Overview of numerical methods for ODEs

In this course note we provide a brief overview of common numerical methods to approximate the solution of the initial value problem (IVP)

$$\begin{cases} \dfrac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}, t) \\ \boldsymbol{y}(0) = \boldsymbol{y}_0 \end{cases} \tag{1}$$

where $\boldsymbol{y}(t) = [y_1(t) \cdots y_n(t)]^T$ is a (column) vector of phase variables, $\boldsymbol{f} : D \times [0, T] \to \mathbb{R}^n$, D is a subset of $\mathbb{R}^n$, and T is the integration period. We have seen in AM 214 that if $\boldsymbol{f}(\boldsymbol{y}, t)$ is Lipschitz continuous in an open set $D \subseteq \mathbb{R}^n$ then the IVP (1) well-posed in D, i.e., the solution exists and is unique for all $\boldsymbol{y}_0 \in D$, at least for some time $\tau > 0$. The initial value problem (1) can be written as

$$\boldsymbol{y}(t) = \boldsymbol{y}(0) + \int_0^t \boldsymbol{f}(\boldsymbol{y}(s), s) ds, \tag{2}$$

i.e., as an integral equation for $\boldsymbol{y}(t)$.

Picard iteration method. The Picard iteration method is rarely used in practice to compute numerical solutions to ODEs, but rather to prove existence and uniqueness of solutions to ODEs. Picard's method is essentially a fixed point iteration in which the solution $\boldsymbol{y}(t)$ is approximated within some time interval $[0, t_1]$ by the *sequence of functions*

$$\boldsymbol{y}^{[0]}(t) \to \boldsymbol{y}^{[1]}(t) \to \cdots \to \boldsymbol{y}^{[k]}(t) \to \cdots \qquad t \in [0, t_1] \tag{3}$$

generated by equation (2), i.e.,

$$\boldsymbol{y}^{[k+1]}(t) = \boldsymbol{y}_0 + \int_0^t \boldsymbol{f}\left(\boldsymbol{y}^{[k]}(s), s\right) ds, \qquad k = 0, 1, 2, \ldots, \qquad t \in [0, t_1]. \tag{4}$$

Setting $\boldsymbol{y}^{[0]}(t) = \boldsymbol{y}_0$ in (4) yields $\boldsymbol{y}^{[1]}(t)$. With $\boldsymbol{y}^{[1]}(t)$ available we can compute $\boldsymbol{y}^{[2]}(t)$, etc. A necessary and sufficient condition for convergence of the sequence of functions $\boldsymbol{y}^{[k]}(t)$ is that the nonlinear operator at the right-hand-side of (4), i.e.,

$$\boldsymbol{N}(\boldsymbol{y})(t) = \boldsymbol{y}_0 + \int_0^t \boldsymbol{f}\left(\boldsymbol{y}(s), s\right) ds \tag{5}$$

is a *contraction*[1] (Banach fixed point theorem). This statement essentially sets an upper bound on t_1 (integration time) that depends on the Lipschitz constant of $\boldsymbol{f}$. The larger the Lipschitz constant, the smaller t_1. In other words, Picard iterations converge if and only only if t_1 is chosen small enough, where "how small" depends on the Lipschitz constant of $\boldsymbol{f}$. Clearly, once $\boldsymbol{y}(t)$ has been computed to the desired accuracy in $[0, t_1]$, we can then use $\boldsymbol{y}(t_1)$ as initial condition for the next Picard iterations, i.e.,

$$\boldsymbol{y}^{[k+1]}(t) = \boldsymbol{y}(t_1) + \int_{t_1}^t \boldsymbol{f}\left(\boldsymbol{y}^{[k]}(s), s\right) ds \qquad t \in [t_1, t_2]. \tag{8}$$

The Picard iteration method is not practical as it involves the evaluation of a integral at each iteration k. Moreover, the integral has "t" as one of the endpoints.

[1] Let X be a function space with norm $\|\cdot\|_X$. An operator $\boldsymbol{N} : X \mapsto X$ is called a (strict) *contraction* if

$$\left\| \boldsymbol{N}\left(\boldsymbol{y}^{[k]}\right) - \boldsymbol{N}\left(\boldsymbol{y}^{[j]}\right) \right\|_X < \left\| \boldsymbol{y}^{[k]} - \boldsymbol{y}^{[j]} \right\|_X \tag{6}$$

In the case of (5) we have

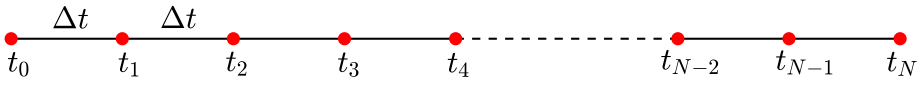
$$\left\| \boldsymbol{N}\left(\boldsymbol{y}^{[k]}\right) - \boldsymbol{N}\left(\boldsymbol{y}^{[j]}\right)(t) \right\|_X \leq \left\| \int_0^{t_1} \left[\boldsymbol{f}\left(\boldsymbol{y}^{[k]}(s), s\right) - \boldsymbol{f}\left(\boldsymbol{y}^{[j]}(s), s\right) \right] ds \right\|_X . \tag{7}$$

Elementary numerical methods

Let us partition the time interval $[0,T]$ into an evenly-spaced grid with N time instants

$$t_0 = 0, \qquad t_N = T, \qquad t_{i+1} = t_i + \Delta t \qquad i = 0, \ldots, N-1, \tag{9}$$

where $\Delta t = T/N$.



By using the semi-group property satisfied by the solution of ODE (1) we can write (2) within each time interval $[t_k, t_{k+1}]$ as

$$\boldsymbol{y}(t_{k+1}) = \boldsymbol{y}(t_k) + \int_{t_k}^{t_{k+1}} \boldsymbol{f}\left(\boldsymbol{y}(s), s\right) ds. \tag{10}$$

This formulation is quite convenient for developing numerical methods for ODEs based on *quadrature formulas*, i.e., numerical approximations of the one-dimensional temporal integral appearing at the right hand side of (10).

Euler and Crank-Nicolson methods

These are basic numerical schemes that can be obtained by approximating the integral at the right hand side of equation (10) by using the rectangle rule or the trapezoidal rule (see Figure 1). Specifically,

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}\left(\boldsymbol{y}(s), s\right) ds \simeq \Delta t \boldsymbol{f}\left(\boldsymbol{y}(t_k), t_k\right), \tag{11}$$

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}\left(\boldsymbol{y}(s), s\right) ds \simeq \Delta t \boldsymbol{f}\left(\boldsymbol{y}(t_{k+1}), t_{k+1}\right), \tag{12}$$

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}\left(\boldsymbol{y}(s), s\right) ds \simeq \frac{\Delta t}{2}\left[\boldsymbol{f}\left(\boldsymbol{y}(t_k), t_k\right) + \boldsymbol{f}\left(\boldsymbol{y}(t_{k+1}), t_{k+1}\right)\right]. \tag{13}$$

Note that all these approximations are obtained via interpolatory quadratures, i.e., by replacing $\boldsymbol{f}\left(\boldsymbol{y}(s), s\right)$ with a polynomial in $[t_k, t_{k+1}]$ and then integrating the polynomial. In particular, in (11) and (12) we use constant polynomials while in (13) we use a linear polynomial. A substitution of (11)-(13) into (10) yields, respectively, the following numerical schemes

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}\left(\boldsymbol{u}_k, t_k\right) \qquad \text{Euler forward method (explicit)}, \tag{14}$$

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}\left(\boldsymbol{u}_{k+1}, t_{k+1}\right) \qquad \text{Euler backward method (implicit)}, \tag{15}$$

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2}\left[\boldsymbol{f}\left(\boldsymbol{u}_k, t_k\right) + \boldsymbol{f}\left(\boldsymbol{u}_{k+1}, t_{k+1}\right)\right] \qquad \text{Crank-Nicolson method (implicit)}. \tag{16}$$

In these methods $\boldsymbol{u}_k$ represents an approximation of the exact solution $\boldsymbol{y}(t_k)$. In Figure 2 we sketch the difference between the analytical solution of (2) evaluated on a temporal grid, i.e., $\{\boldsymbol{y}(t_0), \boldsymbol{y}(t_1), \ldots\}$ and the numerical solution $\boldsymbol{u}_k$ obtained by iterating any of the schemes (14)-(16) (as a matter of fact any numerical scheme).

