Part I – Numerical methods for ODEs

MAT684 – Spring 2026 – Shukai Du

Topics:

- Euler and trapezoidal method
- Runge-Kutta method
- Multistep method
- Stability
- Symplectic integrator

1 Euler method

Many systems can be formulated as differential equations:

$$\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}), \qquad t \geq t_0, \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0. \tag{1}$$

For instance, $\boldsymbol{y}$ can represent the position and momentum of planets or satellites in our solar system; the state of a molecule; the stress or strain in a mechanical structure (e.g., a car body); or the velocity field of a fluid (or air). Our goal is to predict the future state $\boldsymbol{y}(t)$ from the initial data $\boldsymbol{y}_0$.

Unfortunately, except for special choices of $\boldsymbol{f}$, (1) does not admit a closed-form solution. This motivates the study of numerical methods for approximating the solution of (1).

The following videos give a visual sense of what numerical simulation looks like in practice:

- Molecular dynamics
  https://www.youtube.com/watch?v=ckTqh50r_2w
- Finite element analysis
  https://www.youtube.com/watch?v=XMjqONEbYro
- Electromagnetic wave guide
  https://www.youtube.com/shorts/uPLohDR70RE
- Computational fluid dynamics
  https://www.youtube.com/watch?v=yM2rT3teakE
- Large-scale ODE systems
  https://www.youtube.com/watch?v=kRlhlCWplqk&t=32s

The first numerical approach we will see in this course is the *Euler method*. It is based on the following observation. By a Taylor expansion, we obtain

$$\begin{aligned}\boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \boldsymbol{y}'(t_0)(t - t_0) + \mathcal{O}\big((t - t_0)^2\big) \\ &= \boldsymbol{y}(t_0) + (t - t_0)\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) + \mathcal{O}\big((t - t_0)^2\big).\end{aligned}$$

Thus, $\boldsymbol{y}(t_0) + (t - t_0)\boldsymbol{f}(t_0, \boldsymbol{y}(t_0))$ can be used to predict $\boldsymbol{y}(t)$ when the step size $(t - t_0)$ is small and $\boldsymbol{y}$ is sufficiently smooth.

Since we are not only interested in predicting the near future. In another word, suppose we want to predict $\boldsymbol{y}(T)$ where $T - t_0$ can be large. In this case, $\boldsymbol{y}(t_0) + (T - t_0)\boldsymbol{f}(t_0, \boldsymbol{y}(t_0))$ may not be a good approximation to $\boldsymbol{y}(T)$. To overcome this difficulty, we divide the time interval $[t_0, T]$ into many small steps:

$$t_0 < t_1 < t_2 < \cdots < t_N = T,$$

where $t_i = t_0 + ih$ and h is the time step, chosen so that the prediction over each step is accurate enough. Namely,

$$\boldsymbol{y}(t_i) \approx \boldsymbol{y}(t_{i-1}) + h\, \boldsymbol{f}\big(t_{i-1}, \boldsymbol{y}(t_{i-1})\big) \qquad \text{for } i = 1, 2, \ldots, N.$$

Let $\boldsymbol{y}_i$ denote a numerical approximation to the true solution $\boldsymbol{y}(t_i)$. Then the Euler method is the recursive scheme

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\, \boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}) \qquad \text{for } i = 1, 2, \ldots, N. \tag{2}$$

Note that the initial state agrees with the exact solution $\boldsymbol{y}_0 = \boldsymbol{y}(t_0)$.

Example. Consider

$$\boldsymbol{y}' = \boldsymbol{y}(1 - \boldsymbol{y}), \qquad \boldsymbol{y}(0) = \frac{1}{10}.$$

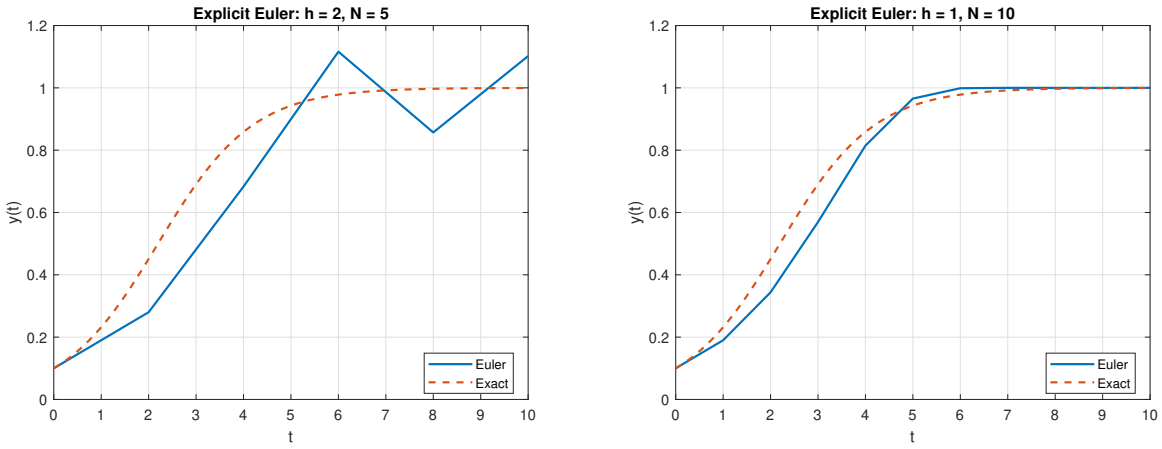
We want to predict the behavior of $\boldsymbol{y}$ on $[0, T]$ with $T = 10$.

- Matlab code

```
% Explicit Euler for y' = y(1 - y),  y(0) = 1/10
clear; clc;
close all

% Parameters
y0 = 1/10;
T  = 10;        % final time
N  = 10;        % number of steps
h  = T/N;       % step size

% Grid and storage
t = linspace(0, T, N+1).';
t_exact = linspace(0, T, 1000).';
y = zeros(N+1, 1);
y(1) = y0;

% Euler iteration: y_{n+1} = y_n + h * f(t_n, y_n)
for n = 1:N
    y(n+1) = y(n) + h * ( y(n) * (1 - y(n)) );
end

y_exact = 1 ./ (1 + 9*exp(-t_exact));

% Plot
```

```
figure;
plot(t, y, 'LineWidth', 1.5); hold on;
plot(t_exact, y_exact, '--', 'LineWidth', 1.5);
grid on;
xlabel('t'); ylabel('y(t)');
ylim([0,1.2])
legend('Euler', 'Exact', 'Location', 'best');
title(sprintf('Explicit Euler: h = %.4g, N = %d', h, N));
```

- Results



From this example, we observe that as we increase the number of time step N (or decrease the step size h), Euler method provides a better approximation to the exact solution. We will next prove that Euler method is convergent, or more precisely, first order convergent.

