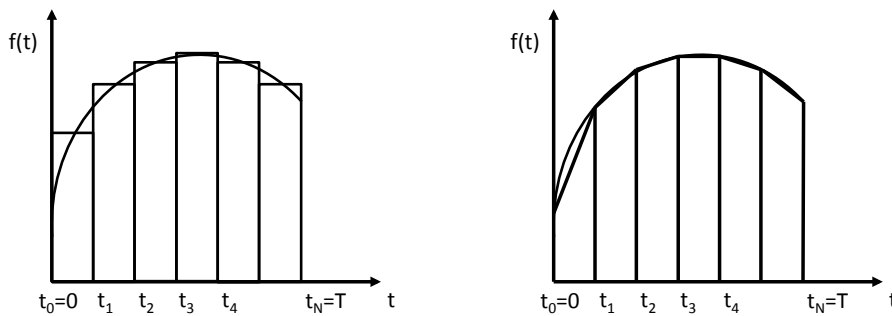


Figure 9.3: Midpoint (left) and trapezoidal (right) quadrature rules, corresponding to interpolation by a piecewise constant and piecewise linear function respectively.

## Interpolation as time stepping

By (9.8), the midpoint and trapezoidal rules can also be formulated as time stepping methods. Although, in the case of a time stepping method, interpolation cannot be directly based on the function $f(u(t), t)$ since $u(t)$ is unknown. Instead we seek an approximate solution $U(t)$ to the initial value

problem (9.1) as a piecewise polynomial of a certain order, determined by $f(U_n, t_n)$ through the successive approximations $U_n$.

---

**Algorithm 16:** Trapezoidal time stepping method

---

$U_0 = u_0$                          ▷ `initial approximation`

**for** $n = 1, 2, ..., N$ **do**

     $U_n = U_{n-1} + \dfrac{k_n}{2}(f(U_n, t_n) + f(U_{n-1}, t_{n-1}))$    ▷ `solve equation`

**end**

---

Both the midpoint method and the trapezoidal method are implicit, and thus require the solution of an algebraic equation, possibly nonlinear, at each time step.

To seek approximate solutions to differential equations as piecewise polynomials is a powerful idea that we will meet many times. Any piecewise polynomial function can be expressed as a linear combination of basis functions, for which the coordinates are to be determined, for example, based on minimization or orthogonality conditions on the residual of the differential equation.
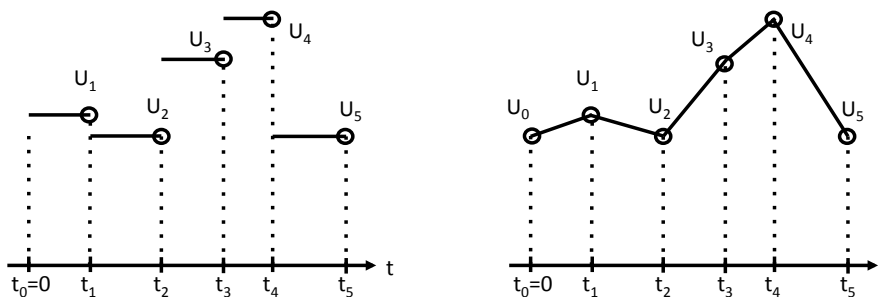


Figure 9.4: Examples of a discontinuous piecewise constant polynomial determined by its value at the right endpoint of the subinterval (left), and a continuous piecewise linear polynomial determined by its value in the nodes of the partition (right).

The residual of (9.1) is given by

$$R(U(t)) = f(U(t), t) - \dot{U}(t), \tag{9.9}$$

and in the case of a continuous piecewise linear approximation $U(t)$, an orthogonality condition enforces the integral of the residual to be zero over

each time interval $I_n$, that is

$$\int_{t_{n-1}}^{t_n} R(U(t)) \; dt = 0, \tag{9.10}$$

or equivalently,

$$U_n = U_{n-1} + \int_{t_{n-1}}^{t_n} f(U(t), t) \; dt,$$

which corresponds to (9.8). If $f(\cdot, \cdot)$ is a linear function, it follows that

$$U_n = U_{n-1} + \frac{k_n}{2}(f(U_n, t_n) + f(U_{n-1}, t_{n-1})), \tag{9.11}$$

else we have to choose a quadrature rule to approximate the integral, with (9.11) corresponding to a trapezoidal rule for a nonlinear function $f(\cdot, \cdot)$.

## The $\theta$-method

We can formulate the forward and backward Euler methods, and the trapezoidal method, as one single method with a parameter $\theta$, the $\theta$-*method*. For $\theta = 1$, we get the explicit Euler method, for $\theta = 0$ the implicit Euler method, and $\theta = 0.5$ corresponds to the trapezoidal rule.