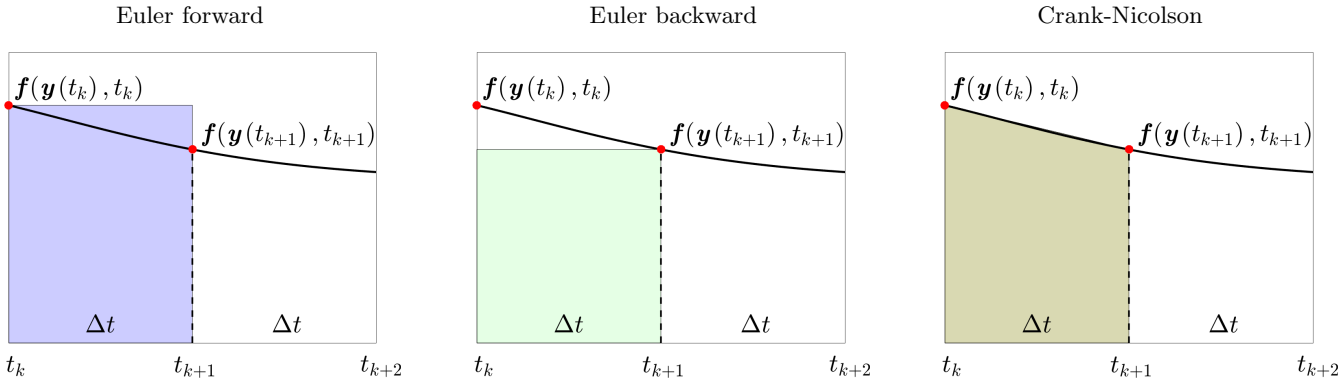


Figure 1: Elementary approximation of the integral $\int_{t_i}^{t_{i+1}} \boldsymbol{f}(\boldsymbol{y}(s), s)ds$ in equation (10) leading to Euler forward, Euler backward, and Crank-Nicolson methods.

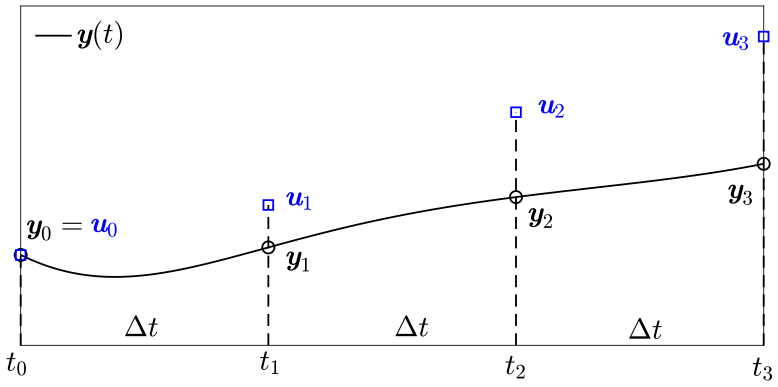


Figure 2: Analytical solution $\boldsymbol{y}(t)$ of the ODE (2) evaluated on a temporal grid $\{t_0, t_1, \ldots\}$, i.e., $\boldsymbol{y}_i = \boldsymbol{y}(t_i)$, and numerical solution $\boldsymbol{u}_i$ obtained by iterating any of the schemes (14)-(16). Here we assume that the initial condition used for the numerical scheme is exact, i.e. we set $\boldsymbol{u}_0 = \boldsymbol{y}_0$.

Both Euler and Crank-Nicolson methods are *one-step methods*. This means that the approximate solution at time t_{k+1}, i.e., $\boldsymbol{u}_{k+1}$, can be computed based on the solution (or its approximation) at time t_k. The Euler forward method allows us to compute $\boldsymbol{u}_{k+1}$ *explicitly*, given $\boldsymbol{u}_k$ and $\boldsymbol{f}(\boldsymbol{u}_k, t_k)$. On the other hand, the Euler backward and Crank-Nicolson are *implicit* methods. This is because the approximate solution at time t_{k+1}, i.e., $\boldsymbol{u}_{k+1}$, cannot be computed explicitly based on $\boldsymbol{u}_k$, but it requires solving a nonlinear (algebraic) equation. Specifically, for the Euler backward method we need to solve

$$\boldsymbol{u}_{k+1} = \boldsymbol{G}(\boldsymbol{u}_{k+1}) \qquad \text{where} \qquad \boldsymbol{G}(\boldsymbol{u}_{k+1}) = \boldsymbol{u}_k + \Delta t \boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}). \tag{17}$$

Nonlinear equations of this form can be solved numerically using iterative methods (fixed point iterations) such as the Newton's method (provided $\boldsymbol{f}$ is of class C^1). Upon definition of

$$\boldsymbol{F}(\boldsymbol{u}_{k+1}) = \boldsymbol{u}_{k+1} - \boldsymbol{G}(\boldsymbol{u}_{k+1}) \tag{18}$$

we can write (17) as

$$\boldsymbol{F}(\boldsymbol{u}_{k+1}) = \boldsymbol{0}, \qquad \Leftrightarrow \qquad \boldsymbol{u}_{k+1} \text{ is a zero of } \boldsymbol{F}. \tag{19}$$

Equivalently, we can say that $\boldsymbol{u}_{k+1}$ is a fixed point of the map $\boldsymbol{G}$ defined in (17).

Newton's method for nonlinear systems. To solve the nonlinear system of equations (19) one can use Newton's method or any other iterative root finding method. As is well known, the Newton's method

generates a sequence of vectors $\boldsymbol{u}_{k+1}^{[j]}$ $(j=0,1,\ldots)$ converging to $\boldsymbol{u}_{k+1}$ under rather mild assumptions (see Theorem 1 below, and [4, Ch. 7]). The Newton's method for nonlinear systems can be written as[2]

$$\underbrace{\left[\boldsymbol{I}-\boldsymbol{J}_{\boldsymbol{G}}\left(\boldsymbol{u}_{k+1}^{[j]}\right)\right]}_{\text{matrix}}\underbrace{\left(\boldsymbol{u}_{k+1}^{[j+1]}-\boldsymbol{u}_{k+1}^{[j]}\right)}_{\text{vector}}=\underbrace{\boldsymbol{G}\left(\boldsymbol{u}_{k+1}^{[j]}\right)-\boldsymbol{u}_{k+1}^{[j]}}_{\text{vector}}\qquad j=0,1,\ldots, \tag{22}$$

where $\boldsymbol{J}_{\boldsymbol{G}}\left(\boldsymbol{u}_{k+1}^{[j]}\right)$ is the Jacobian matrix of $\boldsymbol{G}$ defined in equation (17), evaluated at $\boldsymbol{u}_{k+1}^{[j]}$. Equation (22) allows us to compute $\boldsymbol{u}_{k+1}^{[j+1]}$ given $\boldsymbol{u}_{k+1}^{[j]}$ by solving a linear system. In practice, it is convenient to set the initial guess $\boldsymbol{u}_{k+1}^{[0]}$ for Newton's iteration as $\boldsymbol{u}_{k+1}^{[0]}=\boldsymbol{u}_k$, i.e., the numerical solution at previous time step. For Δt sufficiently small the matrix at the left hand side of (22) is a perturbation of the identity (the norm of $\boldsymbol{J}_{\boldsymbol{G}}$ goes to zero linearly in Δt), and therefore $\boldsymbol{I}-\boldsymbol{J}_{\boldsymbol{G}}$ is always invertible for sufficiently small Δt. Indeed, $\boldsymbol{I}-\boldsymbol{J}_{\boldsymbol{G}}$ is diagonally dominant for small Δt. More rigorously, we have the following convergence result (see [4, Theorem 7.1]).

Theorem 1 (Convergence of Newton's method)**.** Let $\boldsymbol{F}(\boldsymbol{x})$ in equation (18) be of class C^1 in a convex open set $D\subseteq\mathbb{R}^n$ that contains a zero of $\boldsymbol{F}$, i.e., a point $\boldsymbol{x}^*\in D$ such that $\boldsymbol{F}(\boldsymbol{x}^*)=\boldsymbol{0}$. If $\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{x})$ is invertible at $\boldsymbol{x}^*$ (it always is for sufficiently small Δt), and $\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{x})$ is Lipschitz continuous in a neighborhood of $\boldsymbol{x}^*$, i.e.,

$$\|\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{x})-\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{y})\|\leq L\,\|\boldsymbol{x}-\boldsymbol{y}\| \tag{23}$$

then there exists a neighborhood of $\boldsymbol{x}^*$ such that for any initial guess $\boldsymbol{x}^{[0]}$ within such a neighborhood we have that the sequence $\boldsymbol{x}^{[k]}$ generated by

$$\boldsymbol{J}_{\boldsymbol{F}}\left(\boldsymbol{x}^{[k]}\right)\left(\boldsymbol{x}^{[k+1]}-\boldsymbol{x}^{[k]}\right)=-\boldsymbol{F}\left(\boldsymbol{x}^{[k]}\right) \tag{24}$$

converges to $\boldsymbol{x}^*$ with order 2. In other words, there exists $m\in\mathbb{N}$ such that

$$\left\|\boldsymbol{x}^{[k+1]}-\boldsymbol{x}^*\right\|\leq C\left\|\boldsymbol{x}^{[k]}-\boldsymbol{x}^*\right\|^2\quad\text{for all } k\geq m. \tag{25}$$

In equation (23) the norm at the left-hand-side is any matrix norm compatible with the vector norm at the right hand side (see Appendix A).