Theorem 1.1. *The Euler method* (2) *is first-order convergent.*

Proof. We assume that $\boldsymbol{f}$ is sufficiently smooth so that the exact solution $\boldsymbol{y}$ is also smooth and admits a well-defined Taylor expansion. In addition, we assume that $\boldsymbol{f}$ is Lipschitz continuous in its second argument: there exists a constant $\lambda > 0$ such that

$$\|\boldsymbol{f}(t, \boldsymbol{x}) - \boldsymbol{f}(t, \boldsymbol{y})\| \leq \lambda \|\boldsymbol{x} - \boldsymbol{y}\|.$$

We denote by $\boldsymbol{e}_i := \boldsymbol{y}_i - \boldsymbol{y}(t_i)$ the error at time t_i. Our goal is to show that $\|\boldsymbol{e}_i\| \to 0$ for all $i = 1, 2, \ldots, N$ as the step size $h \to 0$, and in fact that the error is $\mathcal{O}(h)$.

By Taylor expansion of the exact solution, we have

$$\boldsymbol{y}(t_i) = \boldsymbol{y}(t_{i-1}) + h\boldsymbol{f}(t_{i-1}, \boldsymbol{y}(t_{i-1})) + \mathcal{O}(h^2).$$

The Euler method gives

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\, \boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}).$$

Taking the difference between the above two equations, we obtain

$$\boldsymbol{e}_i = \boldsymbol{e}_{i-1} + h\left(\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}) - \boldsymbol{f}(t_{i-1}, \boldsymbol{y}(t_{i-1}))\right) + \mathcal{O}(h^2).$$

Therefore, by the triangle inequality and the Lipschitz property,

$$\begin{aligned}\|\boldsymbol{e}_i\| &\le \|\boldsymbol{e}_{i-1}\| + h\big\|\boldsymbol{f}(t_{i-1},\boldsymbol{y}_{i-1}) - \boldsymbol{f}(t_{i-1},\boldsymbol{y}(t_{i-1}))\big\| + ch^2\\ &\le (1+h\lambda)\|\boldsymbol{e}_{i-1}\| + ch^2, \qquad i=1,2,\dots,N,\end{aligned}$$

where $c>0$ is independent of h and i.

Since $\boldsymbol{e}_0 = \boldsymbol{0}$, iterating the above inequality yields

$$\|\boldsymbol{e}_i\| \le ch^2 \sum_{k=0}^{i-1}(1+h\lambda)^k = \frac{c}{\lambda}h\big[(1+h\lambda)^i - 1\big].$$

Finally, using the standard estimate $1+h\lambda \le e^{h\lambda}$, we obtain

$$(1+h\lambda)^i \le e^{ih\lambda} \le e^{T\lambda},$$

and hence

$$\|\boldsymbol{e}_i\| \le \frac{c}{\lambda}h(e^{\lambda T}-1), \qquad i=1,2,\dots,N.$$

This shows that $\max_{1\le i\le N}\|\boldsymbol{e}_i\| = \mathcal{O}(h)$, i.e., the Euler method is first-order convergent. □

Local truncation error and the order of a method. A key step in the proof is to derive a recurrence for the error at step i:

$$\boldsymbol{e}_i = \boldsymbol{e}_{i-1} + h\left(\boldsymbol{f}(t_{i-1},\boldsymbol{y}_{i-1}) - \boldsymbol{f}(t_{i-1},\boldsymbol{y}(t_{i-1}))\right) + \mathcal{O}(h^2).$$

The term $\mathcal{O}(h^2)$ plays an important role in deriving the convergence $\|\boldsymbol{e}_i\| \to 0$.

Note that the $\mathcal{O}(h^2)$ term can be obtained by replacing the Euler method's state variable $\boldsymbol{y}_i$ by the exact solution $\boldsymbol{y}(t_i)$ and examining the *one-step residual*:

$$\boldsymbol{y}(t_i) - \boldsymbol{y}(t_{i-1}) - h\boldsymbol{f}(t_{i-1},\boldsymbol{y}(t_{i-1})) = \mathcal{O}(h^2).$$

We call this one-step residual the *local truncation error*.

In general, if the local truncation error (i.e., the one-step residual) is $\mathcal{O}(h^{p+1})$, then we say the method is of order p.

2 Trapezoidal rule

Consider again the differential equation

$$\boldsymbol{y}' = \boldsymbol{f}(t,\boldsymbol{y}). \tag{3}$$

We focus on a single time-step interval $[t_{i-1},t_i]$. Integrating from t_{i-1} to t_i yields

$$\begin{aligned}\boldsymbol{y}(t_i) &= \boldsymbol{y}(t_{i-1}) + \int_{t_{i-1}}^{t_i} \boldsymbol{f}(\tau,\boldsymbol{y}(\tau))\,\mathrm{d}\tau\\ &\approx \boldsymbol{y}(t_{i-1}) + \boldsymbol{f}(t_{i-1},\boldsymbol{y}(t_{i-1}))(t_i - t_{i-1}).\end{aligned}$$

Thus, the Euler method approximates the time integral using only the left-endpoint value $\boldsymbol{f}(t_{i-1}, \boldsymbol{y}(t_{i-1}))$, but it discards the information at the right endpoint $\boldsymbol{f}(t_i, \boldsymbol{y}(t_i))$.