---

**Algorithm 17:** The $\theta$-method for initial value problems

---
$U_0 = u_0$          $\triangleright$ `initial approximation`
**for** $n = 1, 2, ..., N - 1$ **do**
   |   $U_n = U_{n-1} + k_n((1 - \theta)f(U_n, t_n) + \theta f(U_{n-1}, t_{n-1}))$     $\triangleright$ `update`
**end**

---

**Theorem 16** (Local error estimate for the $\theta$-method). *For the $\theta$-method over one subinterval $I_n = (t_{n-1}, t_n)$ of length $k_n = t_n - t_{n-1}$, with $U_{n-1} = u(t_{n-1})$, we have the following local error estimate,*

$$|u(t_n) - U_n| = \mathcal{O}(k_n^3), \tag{9.12}$$

*for $\theta = 1/2$, and if $\theta \neq 1/2$,*

$$|u(t_n) - U_n| = \mathcal{O}(k_n^2). \tag{9.13}$$

*Proof.* With the notation $f_n = f(U_n, t_n)$, so that $f_{n-1} = \dot{u}(t_{n-1})$ and $f_n = \dot{u}(t_n)$, and observing that $U_{n-1} = u(t_{n-1})$, we have by Taylor's formula that

$$
\begin{aligned}
|u(t_n) - U_n| &= |u(t_n) - (U_{n-1} + k_n((1-\theta)f_n + \theta f_{n-1}))| \\
&= |u(t_{n-1}) + k_n \dot{u}(t_{n-1}) + \frac{1}{2}k_n^2 \ddot{u}(t_{n-1}) + \mathcal{O}(k_n^3) \\
&\quad - (u(t_{n-1}) + k_n((1-\theta)\dot{u}(t_n) + \theta \dot{u}(t_{n-1})))| \\
&= |k_n \dot{u}(t_{n-1}) + \frac{1}{2}k_n^2 \ddot{u}(t_{n-1}) + \mathcal{O}(k_n^3) \\
&\quad - (k_n((1-\theta)(\dot{u}(t_{n-1}) + k_n \ddot{u}(t_{n-1}) + \mathcal{O}(k_n^2)) + \theta \dot{u}(t_{n-1})))| \\
&= |\theta - \frac{1}{2}||\ddot{u}(t_{n-1})|k_n^2 + \mathcal{O}(k_n^3).
\end{aligned}
$$

$\square$

## 9.3   System of initial value problems

We now consider systems of initial value problems for a vector valued function $u : \mathbb{R}^+ \to \mathbb{R}^n$ defined on the interval $I = [0, T]$, with derivative $\dot{u} = du/dt = (du_1/dt, ..., du_n/dt)^T$ defined by the function $f : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}^n$, such that

$$
\begin{aligned}
\dot{u}(t) &= f(u(t), t), \quad 0 < t \le T, \\
u(0) &= u_0.
\end{aligned}
\tag{9.14}
$$

Time stepping methods for (9.14) are analogous to the scalar case (9.1), including the $\theta$-method of Algorithm 17, with the difference that for implicit methods a system of (possibly nonlinear) equations needs to be solved.

### Newton's laws of motion

Newton's laws of motion for a particle can be formulated as an initial value problem (9.14), with *Newton's 2nd law* expressing that force equals mass times acceleration,

$$
m\ddot{x}(t) = F(t),
\tag{9.15}
$$

given by

$$
u = \begin{bmatrix} v \\ x \end{bmatrix}, \quad f = \begin{bmatrix} F/m \\ v \end{bmatrix},
\tag{9.16}
$$

for $x = x(t)$ the particle position, $v = v(t) = \dot{x}(t)$ the velocity, $m$ the mass of the particle, and $F = F(t)$ the force applied.

For example, the force $F = mg$ models gravitation, with $g$ the gravitation constant, and the force $F = -kx$ models an elastic spring (Hooke's law) with spring constant $k$. *Newton's first law* that expresses that a particle remains in its state in the absence of a force, follows from (9.15) in the case $F = 0$.

## The $N$-body problem

*Newton's third law* states that if one particle $p_i$ exerts a force $F_{ji}$ on another particle $p_j$, then $p_j$ exerts a force $F_{ij} = -F_{ji}$ on $p_i$, of the same magnitude but in the opposite direction.

The *N-body problem* refers to the initial value problem (9.14) describing Newton's laws of motion for a system of $N$ particles $\{p_i\}_{i=1}^N$, with the pairwise force interactions $F_{ij}$ given by the system under study, for example gravitation in celestial mechanics, Coulomb interactions in electrostatics, Hookean springs in elasticity theory, or interatomic potentials in molecular dynamics simulations.

The $N$-body problem takes the form (9.14) in $\mathbb{R}^{2N}$,

$$u = \begin{bmatrix} v_1 \\ \vdots \\ v_N \\ x_1 \\ \vdots \\ x_N \end{bmatrix}, \quad f = \begin{bmatrix} F_1/m_1 \\ \vdots \\ F_N/m_N \\ v_1 \\ \vdots \\ v_N \end{bmatrix} \tag{9.17}$$

with the resulting force on particle $p_i$ given by the sum of all pairwise interactions,

$$F_i = \sum_{j \neq i}^N F_{ij}. \tag{9.18}$$

To solve (9.17) using a time-stepping is an $\mathcal{O}(N^2)$ algorithm, which is very expensive for $N$ large. Optimized algorithms of order $\mathcal{O}(N)$ can be developed based on the idea of clustering the force from multiple particles at a distance.

## Celestial mechanics

Newton's gravitational law models pairwise gravitational force interactions, which can be used to model the solar system for example. Every particle

$p_i$ is effected by the sum of all other particles, and the force $F_{ij}$ acting on particle $p_i$ by $p_j$, is given by,

$$F_{ij} = G\frac{m_i m_j}{\|x_i - x_j\|^2},$$ (9.19)

where $m_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^n$ denotes the mass and position of particle $p_i$.

## Mass-spring model

The forces in a mass-spring model represent pairwise force interactions between adjacent particles in a *lattice*, connected via springs, such that the force $F_{ij}$ acting on particle $p_i$ by $p_j$ is given by,

$$F_{ij} = -k_{ij}(x_i - x_j),$$ (9.20)

with $k_{ij} = k_{ji}$ the relevant spring constant.