Remark: We emphasize that explicit and implicit Euler methods (14)-(15) can be also derived by replacing $d\boldsymbol{y}/dt$ in (1) with the first-order forward/backward finite-differentiation approximation

$$\frac{d\boldsymbol{y}(t_k)}{dt}\simeq\frac{\boldsymbol{y}(t_{k+1})-\boldsymbol{y}(t_k)}{\Delta t}=\boldsymbol{f}(\boldsymbol{y}(t_k),t_k), \tag{26}$$

$$\frac{d\boldsymbol{y}(t_{k+1})}{dt}\simeq\frac{\boldsymbol{y}(t_{k+1})-\boldsymbol{y}(t_k)}{\Delta t}=\boldsymbol{f}(\boldsymbol{y}(t_{k+1}),t_{k+1}). \tag{27}$$

[2] To derive the Newton's method for nonlinear systems of algebraic equations, consider the Taylor series

$$\boldsymbol{F}\left(\boldsymbol{u}_{k+1}^{[j+1]}\right)=\boldsymbol{F}\left(\boldsymbol{u}_{k+1}^{[j]}\right)+\boldsymbol{J}_{\boldsymbol{F}}\left(\boldsymbol{u}_{k+1}^{[j]}\right)\left(\boldsymbol{u}_{k+1}^{[j+1]}-\boldsymbol{u}_{k+1}^{[j]}\right)+\cdots\qquad j=0,1,\ldots. \tag{20}$$

Setting $\boldsymbol{F}\left(\boldsymbol{u}_{k+1}^{[j+1]}\right)=\boldsymbol{0}$ yields

$$\boldsymbol{J}_{\boldsymbol{F}}\left(\boldsymbol{u}_{k+1}^{[j]}\right)\left(\boldsymbol{u}_{k+1}^{[j+1]}-\boldsymbol{u}_{k+1}^{[j]}\right)=-\boldsymbol{F}\left(\boldsymbol{u}_{k+1}^{[j]}\right)\qquad j=0,1,\ldots \tag{21}$$

which, upon substitution of (18), coincides with the Newton method (22).

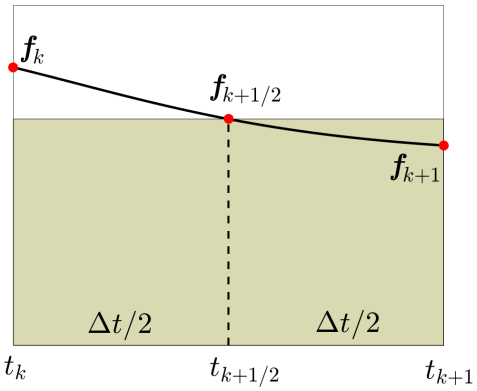


Figure 3: Approximation of the integral $\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s),s)ds$ in (10) leading to midpoint method. Here we set $\boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{y}(t_k),t_k)$.

Heun's method

If we replace $\boldsymbol{u}_{k+1}$ at the right hand side of (16) with one step of the Euler forward scheme (14) we obtain the *Heun method*

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2}\left[\boldsymbol{f}\left(\boldsymbol{u}_k,t_k\right) + \boldsymbol{f}\left(\boldsymbol{u}_k + \Delta t \boldsymbol{f}(\boldsymbol{u}_k,t_k),t_{k+1}\right)\right] \qquad \text{Heun method (explicit)} \tag{28}$$

The Heun method is a one-step explicit method that belongs to the class of two-stage explicit Runge-Kutta methods discussed below.

Midpoint method

We approximate the integral at the right hand side of (10) using the midpoint rule yields

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s),s)ds \simeq \Delta t \boldsymbol{f}\left(\boldsymbol{y}\left(t_k + \frac{\Delta t}{2}\right), t_k + \frac{\Delta t}{2}\right). \tag{29}$$

At this point, we approximate $\boldsymbol{y}(t_k + \Delta t/2)$ using the Euler forward method to obtain

$$\boldsymbol{y}\left(t_k + \frac{\Delta t}{2}\right) \simeq \boldsymbol{y}(t_k) + \frac{\Delta t}{2}\boldsymbol{f}(\boldsymbol{y}(t_k),t_k) \tag{30}$$

to obtain

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}\left(\boldsymbol{u}_k + \frac{\Delta t}{2}\boldsymbol{f}(\boldsymbol{u}_k,t_k), t_k + \frac{\Delta t}{2}\right) \qquad \text{explicit mipoint method} \tag{31}$$

where $\boldsymbol{u}_k$ is an approximation of $\boldsymbol{y}(t_k)$. The explicit midpoint method is a one-step method that belongs to the class of two-stage explicit Runge-Kutta methods. Note that the explicit midpoint method is different from the Heun method (28).

Approximating $\boldsymbol{y}(t_k + \Delta t/2)$ by the average

$$\boldsymbol{y}\left(t_k + \frac{\Delta t}{2}\right) \simeq \frac{\boldsymbol{y}(t_k) + \boldsymbol{y}(t_{k+1})}{2} \tag{32}$$

yields

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}\left(\frac{\boldsymbol{u}_k + \boldsymbol{u}_{k+1}}{2}, t_k + \frac{\Delta t}{2}\right) \qquad \text{implicit mipoint method} \tag{33}$$

The implicit midpoint method is a *one-step symplectic integrator*, i.e., the scheme preserves the Hamiltonian when applied to Hamiltonian dynamical systems, e.g., pendulum or double-pendulum equations. There is a vast literature on *structure-preserving* integration methods for ODEs (see, e.g., [1]).

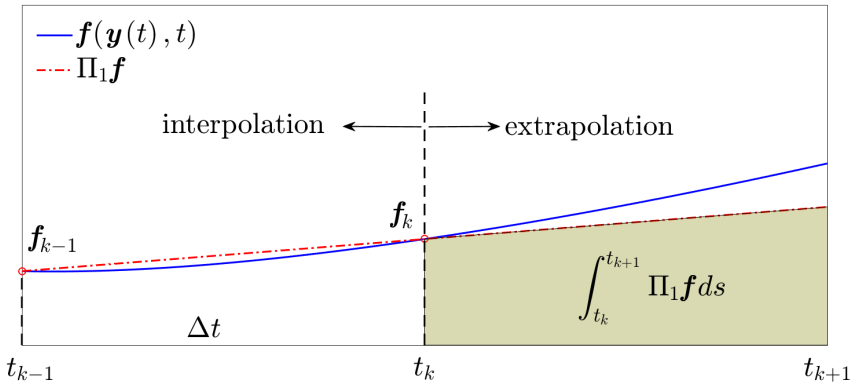


Figure 4: Derivation of the two-step Adams-Bashforth scheme (AB2). We first construct the polynomial (line) $\Pi_1\boldsymbol{f}$ that interpolates $\boldsymbol{f}(\boldsymbol{y}(s),s)$ at t_{k-1} and t_k. Subsequently, we extrapolate $\Pi_1\boldsymbol{f}$ to $[t_k,t_{k+1}]$ and compute the integral in equation (34).

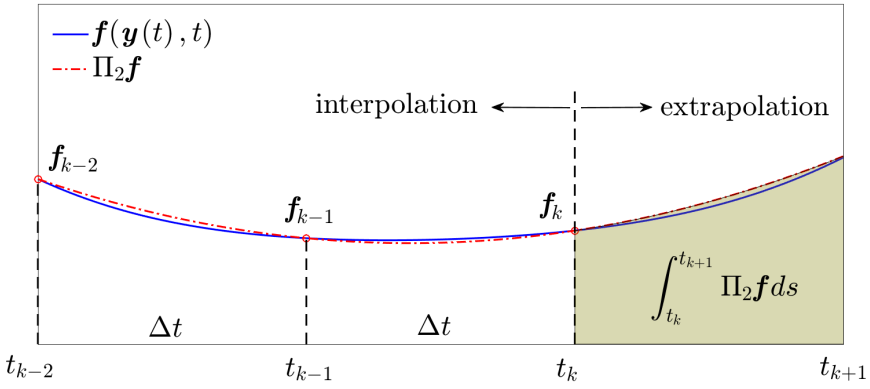


Figure 5: Derivation of the three-step Adams-Bashforth scheme (AB3). We first construct the polynomial $\Pi_2\boldsymbol{f}$ that interpolates $\boldsymbol{f}(\boldsymbol{y}(s),s)$ at t_{k-2}, t_{k-1} and t_k. Subsequently, we extrapolate $\Pi_2\boldsymbol{f}$ to $[t_k,t_{k+1}]$ and compute the integral in equation (34).