In MAT 683, we learned that numerical integration can be made more accurate by using both endpoint values. Namely,

$$\begin{aligned}\boldsymbol{y}(t_i) &= \boldsymbol{y}(t_{i-1}) + \int_{t_{i-1}}^{t_i} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))\,\mathrm{d}\tau \\ &\approx \boldsymbol{y}(t_{i-1}) + \frac{1}{2}\left(\boldsymbol{f}(t_{i-1}, \boldsymbol{y}(t_{i-1})) + \boldsymbol{f}(t_i, \boldsymbol{y}(t_i))\right)(t_i - t_{i-1}).\end{aligned}$$

This motivates the trapezoidal rule:

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + \frac{1}{2}h\left(\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}) + \boldsymbol{f}(t_i, \boldsymbol{y}_i)\right). \tag{4}$$

Theorem 2.1. *The trapezoidal rule (4) is second-order convergent.*

Proof. The proof is similar to that of the Euler method (Theorem 1.1), so we only highlight the step that differs. We show that the local truncation error is $\mathcal{O}(h^3)$; it then follows that the global error is uniformly bounded by $\mathcal{O}(h^2)$. (Recall that for the Euler method, the local truncation error is $\mathcal{O}(h^2)$ and the global error is $\mathcal{O}(h)$.)

To this end, we substitute the exact solution into (4) and estimate the residual:

$$\begin{aligned}&\boldsymbol{y}(t_i) - \left[\boldsymbol{y}(t_{i-1}) + \frac{1}{2}h\left(\boldsymbol{f}(t_{i-1}, \boldsymbol{y}(t_{i-1})) + \boldsymbol{f}(t_i, \boldsymbol{y}(t_i))\right)\right] \\ &= \boldsymbol{y}(t_i) - \boldsymbol{y}(t_{i-1}) - \frac{1}{2}h\left(\boldsymbol{y}'(t_{i-1}) + \boldsymbol{y}'(t_i)\right).\end{aligned}$$

Using Taylor expansions about t_{i-1},

$$\begin{aligned}\boldsymbol{y}(t_i) &= \boldsymbol{y}(t_{i-1}) + \boldsymbol{y}'(t_{i-1})h + \frac{1}{2}\boldsymbol{y}''(t_{i-1})h^2 + \mathcal{O}(h^3), \\ \boldsymbol{y}'(t_i) &= \boldsymbol{y}'(t_{i-1}) + \boldsymbol{y}''(t_{i-1})h + \mathcal{O}(h^2).\end{aligned}$$

Substituting $\boldsymbol{y}''(t_{i-1})$ from the first expansion using the second gives

$$\boldsymbol{y}(t_i) = \boldsymbol{y}(t_{i-1}) + \boldsymbol{y}'(t_{i-1})h + \frac{1}{2}h\left(\boldsymbol{y}'(t_i) - \boldsymbol{y}'(t_{i-1})\right) + \mathcal{O}(h^3).$$

Thus, the local truncation error is $\mathcal{O}(h^3)$. This completes the proof. □

Example. Consider again

$$\boldsymbol{y}' = \boldsymbol{y}(1 - \boldsymbol{y}), \qquad \boldsymbol{y}(0) = \frac{1}{10}.$$

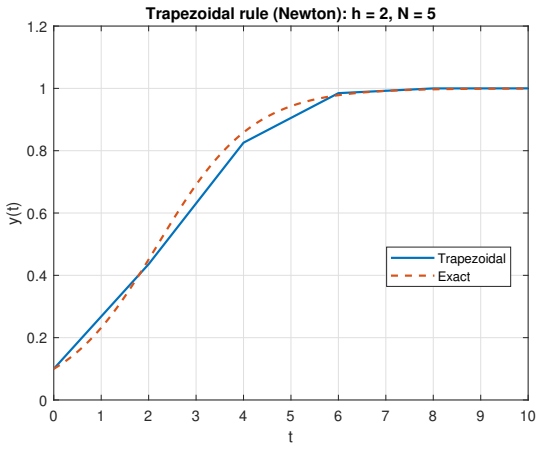
We want to predict the behavior of $\boldsymbol{y}$ on $[0, T]$ with $T = 10$.

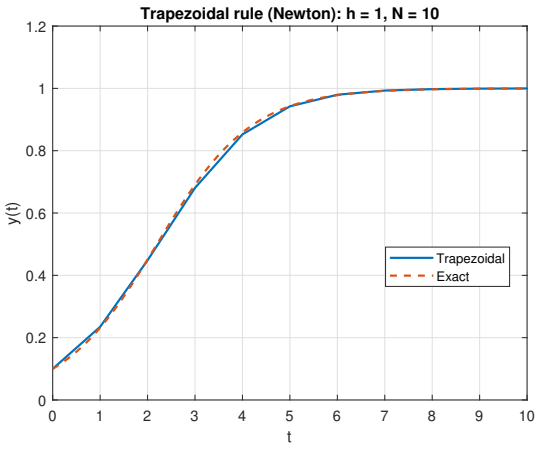
- Matlab code of trapezoidal rule

```
% Trapezoidal rule (implicit) for y' = y(1 - y),  y(0) = 1/10
clear; clc;
close all

% Parameters
y0 = 1/10;
T  = 10;        % final time
N  = 10;        % number of steps
h  = T/N;       % step size

% Grid and storage
t = linspace(0, T, N+1).';
t_exact = linspace(0, T, 1000).';
y = zeros(N+1, 1);
y(1) = y0;

% f(y) and f'(y)
f  = @(z) z .* (1 - z);
df = @(z) 1 - 2*z;

% Newton parameters
tol = 1e-12;
maxit = 20;

% Trapezoidal iteration
for n = 1:N
    yn = y(n);

    % Initial guess: Euler step
    ynew = yn + h * f(yn);

    % Newton iteration
    for k = 1:maxit
        G  = ynew - yn - (h/2)*( f(yn) + f(ynew) );
        dG = 1 - (h/2)*df(ynew);

        delta = -G / dG;
        ynew  = ynew + delta;

        if abs(delta) < tol
            break;
        end
    end

    y(n+1) = ynew;
end

% Exact solution
y_exact = 1 ./ (1 + 9*exp(-t_exact));

% Plot
figure;
plot(t, y, 'LineWidth', 1.5); hold on;
plot(t_exact, y_exact, '--', 'LineWidth', 1.5);
```