Adams-Bashforth methods

These are *explicit multistep methods* constructed by replacing the integral at the right hand side of (10) with the integral of a polynomial interpolant of $\boldsymbol{f}(\boldsymbol{y}(s),s)$ at $\{t_k,t_{k-1},\cdots t_{k-q}\}$ which is then extrapolated into $[t_k,t_{k+1}]$ to compute the integral. In other words, we introduce the following approximation

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s),s)ds \simeq \int_{t_k}^{t_{k+1}} \Pi_q\boldsymbol{f}(\boldsymbol{y}(s),s)ds \tag{34}$$

where $\Pi_q\boldsymbol{f}(\boldsymbol{y}(s),s)$ is a polynomial of degree q interpolating

$$\{\boldsymbol{f}_k,\boldsymbol{f}_{k-1},\ldots,\boldsymbol{f}_{k-q}\} \qquad \text{at} \qquad \{t_k,t_{k-1},\ldots,t_{k-q}\} \tag{35}$$

where $\boldsymbol{f}_k=\boldsymbol{f}(\boldsymbol{y}(t_k),t_k)$.

It is very convenient to use Lagrangian interpolation to derive the polynomial $\Pi_q\boldsymbol{f}$. Hereafter we derive the Adams-Bashforth (AB) methods for $q=0,1,2$.

- **One-step Adams-Bashforth method (AB1)**:

$$q=0 \quad \Rightarrow \quad \Pi_0\boldsymbol{f}(\boldsymbol{y}(s),s)=\boldsymbol{f}_k \tag{36}$$

where $\boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{y}(t_k), t_k)$. Hence,

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s), s)ds \simeq \int_{t_k}^{t_{k+1}} \Pi_0 \boldsymbol{f}(\boldsymbol{y}(s), s)ds = \Delta t \boldsymbol{f}(\boldsymbol{y}(t_k), t_k). \tag{37}$$

This yields

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{f}(\boldsymbol{u}_k, t_k) \qquad \text{AB1 method.} \tag{38}$$

Note that the AB1 method coincides with the Euler forward method.

- **Two-step Adams-Bashforth method (AB2)**: Here we have (see Figure 4)

$$q = 1 \quad \Rightarrow \quad \Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s) = \boldsymbol{f}_k l_k(s) + \boldsymbol{f}_{k-1} l_{k-1}(s) \tag{39}$$

where $l_k(s)$ and $l_{k-1}(s)$ are Lagrange characteristic polynomials

$$l_k(s) = \frac{s - t_{k-1}}{t_k - t_{k-1}}, \qquad l_{k-1}(s) = \frac{s - t_k}{t_{k-1} - t_k}. \tag{40}$$

The interpolant can be explicitly written as

$$\Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s) = \boldsymbol{f}_k \frac{s - t_{k-1}}{t_k - t_{k-1}} + \boldsymbol{f}_{k-1} \frac{s - t_k}{t_{k-1} - t_k}. \tag{41}$$

The (linear) polynomial $\Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s)$ is constructed in $[t_{k-1}, t_k]$ and it can be integrated exactly in $[t_k, t_{k+1}]$ using the trapezoidal rule. To this end, we notice that

$$\Pi_1 \boldsymbol{f}(\boldsymbol{y}(t_k), t_k) = \boldsymbol{f}_k, \tag{42}$$

$$\begin{aligned} \Pi_1 \boldsymbol{f}(\boldsymbol{y}(t_{k+1}), t_{k+1}) =& \boldsymbol{f}_k \frac{t_{k+1} - t_{k-1}}{t_k - t_{k-1}} + \boldsymbol{f}_{k-1} \frac{t_{k+1} - t_k}{t_{k-1} - t_k} \\ =& 2\boldsymbol{f}_k - \boldsymbol{f}_{k-1}. \end{aligned} \tag{43}$$

which gives us

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s), s)ds \simeq \int_{t_k}^{t_{k+1}} \Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s)ds = \frac{\Delta t}{2} \left(3\boldsymbol{f}_k - \boldsymbol{f}_{k-1}\right). \tag{44}$$

Substituting this back into (10) yields the scheme

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2} \left[3\boldsymbol{f}(\boldsymbol{u}_k, t_k) - \boldsymbol{f}(\boldsymbol{u}_{k-1}, t_{k-1})\right] \qquad \text{AB2 method.} \tag{45}$$

- **Three-step Adams-Bashforth method (AB3)**: With reference to Figure 5

$$q = 2 \quad \Rightarrow \quad \Pi_2 \boldsymbol{f}(\boldsymbol{y}(s), s) = \boldsymbol{f}_k l_k(s) + \boldsymbol{f}_{k-1} l_{k-1}(s) + \boldsymbol{f}_{k-2} l_{k-2}(s), \tag{46}$$

where $l_k(s)$, $l_{k-1}(s)$ and $l_{k-2}(s)$ are Lagrange characteristic polynomials

$$l_k(s) = \frac{s - t_{k-1}}{t_k - t_{k-1}} \frac{s - t_{k-2}}{t_k - t_{k-2}}, \tag{47}$$

$$l_{k-1}(s) = \frac{s - t_k}{t_{k-1} - t_k} \frac{s - t_{k-2}}{t_{k-1} - t_{k-2}}, \tag{48}$$

$$l_{k-2}(s) = \frac{s - t_k}{t_{k-2} - t_k} \frac{s - t_{k-1}}{t_{k-2} - t_{k-1}}. \tag{49}$$

By integrating $\Pi_2 \boldsymbol{f}(\boldsymbol{y}(s), s)$ from t_k to t_{k+1} we obtain

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s), s)ds \simeq \int_{t_k}^{t_{k+1}} \Pi_2 \boldsymbol{f}(\boldsymbol{y}(s), s)ds = \frac{\Delta t}{12} \left(23\boldsymbol{f}_k - 16\boldsymbol{f}_{k-1} + 5\boldsymbol{f}_{k-2}\right). \tag{50}$$

Substituting this back into (10) yields the scheme

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{12} \left[23\boldsymbol{f}(\boldsymbol{u}_k, t_k) - 16\boldsymbol{f}(\boldsymbol{u}_{k-1}, t_{k-1}) + 5\boldsymbol{f}(\boldsymbol{u}_{k-2}, t_{k-2})\right] \qquad \text{AB3 method.} \tag{51}$$

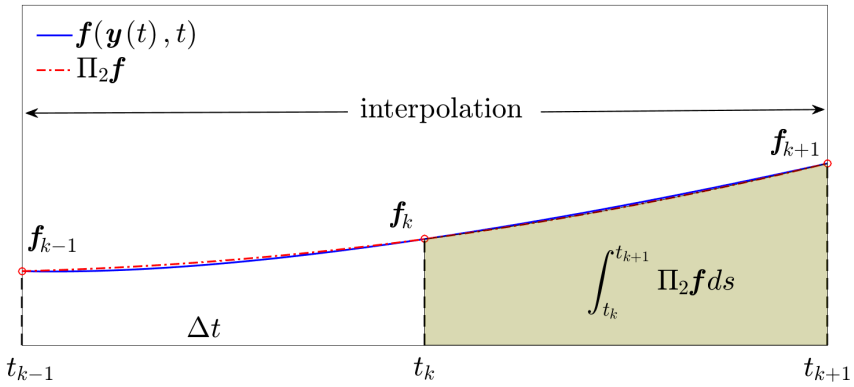


Figure 6: Derivation of the two-step Adams-Moulton scheme (AM2). We first construct the polynomial $\Pi_2\boldsymbol{f}$ that interpolates $\boldsymbol{f}(\boldsymbol{y}(s),s)$ at t_{k-1}, t_k and t_{k+1}. Subsequently, we use $\Pi_2\boldsymbol{f}$ polynomial to approximate the integral in equation (34).

Higher-order Adams-Bashforth schemes can be obtained similarly. Note that in order to start-up a linear multistep scheme we need to compute the solution at the intermediate steps using different methods. For example, we could start-up the AB2 method with one step of the Heun method (to compute $\boldsymbol{u}_1$) and then carry on the integration using the scheme (45)

Adams-Moulton methods

These are implicit multistep methods in which the time integral at the right hand-side of (10) is approximated by replacing $\boldsymbol{f}(\boldsymbol{y}(s),s)$ by a polynomial $\Pi_q\boldsymbol{f}(\boldsymbol{y}(s),s)$ of degree q interpolating

$$\{\boldsymbol{f}_{k+1},\boldsymbol{f}_k,\ldots,\boldsymbol{f}_{k-q+1}\} \qquad \text{at} \qquad \{t_{k+1},t_{k-1},\ldots,t_{k-q+1}\} \tag{52}$$

The main difference with respect to Adams-Bashforth methods is that there is no extrapolation step, i.e., the point $(t_{k+1},\boldsymbol{f}_{k+1})$ is included in the interpolation process (see Figure 6). Let us derive the Adams-Moulton schemes for $q=0,1,2$.