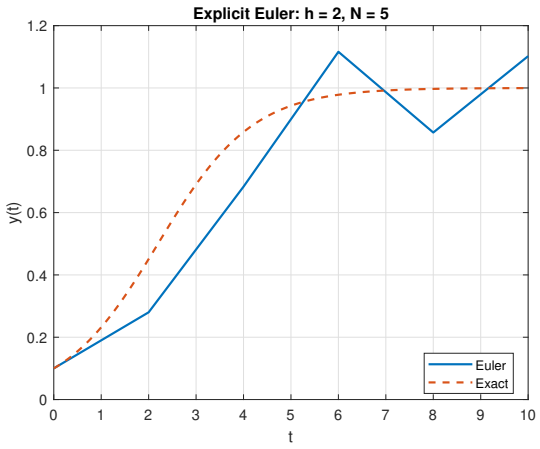
```
grid on;
xlabel('t'); ylabel('y(t)');
ylim([0,1.2])
legend('Trapezoidal', 'Exact', 'Location', 'best');
title(sprintf('Trapezoidal rule (Newton): h = %.4g, N = %d', h, N));
```

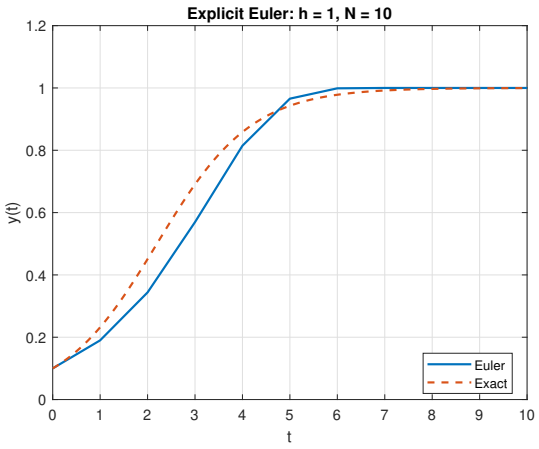
- Results by Trapezoidal method:





To compare with the following results by Euler method:





Compared with the results by Euler method, Trapezoidal rule gives more accurate approximation to the exact solution.

Remark 2.1 (Convergence depends on smoothness)**.** *For both the Euler method and the trapezoidal rule, convergence requires sufficient smoothness of the vector field* $\boldsymbol{f}$ *and of the exact solution* $\boldsymbol{y}$*. In particular, we assume that* $\boldsymbol{f}(t, \boldsymbol{y})$ *is Lipschitz continuous in* $\boldsymbol{y}$*, and that* $\boldsymbol{y}$ *is sufficiently smooth in time so that the remainder terms in the Taylor expansions are controlled by* $\mathcal{O}(h^2)$ *(Euler) and* $\mathcal{O}(h^3)$ *(trapezoidal). Since the smoothness of* $\boldsymbol{y}$ *is inherited from that of* $\boldsymbol{f}$ *through the ODE, the convergence of these methods ultimately depends on the smoothness of* $\boldsymbol{f}$*. If* $\boldsymbol{f}$ *(and hence* $\boldsymbol{y}$*) lacks the required smoothness, the expected convergence order may fail, and the method may even fail to converge.*

Implicit and explicit methods. There is a fundamental difference between the Euler method and the trapezoidal rule. Recall that

$$\text{(Euler)} \qquad \boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\,\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}),$$
$$\text{(Trapezoidal)} \qquad \boldsymbol{y}_i = \boldsymbol{y}_{i-1} + \frac{h}{2}\left(\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}) + \boldsymbol{f}(t_i, \boldsymbol{y}_i)\right).$$

Given the state at step $i-1$, i.e., $\boldsymbol{y}_{i-1}$, the Euler update can be computed directly by evaluating $\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1})$. In contrast, the trapezoidal rule is *implicit* in $\boldsymbol{y}_i$: at each time step one must solve the equation

$$\boldsymbol{y}_i - \frac{h}{2}\,\boldsymbol{f}(t_i, \boldsymbol{y}_i) = \boldsymbol{y}_{i-1} + \frac{h}{2}\,\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}).$$

Accordingly, explicit methods compute $\boldsymbol{y}_i$ from already-known quantities (no equation needs to be solved), whereas implicit methods determine $\boldsymbol{y}_i$ by solving an equation at each step.

We can also modify the Euler method by evaluating the right-hand side at the new time level. This yields

$$\text{(Implicit Euler)} \qquad \boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\,\boldsymbol{f}(t_i, \boldsymbol{y}_i),$$

which is called *implicit Euler* because $\boldsymbol{y}_i$ appears on both sides and must be found by solving an equation. The original Euler method is therefore often called *explicit Euler*.

In practice, such equations are commonly solved by a nonlinear solver such as Newton's method when $\boldsymbol{f}$ is a nonlinear function of $\boldsymbol{y}$. Because solving an equation is typically more expensive than a single evaluation of $\boldsymbol{f}$, implicit methods usually cost more per time step than explicit methods.

3 Runge-Kutta methods

So far, we have learned two methods:

$$\text{(Euler)} \qquad \boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\,\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}),$$
$$\text{(Trapezoidal)} \qquad \boldsymbol{y}_i = \boldsymbol{y}_{i-1} + \frac{1}{2}h\left(\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}) + \boldsymbol{f}(t_i, \boldsymbol{y}_i)\right).$$

The Euler method is first-order accurate and explicit, while the trapezoidal method is second-order accurate but implicit. Since implicit methods require solving (generally nonlinear) algebraic equations at each step, when this solve is expensive (e.g., nonlinear or high-dimensional problems), we would like to avoid implicit methods.

Can we have a higher-order method (second order or higher) that is explicit? In this section, we will cover several important high-order explicit methods. They belong to the class of *explicit Runge–Kutta methods*.

To derive a general Runge-Kutta method, we proceed as follows. Integrating the differential equation (1) over $[t_n, t_n + h]$ gives

$$\begin{aligned}\boldsymbol{y}(t_n + h) &= \boldsymbol{y}(t_n) + \int_{t_n}^{t_n+h} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))\,\mathrm{d}\tau \\ &= \boldsymbol{y}(t_n) + h\int_0^1 \boldsymbol{f}(t_n + h\tau, \boldsymbol{y}(t_n + h\tau))\,\mathrm{d}\tau.\end{aligned}$$

We approximate the time integral by a numerical quadrature rule:

$$\boldsymbol{y}(t_n + h) \approx \boldsymbol{y}(t_n) + h \sum_{j=1}^{\nu} b_j \, \boldsymbol{f}(t_n + c_j h, \boldsymbol{y}(t_n + c_j h)),$$

where b_j are the quadrature weights and c_j are the quadrature nodes.

Unfortunately, we do not know the intermediate states $\boldsymbol{y}(t_n + c_j h)$, so we need to approximate them. Let $\boldsymbol{\xi}_j$ denote an approximation to $\boldsymbol{y}(t_n + c_j h)$. Given the current state $\boldsymbol{y}_n \approx \boldsymbol{y}(t_n)$, we aim to compute $\boldsymbol{y}_{n+1} \approx \boldsymbol{y}(t_n + h)$. An explicit Runge–Kutta method updates the stage values $\boldsymbol{\xi}_j$ sequentially and then computes $\boldsymbol{y}_{n+1}$ by

$$\begin{aligned}
&\text{(stage 1)} && \boldsymbol{\xi}_1 = \boldsymbol{y}_n \\
&\text{(stage 2)} && \boldsymbol{\xi}_2 = \boldsymbol{y}_n + h a_{21} \boldsymbol{f}(t_n + c_1 h, \boldsymbol{\xi}_1) \\
&\text{(stage 3)} && \boldsymbol{\xi}_3 = \boldsymbol{y}_n + h a_{31} \boldsymbol{f}(t_n + c_1 h, \boldsymbol{\xi}_1) + h a_{32} \boldsymbol{f}(t_n + c_2 h, \boldsymbol{\xi}_2) \\
&\text{(stage } \nu\text{)} && \boldsymbol{\xi}_\nu = \boldsymbol{y}_n + h \sum_{i=1}^{\nu-1} a_{\nu,i} \boldsymbol{f}(t_n + c_i h, \boldsymbol{\xi}_i) \\
&\text{(update)} && \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{j=1}^{\nu} b_j \boldsymbol{f}(t_n + c_j h, \boldsymbol{\xi}_j).
\end{aligned}$$

From the above formulation, we observe that a Runge-Kutta method is completely determined by the coefficients a_{ij}, b_i, and c_i. It is convenient to collect them into a single tabular form:

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Note that $c_1 = 0$, since it corresponds to the state $\boldsymbol{y}_n$. This tabular representation is known as the *Butcher tableau*, and it uniquely specifies a Runge-Kutta method.

We next consider how to determine the coefficients in the Runge–Kutta tableau. We begin with the simplest case $\nu = 2$. In this case, the tableau has the form

$$\begin{array}{c|cc}
0 & & \\
c_2 & a_{21} & \\
\hline
 & b_1 & b_2
\end{array}$$

and the corresponding Runge-Kutta method reads

$$\boldsymbol{\xi}_1 = \boldsymbol{y}_n, \tag{5a}$$

$$\boldsymbol{\xi}_2 = \boldsymbol{y}_n + h a_{21} \boldsymbol{f}(t_n, \boldsymbol{\xi}_1), \tag{5b}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\Big(b_1 \boldsymbol{f}(t_n, \boldsymbol{\xi}_1) + b_2 \boldsymbol{f}(t_n + c_2 h, \boldsymbol{\xi}_2)\Big). \tag{5c}$$

We refer to this as an RK2 method since it uses $\nu = 2$ stages. Thus, we only need to determine c_2, a_{21}, b_1, and b_2.

Our goal is to construct a method with fast convergence. As we did for the Euler and trapezoidal methods, we analyze the (local) truncation error using Taylor expansion. First, we expand $\boldsymbol{f}(t_n + c_2 h, \boldsymbol{\xi}_2)$ about $(t_n, \boldsymbol{y}_n)$. Since

$$\boldsymbol{\xi}_2 = \boldsymbol{y}_n + h a_{21} \boldsymbol{f}(t_n, \boldsymbol{y}_n),$$

we obtain

$$\begin{aligned}\boldsymbol{f}(t_n + c_2 h, \boldsymbol{\xi}_2) &= \boldsymbol{f}(t_n + c_2 h, \boldsymbol{y}_n + h a_{21} \boldsymbol{f}(t_n, \boldsymbol{y}_n)) \\ &= \boldsymbol{f}(t_n, \boldsymbol{y}_n) + h \left[c_2 \frac{\partial \boldsymbol{f}}{\partial t}(t_n, \boldsymbol{y}_n) + a_{21} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_n, \boldsymbol{y}_n)\, \boldsymbol{f}(t_n, \boldsymbol{y}_n) \right] + \mathcal{O}(h^2).\end{aligned}$$

Substituting this expansion into the update formula (5c) yields

$$\begin{aligned}\boldsymbol{y}_{n+1} = \boldsymbol{y}_n &+ h(b_1 + b_2) \boldsymbol{f}(t_n, \boldsymbol{y}_n) \\ &+ h^2 b_2 \left[c_2 \frac{\partial \boldsymbol{f}}{\partial t}(t_n, \boldsymbol{y}_n) + a_{21} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_n, \boldsymbol{y}_n)\, \boldsymbol{f}(t_n, \boldsymbol{y}_n) \right] + \mathcal{O}(h^3).\end{aligned}$$

Now we replace $\boldsymbol{y}_n$ by the exact solution value $\boldsymbol{y}(t_n)$ to obtain the local truncation error:

$$\begin{aligned}-\boldsymbol{y}(t_n + h) + \boldsymbol{y}(t_n) &+ h(b_1 + b_2) \boldsymbol{y}'(t_n) \\ &+ h^2 b_2 \left[c_2 \frac{\partial \boldsymbol{f}}{\partial t}(t_n, \boldsymbol{y}(t_n)) + a_{21} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_n, \boldsymbol{y}(t_n))\, \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) \right] + \mathcal{O}(h^3). \quad (6)\end{aligned}$$

On the other hand, differentiating the ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ with respect to t gives

$$\boldsymbol{y}'' = \frac{\partial \boldsymbol{f}}{\partial t} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} \boldsymbol{y}' = \frac{\partial \boldsymbol{f}}{\partial t} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} \boldsymbol{f}.$$