- **One-step Adams-Moulton method (AM0)**:

$$q=0 \quad \Rightarrow \quad \Pi_0\boldsymbol{f}(\boldsymbol{y}(s),s)=\boldsymbol{f}_{k+1} \tag{53}$$

  where $\boldsymbol{f}_{k+1}=\boldsymbol{f}(\boldsymbol{y}(t_{k+1}),t_{k+1})$. Hence,

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s),s)ds \simeq \int_{t_k}^{t_{k+1}} \Pi_0\boldsymbol{f}(\boldsymbol{y}(s),s)ds = \Delta t\boldsymbol{f}(\boldsymbol{y}(t_{k+1}),t_{k+1}). \tag{54}$$

  This yields

$$\boldsymbol{u}_{k+1}=\boldsymbol{u}_k+\Delta t\boldsymbol{f}(\boldsymbol{u}_{k+1},t_{k+1}) \qquad \text{AM0 method.} \tag{55}$$

  Note that the AM0 method coincides with the Euler backward method.

- **One-step Adams-Moulton method (AM1)**:

$$q=1 \quad \Rightarrow \quad \Pi_1\boldsymbol{f}(\boldsymbol{y}(s),s)=\boldsymbol{f}_{k+1}l_{k+1}(s)+\boldsymbol{f}_k l_k(s) \tag{56}$$

  where $l_k(s)$ and $l_{k-1}(s)$ are Lagrange characteristic polynomials

$$l_{k+1}(s)=\frac{s-t_k}{t_{k+1}-t_k}, \qquad l_k(s)=\frac{s-t_{k+1}}{t_k-t_{k+1}}. \tag{57}$$

This yields

$$\Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s) = \boldsymbol{f}_{k+1} \frac{s - t_k}{t_{k+1} - t_k} + \boldsymbol{f}_k \frac{s - t_{k+1}}{t_k - t_{k+1}}. \tag{58}$$

The (linear) polynomial $\Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s)$ is constructed in $[t_k, t_{k+1}]$ and it can be integrated exactly in the same interval. To this end, we notice that

$$\Pi_1 \boldsymbol{f}(\boldsymbol{y}(t_k), t_k) = \boldsymbol{f}_k, \tag{59}$$

$$\Pi_1 \boldsymbol{f}(\boldsymbol{y}(t_{k+1}), t_{k+1}) = \boldsymbol{f}_{k+1} \tag{60}$$

which imply

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s), s) ds \simeq \int_{t_k}^{t_{k+1}} \Pi_1 \boldsymbol{f}(\boldsymbol{y}(s), s) ds = \frac{\Delta t}{2} \left(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k \right). \tag{61}$$

Substituting this back into (10) yields the scheme

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2} \left[\boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}) + \boldsymbol{f}(\boldsymbol{u}_k, t_k) \right] \qquad \text{AM1 method} \tag{62}$$

Hence, the AM1 scheme coincides with the Crank-Nicolson scheme.

- **Two-step Adams-Moulton method (AM2)**:

$$q = 2 \quad \Rightarrow \quad \Pi_2 \boldsymbol{f}(\boldsymbol{y}(s), s) = \boldsymbol{f}_{k+1} l_{k+1}(s) + \boldsymbol{f}_k l_k(s) + \boldsymbol{f}_{k-1} l_{k-1}(s). \tag{63}$$

where $l_{k+1}(s)$, $l_k(s)$ and $l_{k-1}(s)$ are Lagrange characteristic polynomials

$$l_{k+1}(s) = \frac{s - t_k}{t_{k+1} - t_k} \frac{s - t_{k-1}}{t_{k+1} - t_{k-1}}, \tag{64}$$

$$l_k(s) = \frac{s - t_{k+1}}{t_k - t_{k+1}} \frac{s - t_{k-1}}{t_k - t_{k-1}}, \tag{65}$$

$$l_{k-1}(s) = \frac{s - t_{k+1}}{t_{k-1} - t_{k+1}} \frac{s - t_k}{t_{k-1} - t_k}. \tag{66}$$

By integrating $\Pi_2 \boldsymbol{f}(\boldsymbol{y}(s), s)$ from t_k to t_{k+1} we obtain

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s), s) ds \simeq \int_{t_k}^{t_{k+1}} \Pi_2 \boldsymbol{f}(\boldsymbol{y}(s), s) ds = \frac{\Delta t}{12} \left(5\boldsymbol{f}_{k+1} + 8\boldsymbol{f}_k - \boldsymbol{f}_{k-1} \right). \tag{67}$$

Substituting this back into (10) yields the scheme

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{12} \left[5\boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}) + 8\boldsymbol{f}(\boldsymbol{u}_k, t_k) - \boldsymbol{f}(\boldsymbol{u}_{k-1}, t_{k-1}) \right] \qquad \text{AM2 method.} \tag{68}$$

Backward differentiation formulas (BDF) methods

These are implicit multistep methods that perform well for stiff problems. These methods are obtained by approximating $d\boldsymbol{y}(t)/dt$ in (1) using a backward finite-difference (BFD) formula. These formulas are obtained by interpolating $\boldsymbol{y}(s)$ at $\{t_{k+1}, t_k, \ldots, t_{k-q+1}\}$ with a polynomial of degree q, differentiating the polynomial and evaluating the derivative at $t = t_{k+1}$ (see Figure 7). Let us derive the first few BDF methods.

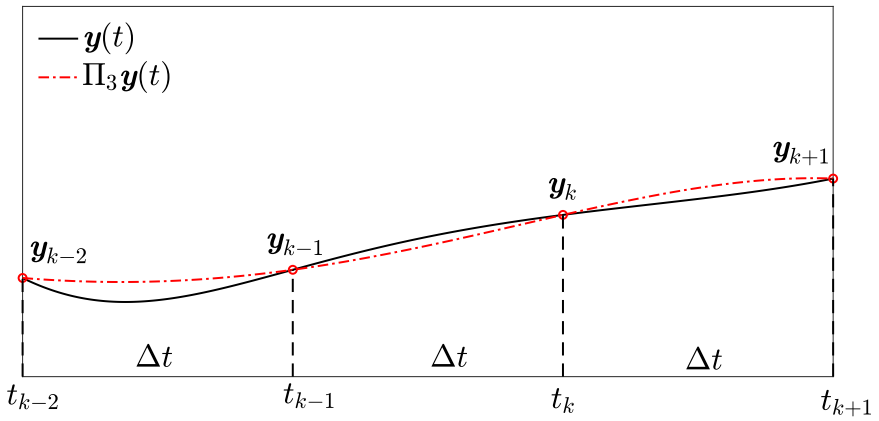


Figure 7: Derivation of the three-step backward differentiation formula (BDF3) method. We first construct the polynomial $\Pi_3\boldsymbol{f}$ that interpolates $\boldsymbol{y}(t)$ at t_{k-2}, t_{k-1}, t_k, and t_{k+1}. Subsequently, approximate the derivative of $\boldsymbol{y}(t)$ at t_{k+1} with the derivative of the polynomial $\Pi_3\boldsymbol{y}(t)$ at t_{k+1}.

- **One-step BDF method (BDF1):**

$$\Pi_1\boldsymbol{y}(s) = \boldsymbol{y}_{k+1}l_{k+1}(s) + \boldsymbol{y}_k l_k(s) \tag{69}$$

where the Lagrange characteristic polynomials are given by

$$l_{k+1}(s) = \frac{s - t_k}{t_{k+1} - t_k}, \qquad l_k(s) = \frac{s - t_{k+1}}{t_k - t_{k+1}} \tag{70}$$

We approximate the derivative of $\boldsymbol{y}(s)$ at t_{k+1} with the derivative of the polynomial $\Pi_1\boldsymbol{y}(s)$ at t_{k+1}, i.e.,

$$\frac{d\boldsymbol{y}(t_{k+1})}{dt} \simeq \frac{d\Pi_1\boldsymbol{y}(t_{k+1})}{dt} = \frac{\boldsymbol{y}_{k+1} - \boldsymbol{y}_k}{\Delta t}. \tag{71}$$

Substituting this approximation into the exact equation

$$\frac{d\boldsymbol{y}(t_{k+1})}{dt} = \boldsymbol{f}(\boldsymbol{y}(t_{k+1}), t_{k+1}) \tag{72}$$

yields the scheme

$$\boldsymbol{u}_{k+1} - \boldsymbol{u}_k = \Delta t\boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}). \tag{73}$$

Note that BDF1 coincides with the Euler backward scheme.