Therefore, Taylor's theorem implies

$$\boldsymbol{y}(t_n + h) = \boldsymbol{y}(t_n) + h \boldsymbol{y}'(t_n) + \frac{h^2}{2} \left(\frac{\partial \boldsymbol{f}}{\partial t}(t_n, \boldsymbol{y}(t_n)) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_n, \boldsymbol{y}(t_n))\, \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) \right) + \mathcal{O}(h^3). \quad (7)$$

Comparing (6) with (7), we see that in order for the truncation error to be $\mathcal{O}(h^3)$ (and hence for the method to be second-order), it suffices to enforce

$$b_1 + b_2 = 1, \qquad b_2 c_2 = \frac{1}{2}, \qquad a_{21} = c_2.$$

Popular choices of parameters are given by the following RK2 tableau:

$$\begin{array}{c|cc} 0 & & \\ 1/2 & 1/2 & \\ \hline & 0 & 1 \end{array}, \qquad \begin{array}{c|cc} 0 & & \\ 2/3 & 2/3 & \\ \hline & 1/4 & 3/4 \end{array}, \qquad \begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}. \quad (8)$$

Similar ideas can be used to derive the parameters for Runge-Kutta methods with $\nu \geq 3$ stages. For example, an explicit three-stage Runge-Kutta method (RK3) has the tableau

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

To obtain third-order methods, one can again match the numerical update with Taylor expansions of the exact solution. This leads to the following order conditions:

$$c_2 = a_{21}, \quad c_3 = a_{31} + a_{32}$$
$$b_1 + b_2 + b_3 = 1, \quad b_2 c_2 + b_3 c_3 = \frac{1}{2}, \quad b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3}, \quad b_3 a_{32} c_2 = \frac{1}{6}.$$

Two popular RK3 choices are:

(Classical RK3) $\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & -1 & 2 & \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$, (Nystrom RK3) $\begin{array}{c|ccc} 0 & & & \\ 2/3 & 2/3 & & \\ 2/3 & 0 & 2/3 & \\ \hline & 1/4 & 3/8 & 3/8 \end{array}$.

Finally, we present the classical fourth-order Runge-Kutta method (RK4). Its derivation follows the same philosophy, but the computation becomes more involved, so we only present the tableau:

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

Consistency conditions for general (explicit) RK methods. Although deriving the full set of conditions that guarantee the optimal order ν for a ν-stage Runge–Kutta (RK) method can be complicated, it is much easier to obtain a few basic *consistency* conditions that must hold for any explicit RK method. From the tableaus for RK3 and RK4, we observe that

$$\sum_{j=1}^{\nu} b_j = 1, \qquad c_j = \sum_{i<j} a_{ji}, \quad (j = 1, \ldots, \nu). \tag{9}$$

These two conditions hold for any ν-stage explicit RK method. We now show how they can be derived.

Consider the ODE

$$y'(t) = f(t, y(t)), \qquad y(t_n) = \tilde{y}_n,$$

and a ν-stage explicit RK method given by

$$\begin{aligned}
\xi_1 &= f(t_n, \tilde{y}_n), \\
\xi_j &= f\left(t_n + c_j h,\ \tilde{y}_n + h \sum_{i<j} a_{ji}\xi_i\right), \qquad j = 2, \ldots, \nu, \\
y_{n+1} &= \tilde{y}_n + h \sum_{j=1}^{\nu} b_j \xi_j.
\end{aligned}$$

To derive $\sum_{j=1}^{\nu} b_j = 1$, apply the method to the simple ODE $y'(t) = 1$ (i.e. $f(t, y) \equiv 1$). The exact solution satisfies $y(t_n + h) = \tilde{y}_n + h$. For the RK method, all stages satisfy $\xi_j = 1$, and hence

$$y_{n+1} = \tilde{y}_n + h \sum_{j=1}^{\nu} b_j.$$

Requiring $y_{n+1} = \tilde{y}_n + h$ immediately yields $\sum_{j=1}^{\nu} b_j = 1$. In other words, this condition is necessary for the method to be consistent even for a constant right-hand side f.

To derive $c_j = \sum_{i<j} a_{ji}$, note that the j-th stage ξ_j is intended to approximate the derivative at time $t_n + c_j h$:

$$\xi_j = f\left(t_n + c_j h,\ \tilde{y}_n + h \sum_{i<j} a_{ji}\xi_i\right) \approx y'(t_n + c_j h).$$

Therefore, the internal stage value

$$Y_j := \tilde{y}_n + h \sum_{i<j} a_{ji}\xi_i$$

should approximate the exact solution value $y(t_n + c_j h)$. Using a Taylor expansion,

$$y(t_n + c_j h) = \tilde{y}_n + c_j h\, y'(t_n) + \mathcal{O}(h^2) = \tilde{y}_n + c_j h\, f(t_n, \tilde{y}_n) + \mathcal{O}(h^2).$$

On the other hand, for an explicit RK method we have $\xi_i = f(t_n, \tilde{y}_n) + \mathcal{O}(h)$ for all $i < j$, and therefore

$$Y_j = \tilde{y}_n + h\Big(\sum_{i<j} a_{ji}\Big) f(t_n, \tilde{y}_n) + \mathcal{O}(h^2).$$

Matching the $\mathcal{O}(h)$ terms gives $c_j = \sum_{i<j} a_{ji}$ (in particular, $c_1 = 0$). This establishes (9).

Example. Consider again

$$\boldsymbol{y}' = \boldsymbol{y}(1 - \boldsymbol{y}), \qquad \boldsymbol{y}(0) = \frac{1}{10}.$$

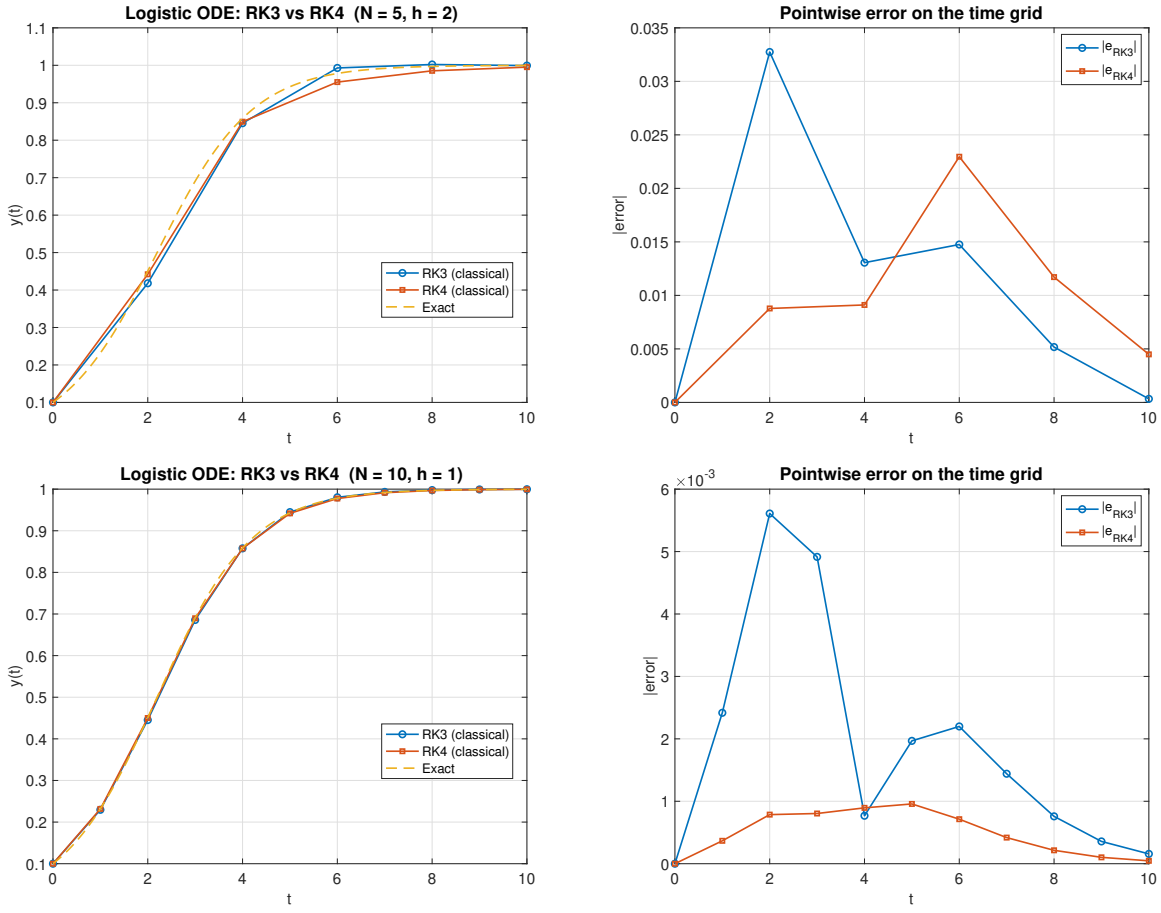
We want to predict the behavior of $\boldsymbol{y}$ on $[0, T]$ with $T = 10$.

- Matlab code of Runge-Kutta methods

```
% Compare RK3 vs RK4 for y' = y(1 - y),  y(0) = 1/10
clear; clc; close all

% ---------- Global font settings ----------
set(groot, 'defaultAxesFontSize', 14);
set(groot, 'defaultTextFontSize', 14);
set(groot, 'defaultLegendFontSize', 12);

% Parameters
y0 = 1/10;
T  = 10;
N  = 10;           % try 10, 20, 50, 100, ...
h  = T/N;

% Grid
t = linspace(0, T, N+1).';

% RHS and exact solution
f = @(tt, yy) yy .* (1 - yy);
y_exact = @(tt) 1 ./ (1 + 9*exp(-tt));

% Storage
y_rk3 = zeros(N+1,1); y_rk3(1) = y0;
y_rk4 = zeros(N+1,1); y_rk4(1) = y0;

% Time stepping
for n = 1:N
    tn = t(n);

    % ---- RK3 (classical) ----
    yn = y_rk3(n);
    k1 = f(tn, yn);
    k2 = f(tn + h/2, yn + (h/2)*k1);
    k3 = f(tn + h,   yn - h*k1 + 2*h*k2);
    y_rk3(n+1) = yn + h*( (1/6)*k1 + (2/3)*k2 + (1/6)*k3 );

    % ---- RK4 (classical) ----
    yn = y_rk4(n);
    k1 = f(tn, yn);
    k2 = f(tn + h/2, yn + (h/2)*k1);
    k3 = f(tn + h/2, yn + (h/2)*k2);
    k4 = f(tn + h,   yn + h*k3);
    y_rk4(n+1) = yn + (h/6)*(k1 + 2*k2 + 2*k3 + k4);
end

% Exact on same grid (for error)
y_ex = y_exact(t);

% Errors
e_rk3 = y_rk3 - y_ex;
e_rk4 = y_rk4 - y_ex;

err_rk3_inf = norm(e_rk3, inf);
err_rk4_inf = norm(e_rk4, inf);
```

```
err_rk3_rms = norm(e_rk3, 2) / sqrt(numel(e_rk3));
err_rk4_rms = norm(e_rk4, 2) / sqrt(numel(e_rk4));

fprintf('T = %.2f, N = %d, h = %.4g\n', T, N, h);
fprintf('RK3: ||e||_inf = %.3e,  RMS = %.3e\n', err_rk3_inf, err_rk3_rms);
fprintf('RK4: ||e||_inf = %.3e,  RMS = %.3e\n', err_rk4_inf, err_rk4_rms);

% ---------- Plot solutions ----------
t_fine = linspace(0, T, 1000).';

figure;
plot(t, y_rk3, 'o-', 'LineWidth', 1.5); hold on;
plot(t, y_rk4, 's-', 'LineWidth', 1.5);
plot(t_fine, y_exact(t_fine), '--', 'LineWidth', 1.5);
grid on;
xlabel('t', 'FontSize', 14);
ylabel('y(t)', 'FontSize', 14);
legend('RK3 (classical)', 'RK4 (classical)', 'Exact', ...
       'Location', 'best', 'FontSize', 12);
title(sprintf('Logistic ODE: RK3 vs RK4  (N = %d, h = %.4g)', N, h), ...
      'FontSize', 16);

% ---------- Plot pointwise errors ----------
figure;
plot(t, abs(e_rk3), 'o-', 'LineWidth', 1.5); hold on;
plot(t, abs(e_rk4), 's-', 'LineWidth', 1.5);
grid on;
xlabel('t', 'FontSize', 14);
ylabel('|error|', 'FontSize', 14);
legend('|e_{RK3}|', '|e_{RK4}|', ...
       'Location', 'best', 'FontSize', 12);
title('Pointwise error on the time grid', 'FontSize', 16);
```

- Results



Implicit Runge–Kutta methods. Although explicit Runge-Kutta (RK) methods are the most widely used, implicit Runge–Kutta (IRK) methods are important in certain scenarios. In particular, IRK methods often have much better stability. The tradeoff is that each time step typically requires solving a possibly nonlinear system for the stage values.