- **Two-step BDF method (BDF2):**

$$\Pi_2\boldsymbol{y}(s) = \boldsymbol{y}_{k+1}l_{k+1}(s) + \boldsymbol{y}_k l_k(s) + \boldsymbol{y}_{k-1}l_{k-1}(s) \tag{74}$$

where the Lagrange characteristic polynomials are given by

$$l_{k+1}(s) = \frac{s - t_k}{t_{k+1} - t_k}\frac{s - t_{k-1}}{t_{k+1} - t_{k-1}}, \tag{75}$$

$$l_k(s) = \frac{s - t_{k+1}}{t_k - t_{k+1}}\frac{s - t_{k-1}}{t_k - t_{k-1}}, \tag{76}$$

$$l_{k-1}(s) = \frac{s - t_{k+1}}{t_{k-1} - t_{k+1}}\frac{s - t_k}{t_{k-1} - t_k}. \tag{77}$$

We approximate the derivative of $\boldsymbol{y}(s)$ at t_{k+1} with the derivative of the polynomial $\Pi_1\boldsymbol{y}(s)$ at t_{k+1}, i.e.,

$$\frac{d\boldsymbol{y}(t_{k+1})}{dt} \simeq \frac{d\Pi_2\boldsymbol{y}(t_{k+1})}{dt} = \frac{3\boldsymbol{y}_{k+1} - 4\boldsymbol{y}_k + \boldsymbol{y}_{k-1}}{2\Delta t}. \tag{78}$$

Substituting this approximation into the exact equation

$$\frac{d\boldsymbol{y}(t_{k+1})}{dt} = \boldsymbol{f}(\boldsymbol{y}(t_{k+1}), t_{k+1}) \tag{79}$$

yields the scheme

$$\frac{3}{2}\boldsymbol{u}_{k+1} - 2\boldsymbol{u}_k - \frac{1}{2}\boldsymbol{u}_{k-1} = \Delta t\boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}). \tag{80}$$

- **Three-step BDF method (BDF3):** By following a similar procedure as in BDF2 it is straightforward to show that

$$\frac{11}{6}\boldsymbol{u}_{k+1} - 3\boldsymbol{u}_k + \frac{3}{2}\boldsymbol{u}_{k-1} - \frac{1}{3}\boldsymbol{u}_{k-2} = \Delta t\boldsymbol{f}(\boldsymbol{u}_{k+1}, t_{k+1}). \tag{81}$$

Linear multistep methods (LMM)

The general form of a linear multistep method is

$$\sum_{j=0}^{q} \alpha_j \boldsymbol{u}_{k+j} = \Delta t \sum_{j=0}^{q} \beta_j \underbrace{\boldsymbol{f}(\boldsymbol{u}_{k+j}, t_{k+j})}_{\boldsymbol{f}_{k+j}}. \tag{82}$$

Note that Adams-Bashforth, Adams-Moulton and BDF methods are all in the form (82). To avoid non-uniqueness of coefficients due to rescaling we set $\alpha_q = 1$. Clearly, if $\beta_q = 0$ and then the method is *explicit*. On the other hand, if $\beta_q \neq 0$ then the method is *implicit*. Let us provide a few examples.

Example: The AB3 method

$$\boldsymbol{u}_{k+3} = \boldsymbol{u}_{k+2} + \frac{\Delta t}{12}\left(23\boldsymbol{f}_{k+2} - 16\boldsymbol{f}_{k+1} + 5\boldsymbol{f}_k\right) \tag{83}$$

can be written in the form (82) by setting

$$\alpha_3 = 1 \quad \alpha_2 = -1, \quad \alpha_1 = 0, \qquad \alpha_0 = 0,$$

$$\beta_3 = 0, \quad \beta_2 = \frac{23}{12}, \quad \beta_1 = -\frac{16}{12}, \quad \beta_0 = \frac{5}{12}.$$

Note that $(\alpha_3, \beta_3) = (1, 0)$ (the method is explicit) and

$$\sum_{j=0}^{3} \beta_j = 1. \tag{84}$$

Example: The BDF2 method

$$\boldsymbol{u}_{k+2} - \frac{4}{3}\boldsymbol{u}_{k+1} + \frac{1}{3}\boldsymbol{u}_k = \frac{2}{3}\Delta t\boldsymbol{f}_{k+2} \tag{85}$$

can be written in the form (82) by setting

$$\alpha_2 = 1, \qquad \alpha_1 = -\frac{4}{3}, \qquad \alpha_0 = \frac{1}{3}, \qquad \beta_2 = \frac{2}{3}, \qquad \beta_1 = 0, \qquad \beta_0 = 0. \tag{86}$$

Note that $(\alpha_2, \beta_2) = (1, 1/3)$ (the method is implicit) and

$$\sum_{j=0}^{2} \alpha_j = 0, \qquad \sum_{j=0}^{2} (\beta_j - j\alpha_j) = 0. \tag{87}$$

As we will see the conditions (84) and (87) guarantee that AB3 and BDF2 are *consistent* methods, i.e., that the truncation error of these methods goes to zero as we send Δt to zero. Roughly speaking this means that the numerical schemes (83) and (85) converge to the ODE (1) as $\Delta t \to 0$. This is necessary but not sufficient for the numerical solution generated by the scheme to converge to the analytical solution. The other element that is needed for convergence is *zero-stability*. The consistency conditions for a general linear q-step method are

$$\sum_{j=0}^{q} \alpha_j = 0, \qquad \sum_{j=0}^{q} (\beta_j - j\alpha_j) = 0. \tag{88}$$

Remark: All linear multistep methods rely on a polynomial interpolation process on an evenly-spaced temporal grid. As we known, polynomial interpolation on evenly spaced grids is, in general, ill-conditioned and can undergo a severe Runge's phenomenon depending on the function we are interpolating. However, the interpolation process of a function with a polynomial of fixed degree, say $q+1$, within a small a time interval (equal to $q\Delta t$ for a q-step method) is convergent as Δt goes to zero. To show this, recall the following error estimate we obtained in course note 1 (equation (23))

$$\|g - \Pi_q g\|_\infty \leq \Delta t^{q+1} \frac{\left\|g^{(q+1)}\right\|_\infty}{(q+1)}, \tag{89}$$

where $g(t)$ is any function of class C^{q+1} in $[t - q\Delta t, t]$ and $\Pi_q g(t)$ is a polynomial of degree q that interpolates $g(t)$ at $\{t, t - \Delta t, \ldots, t - q\Delta t\}$. The upper bound in (89) goes to zero as we send Δt to zero.

Runge-Kutta methods

Runge-Kutta (RK) methods are one-step methods (implicit or explicit) that aim at increasing accuracy by increasing the number of function evaluations within each time step. The general form of a RK method with s stages is

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \sum_{i=1}^{s} b_i \boldsymbol{K}_i \tag{90}$$

where

$$\boldsymbol{K}_i = \boldsymbol{f}\left(\boldsymbol{u}_k + \Delta t \sum_{j=1}^{s} a_{ij} \boldsymbol{K}_j, t_k + c_i \Delta t\right). \tag{91}$$

The coefficients of the RK method are usually collected in a table called *Butcher array*

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

The elements a_{ij} in the Butcher array can be positive or negative. Constency of RK methods, i.e., the fact that (90) converges to (1) as $\Delta t \to 0$ implies the following condition

$$\sum_{j=1}^{s} b_i = 1. \tag{92}$$

Moreover, it is usually assumed that

$$c_i = \sum_{j=1}^{s} a_{ij}. \tag{93}$$

Such a "row-sum condition" is not needed for a consistent method. If $a_{ij} = 0$ for $i \geq j$ then each $\boldsymbol{K}_i$ can be computed recursively from the previous ones and the RK method is *explicit*. Otherwise the RK method is implicit. Let us provide a few examples of RK methods.

- **Euler forward method:** The Euler forward method (14) can be seen as a one-step explicit RK method. The Butcher array corresponding to such explicit RK1 method is

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

- **Heun method:** The Heun mehod (28) is a two-stage explicit Runge-Kutta method. The Butcher array for the Heun method is:

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

  This table corresponds to the following RK2 method

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2} \left(\boldsymbol{K}_1 + \boldsymbol{K}_2 \right), \tag{94}$$

  where

$$\boldsymbol{K}_1 = \boldsymbol{f} \left(\boldsymbol{u}_k, t_k \right), \tag{95}$$
$$\boldsymbol{K}_2 = \boldsymbol{f} \left(\boldsymbol{u}_k + \Delta t \boldsymbol{f}(\boldsymbol{u}_k, t_k), t_k + \Delta t \right). \tag{96}$$

- **Crank-Nicolson method:** The Crank-Nicolson (CN) mehod (16) is a two-stage implicit Runge-Kutta method. The Butcher array corresponding to such method is:

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

  This table corresponds to the following RK2 method

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2} \left(\boldsymbol{K}_1 + \boldsymbol{K}_2 \right), \tag{97}$$

  where

$$\boldsymbol{K}_1 = \boldsymbol{f} \left(\boldsymbol{u}_k, t_k \right), \tag{98}$$
$$\boldsymbol{K}_2 = \boldsymbol{f} \left(\boldsymbol{u}_k + \frac{\Delta t}{2} \left(\boldsymbol{K}_1 + \boldsymbol{K}_2 \right), t_{k+1} \right). \tag{99}$$

From equation (97) we see that

$$\frac{\Delta t}{2}\left(\boldsymbol{K}_1+\boldsymbol{K}_2\right)=\boldsymbol{u}_{k+1}-\boldsymbol{u}_k. \tag{100}$$

Substituting this expression into (99) yields

$$\boldsymbol{K}_1=\boldsymbol{f}\left(\boldsymbol{u}_k,t_k\right),\qquad \boldsymbol{K}_2=\boldsymbol{f}\left(\boldsymbol{u}_{k+1},t_{k+1}\right). \tag{101}$$

- **Kutta's method**: This method is an explicit 3-stage method corresponding to the Butcher array:

0	0	0	0
1/2	1/2	0	0
1	−1	2	0
	1/6	2/3	1/6

The method can be written as

$$\boldsymbol{K}_1=\boldsymbol{f}(\boldsymbol{u}_k,t_k), \tag{102}$$
$$\boldsymbol{K}_2=\boldsymbol{f}\left(\boldsymbol{u}_k+\frac{\Delta t}{2}\boldsymbol{K}_1,t_k+\frac{\Delta t}{2}\right), \tag{103}$$
$$\boldsymbol{K}_3=\boldsymbol{f}\left(\boldsymbol{u}_k-\Delta t\boldsymbol{K}_1+2\Delta t\boldsymbol{K}_2,t_k+\Delta t\right), \tag{104}$$

$$\boldsymbol{u}_{k+1}=\boldsymbol{u}_k+\frac{\Delta t}{6}\left(\boldsymbol{K}_1+4\boldsymbol{K}_2+\boldsymbol{K}_3\right). \tag{105}$$

- **Runge-Kutta method (RK4):** The most famous RK method is perhaps the one proposed in the original paper by Runge and Kutta. The scheme is 4-stage explicit and can be written as

$$\boldsymbol{K}_1=\boldsymbol{f}(\boldsymbol{u}_k,t_k), \tag{106}$$
$$\boldsymbol{K}_2=\boldsymbol{f}\left(\boldsymbol{u}_k+\frac{\Delta t}{2}\boldsymbol{K}_1,t_k+\frac{\Delta t}{2}\right), \tag{107}$$
$$\boldsymbol{K}_3=\boldsymbol{f}\left(\boldsymbol{u}_k+\frac{\Delta t}{2}\boldsymbol{K}_2,t_k+\frac{\Delta t}{2}\right), \tag{108}$$
$$\boldsymbol{K}_4=\boldsymbol{f}\left(\boldsymbol{u}_k+\Delta t\boldsymbol{K}_3,t_k+\Delta t\right), \tag{109}$$

$$\boldsymbol{u}_{k+1}=\boldsymbol{u}_k+\frac{\Delta t}{6}\left(\boldsymbol{K}_1+2\boldsymbol{K}_2+2\boldsymbol{K}_3+\boldsymbol{K}_4\right). \tag{110}$$

The Butcher array for this scheme is

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	1/3	1/3	1/6

Derivation of explicit RK2 methods. Let us show how to derive an arbitrary explicit two-stage RK method. To this end we first write the Butcher array:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ c & c & 0 \\ \hline & b_1 & b_2 \end{array}$$

The RK2 method corresponding to this table can be written explicitly as

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t(b_1 \boldsymbol{K}_1 + b_2 \boldsymbol{K}_2) \tag{111}$$

where

$$\boldsymbol{K}_1 = \boldsymbol{f}(\boldsymbol{u}_k, t_k), \tag{112}$$
$$\boldsymbol{K}_2 = \boldsymbol{f}(\boldsymbol{u}_k + c\Delta t \boldsymbol{K}_1, t_k + c\Delta t). \tag{113}$$

Expand $\boldsymbol{K}_2$ in a Taylor series in Δt to obtain

$$\boldsymbol{K}_2 = \boldsymbol{K}_1 + c\Delta t \left(\sum_{j=1}^{n} \frac{\partial \boldsymbol{f}}{\partial y_j} K_{1j} + \frac{\partial \boldsymbol{f}}{\partial t} \right) + \cdots . \tag{114}$$

A substitution of this expression into (111) yields

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t(b_1 + b_2)\boldsymbol{f}(\boldsymbol{u}_k, t_k) + b_2 c(\Delta t)^2 \left(\sum_{j=1}^{n} \frac{\partial \boldsymbol{f}(\boldsymbol{u}_k, t_k)}{\partial y_j} f_j(\boldsymbol{u}_k, t_k) + \frac{\partial \boldsymbol{f}(\boldsymbol{u}_k, t_k)}{\partial t} \right) + \cdots . \tag{115}$$

Next, we expand the solution to the ODE (1) in a Taylor series

$$\boldsymbol{y}(t_{k+1}) = \boldsymbol{y}(t_k) + \frac{d\boldsymbol{y}(t_k)}{dt}\Delta t + \frac{(\Delta t)^2}{2}\frac{d^2\boldsymbol{y}(t_k)}{dt^2} + \cdots . \tag{116}$$

By using (1) and the chain rule we obtain:

$$\frac{d\boldsymbol{y}(t_k)}{dt} = \boldsymbol{f}(\boldsymbol{y}_k, t_k), \qquad \frac{d^2\boldsymbol{y}(t_k)}{dt^2} = \sum_{j=1}^{n} \frac{\partial \boldsymbol{f}(\boldsymbol{y}_k, t_k)}{\partial y_j} f_j(\boldsymbol{y}_k, t_k) + \frac{\partial \boldsymbol{f}(\boldsymbol{y}_k, t_k)}{\partial t}. \tag{117}$$

Assuming that $\boldsymbol{y}_k = \boldsymbol{u}_k$ and matching the terms multiplying the same powers of Δt in (115) and (116) we obtain[3]

$$b_1 + b_2 = 1 \qquad \text{and} \qquad c b_2 = \frac{1}{2}. \tag{118}$$

This is a system of 2 equations in 3 unknowns. Thus, there is an *infinite number* of explicit (and consistent) RK2 methods. For example, setting

$$c = 1, \qquad b_1 = b_2 = \frac{1}{2} \qquad \text{yields the Heun method (28).} \tag{119}$$

On the other hand, setting

$$c = \frac{1}{2}, \quad b_1 = 0 \quad b_2 = 1 \qquad \text{yields the explicit midpoint method (31).} \tag{120}$$

[3]The condition $b_1 + b_2 = 1$ is a consistency condition which guarantees that the scheme (111) converges to the ODE (1) in the limit $\Delta t \to 0$.

Derivation of higher order explicit RK methods. By using the Taylor series approach discussed in previous section, it is possible to derive conditions on the entries of the Butcher array for RK methods with an arbitrary number of stages. Essentially, we perform a Taylor series of the RK method in Δt and then match it with the Taylor series expansion of the solution up to a given order. The corresponding equations for the coefficients, e.g., (118) are called *stage-order* conditions. This is discussed, e.g., in [2, §5.9]. For example, it can be shown that the stage-order conditions for a three stage RK method defined by the Butcher array

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ c_2 & a_{21} & 0 & 0 \\ c_3 & a_{31} & a_{32} & 0 \\ \hline & b_1 & b_2 & b_3 \end{array}$$

are

$$\begin{cases} b_1 + b_2 + b_3 = 1 \\ b_2 c_2 + b_3 c_3 = \dfrac{1}{2} \\ b_2 c_2^2 + b_3 c_3^2 = \dfrac{1}{3} \\ b_3 a_{32} c_2 = \dfrac{1}{6} \end{cases} \tag{121}$$

which yields to one two-parameter family of solutions and two one-parameter families of solutions [2, p. 178]. Hereafter we list a few three-stage explicit RK methods satisfying (121):

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 \\ 2/3 & 0 & 2/3 & 0 \\ \hline & 1/4 & 0 & 3/4 \end{array} \qquad \text{Heun's method}$$

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 8/15 & 8/15 & 0 & 0 \\ 2/3 & 1/4 & 5/12 & 0 \\ \hline & 1/4 & 0 & 3/4 \end{array} \qquad \text{Van der Houwen's/Wray method}$$

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 3/4 & 0 & 3/4 & 0 \\ \hline & 2/9 & 1/3 & 4/9 \end{array} \qquad \text{Ralson's method}$$

The Taylor series approach rapidly become intractable as the number of stages increases, and so does the corresponding stage-order conditions. Fortunately, there is a more efficient way to derive conditions such as (118) using *rooted trees* (see [2, §5.6]).

Derivation of implicit RK methods. Implicit RK methods can be derived from the integral formulation (10) of the Cauchy problem. In particular, consider a Gauss quadrature formula with s nodes in $[t_k, t_{k+1}]$ to approximate the integral at the right hand side of (10),

$$\int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{y}(s), s) ds \simeq \Delta t \sum_{j=1}^{s} b_j \boldsymbol{f}\left(\boldsymbol{y}(t_k + c_j \Delta t), t_k + c_j \Delta t\right). \tag{122}$$

Here, we denoted by b_j the quadrature weights and by $t_k + c_j \Delta t$ the quadrature nodes. It can be shown that for any RK scheme of the form (90)-(91) there exists a correspondence between the coefficients b_j and c_j of the formula (122) and the weights and nodes of a Gauss quadrature rule (see, [2, §5.11] for details). For instance, the implicit midpoint method can be seen as a one-step implicit RK method based on a 1-point Gauss-Legendre quadrature rule (i.e., the midpoint of the interval $[t_k, t_{k+1}]$. This method corresponds to the Butcher array

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

and can attain order 2. In general, an implicit s-stage Gauss RK method can attain order $2s$. Similarly, Gauss-Radau and Gauss-Lobatto RK methods can achieve order $2s-1$ and $2s-2$, respectively.

Appendix A: Vector norms and matrix norms

In this appendix we briefly review the most common vector norms in $\mathbb{R}^n$ and their equivalence. We also discuss common matrix norms induced by vector norms.

Vector norms and their equivalence

As is well known, all norms defined in a finite-dimensional vector space (such as $\mathbb{R}^n$) are *equivalent*. This means that if we pick two arbitrary norms in $\mathbb{R}^n$, say $\|\cdot\|_a$ and $\|\cdot\|_b$, then there exist two numbers c_1 and c_2 such that

$$c_1 \|\boldsymbol{y}\|_a \leq \|\boldsymbol{y}\|_b \leq c_2 \|\boldsymbol{y}\|_a \quad \text{for all} \quad \boldsymbol{y} \in \mathbb{R}^n. \tag{123}$$

The most common norms in $\mathbb{R}^n$ are

$$\|\boldsymbol{y}\|_\infty = \max_{k=1,..,n} |y_k| \,, \tag{124}$$

$$\|\boldsymbol{y}\|_1 = \sum_{k=1}^{n} |y_k| \,, \tag{125}$$

$$\|\boldsymbol{y}\|_2 = \left(\sum_{k=1}^{n} |y_k|^2 \right)^{1/2} , \tag{126}$$

$$\vdots \tag{127}$$

$$\|\boldsymbol{y}\|_p = \left(\sum_{k=1}^{n} |y_k|^p \right)^{1/p} \qquad p \in \mathbb{N} \setminus \{\infty\}. \tag{128}$$

Based on these definitions it is easy to show, e.g., that

$$\|\boldsymbol{y}\|_\infty \leq \|\boldsymbol{y}\|_1 \leq n \|\boldsymbol{y}\|_\infty \,, \tag{129}$$

$$\|\boldsymbol{y}\|_2 \leq \|\boldsymbol{y}\|_1 \leq \sqrt{n} \|\boldsymbol{y}\|_2 \,, \tag{130}$$

$$\|\boldsymbol{y}\|_\infty \leq \|\boldsymbol{y}\|_2 \leq \sqrt{n} \|\boldsymbol{y}\|_\infty \,. \tag{131}$$

As an example, if $\boldsymbol{f}(\boldsymbol{y},t)$ in (1) is Lipschitz continuous in D with respect to the 1-norm, i.e.,

$$\|\boldsymbol{f}(\boldsymbol{y}_1,t) - \boldsymbol{f}(\boldsymbol{y}_2,t)\|_1 \leq L_1 \|\boldsymbol{y}_1 - \boldsymbol{y}_2\|_1 \quad \text{for all} \quad \boldsymbol{y}_1, \boldsymbol{y}_2 \in D, \quad \text{for all} \quad t \geq 0 \tag{132}$$

then it is also Lipschitz continuous with respect to the uniform norm. In fact, by using (129) we have

$$\|\boldsymbol{f}(\boldsymbol{y}_1,t) - \boldsymbol{f}(\boldsymbol{y}_2,t)\|_\infty \leq \underbrace{L_1 n}_{L_\infty} \|\boldsymbol{y}_1 - \boldsymbol{y}_2\|_\infty \,. \tag{133}$$

Of course, $\boldsymbol{f}(\boldsymbol{y},t)$ is also Lipschitz continuous with respect to the 2-norm.

Matrix norms induced by vector norms

Let us define the following matrix norm

$$\|\boldsymbol{A}\| = \sup_{\boldsymbol{y} \neq \boldsymbol{0}_{\mathbb{R}^n}} \frac{\|\boldsymbol{A}\boldsymbol{y}\|}{\|\boldsymbol{y}\|} = \sup_{\|\boldsymbol{y}\|=1} \|\boldsymbol{A}\boldsymbol{y}\| \,. \tag{134}$$

where the norm at the right hand side is any vector norm. Clearly, $\|\boldsymbol{A}\|$ is matrix norm (prove it as exercise), which satisfies, by definition, the following inequality

$$\|\boldsymbol{A}\| \geq \frac{\|\boldsymbol{A}\boldsymbol{y}\|}{\|\boldsymbol{y}\|} \qquad \text{i.e.} \qquad \|\boldsymbol{A}\boldsymbol{y}\| \leq \|\boldsymbol{A}\| \, \|\boldsymbol{y}\| \,. \tag{135}$$

It is straightforward to show that

$$\|\boldsymbol{A}\|_\infty = \max_{i=1,..,n} \left(\sum_{j=1}^{n} |A_{ij}| \right), \tag{136}$$

$$\|\boldsymbol{A}\|_1 = \max_{j=1,..,n} \left(\sum_{i=1}^{n} |A_{ij}| \right), \tag{137}$$

$$\|\boldsymbol{A}\|_2 = \sqrt{\lambda_{\max}\left(\boldsymbol{A}^T\boldsymbol{A}\right)} = \sigma_{\max}(\boldsymbol{A}), \tag{138}$$

where $\sigma_{\max}(\boldsymbol{A})$ is the largest singular value of the matrix $\boldsymbol{A}$. For example,

$$\|\boldsymbol{A}\boldsymbol{y}\|_\infty = \max_{i=1,\ldots,n} \left| \sum_{j=1}^{n} A_{ij} y_j \right| \leq \max_{i=1,\ldots,n} \left(\sum_{j=1}^{n} |A_{ij}|\,|y_j| \right) \leq \|\boldsymbol{y}\|_\infty \max_{i=1,\ldots,n} \left(\sum_{j=1}^{n} |A_{ij}| \right) \tag{139}$$

which implies that

$$\frac{\|\boldsymbol{A}\boldsymbol{y}\|_\infty}{\|\boldsymbol{y}\|_\infty} \leq \max_{i=1,\ldots,n} \left(\sum_{j=1}^{n} |A_{ij}| \right) \qquad \text{for all } \boldsymbol{y} \neq \boldsymbol{0}_{\mathbb{R}^n}, \tag{140}$$

i.e.,

$$\sup_{\boldsymbol{y}\neq\boldsymbol{0}_{\mathbb{R}^n}} \frac{\|\boldsymbol{A}\boldsymbol{y}\|_\infty}{\|\boldsymbol{y}\|_\infty} = \max_{i=1,\ldots,n} \left(\sum_{j=1}^{n} |A_{ij}| \right) = \|\boldsymbol{A}\|_\infty . \tag{141}$$

References

[1] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer, 2006.

[2] J. D. Lambert. *Numerical methods for ordinary differential systems: the initial value problem*. Wiley, 1991.

[3] R. LeVeque. *Finite difference methods for ordinary and partial differential equations*. SIAM, 2007.

[4] A. Quarteroni, R. Sacco, and F. Salieri. *Numerical mathematics*. Springer, 2007.