For instance, one two-stage IRK method is

$$\begin{aligned}
\boldsymbol{\xi}_1 &= \boldsymbol{y}_n + h\left[\frac{1}{4}\boldsymbol{f}(t_n, \boldsymbol{\xi}_1) - \frac{1}{4}\boldsymbol{f}\left(t_n + \frac{2}{3}h, \boldsymbol{\xi}_2\right)\right], \\
\boldsymbol{\xi}_2 &= \boldsymbol{y}_n + h\left[\frac{1}{4}\boldsymbol{f}(t_n, \boldsymbol{\xi}_1) + \frac{5}{12}\boldsymbol{f}\left(t_n + \frac{2}{3}h, \boldsymbol{\xi}_2\right)\right], \\
\boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + h\left[\frac{1}{4}\boldsymbol{f}(t_n, \boldsymbol{\xi}_1) + \frac{3}{4}\boldsymbol{f}\left(t_n + \frac{2}{3}h, \boldsymbol{\xi}_2\right)\right].
\end{aligned}$$

Its Butcher tableau is

$$\begin{array}{c|cc}
0 & \frac{1}{4} & -\frac{1}{4} \\
\frac{2}{3} & \frac{1}{4} & \frac{5}{12} \\
\hline
 & \frac{1}{4} & \frac{3}{4}
\end{array}.$$

Compared with the explicit RK2 tableau (8), we see that for explicit RK methods the matrix A is strictly lower triangular, so the stages can be computed sequentially without solving

equations. In contrast, for IRK methods the matrix A is generally full, so the stage equations are coupled and must be solved simultaneously.

Another useful feature of IRK methods is that, with the same number of stages, they can achieve higher order than explicit RK methods. For instance, we have the following theorem.

Theorem 3.1. *For any $\nu \geq 1$, there exists a ν-stage implicit Runge–Kutta method of order 2ν.*

In contrast, for the explicit RK methods we have studied so far, the order is at most ν when using ν stages. We next investigate more closely at these IRK methods which can achieve 2ν convergence.

Collocation implicit Runge–Kutta methods. The methods that achieve the 2ν convergence rate stated in Theorem 3.1 belong to a particularly important subclass of implicit Runge–Kutta methods. The idea is as follows. Let us return to the ODE

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)).$$

Given the state $\boldsymbol{y}(t_n)$, our goal is to construct an accurate prediction of the solution at the next time level, $\boldsymbol{y}(t_n + h)$.

Collocation methods follow a *trial–test formulation*. Namely, we assume that the solution can be well approximated within a chosen class of *trial* functions, and we then enforce the governing equation by *testing* it at selected locations in the time interval. This general trial–test philosophy will appear repeatedly throughout this course, for instance in the finite element method section later in this lecture.

More specifically, since polynomials are powerful approximators for smooth functions, in the collocation methods considered here we choose the trial functions to be polynomials. Then, we enforce the ODE at a finite set of carefully chosen testing points. These points are referred to as *collocation points*.

To make this precise, we first specify the trial. Namely, we seek a polynomial $\boldsymbol{u}$ of degree ν that approximates $\boldsymbol{y}(t)$ on the time interval $[t_n, t_n + h]$.

We then specify the test by selecting a collection of collocation points

$$0 \leq c_1 < c_2 < \cdots < c_\nu \leq 1,$$

so that the corresponding times $t_n + c_j h$ lie in the interval $[t_n, t_n + h]$. At these points, we require that the trial function $\boldsymbol{u}$ satisfies the ODE:

$$\boldsymbol{u}(t_n) = \boldsymbol{y}_n, \tag{10a}$$

$$\boldsymbol{u}'(t_n + c_j h) = \boldsymbol{f}\big(t_n + c_j h, \boldsymbol{u}(t_n + c_j h)\big), \qquad j = 1, 2, \ldots, \nu. \tag{10b}$$

Note that we have imposed $\nu + 1$ conditions, and that $\boldsymbol{u}$ has $\nu + 1$ degrees of freedom as a polynomial of degree ν. Therefore, we can (in principle) solve for the polynomial $\boldsymbol{u}$.[1] Once $\boldsymbol{u}$ is determined, we predict the next state by

$$\boldsymbol{y}_{n+1} = \boldsymbol{u}(t_n + h). \tag{10c}$$

The following lemma shows that the collocation method defined above can be written in the form of an (implicit) Runge–Kutta method.

[1] In general, this leads to a nonlinear system, since $\boldsymbol{f}$ depends on $\boldsymbol{u}$.

Lemma 3.1. *Set*

$$q(t) := \prod_{j=1}^{\nu} (t - c_j), \qquad q_\ell(t) := \frac{q(t)}{t - c_\ell}, \quad \ell = 1, 2, \dots, \nu.$$

Then the collocation method defined by (10) is a Runge–Kutta method with Butcher tableau given by

$$b_j := \int_0^1 \frac{q_j(\tau)}{q_j(c_j)}\, d\tau,$$
$$a_{j,i} := \int_0^{c_j} \frac{q_i(\tau)}{q_i(c_i)}\, d\tau, \qquad i, j = 1, 2, \dots, \nu.$$

Proof. Recall that the goal of the collocation method is to determine the polynomial $\boldsymbol{u}$ of degree ν on $[t_n, t_n + h]$ satisfying (10). Let

$$\boldsymbol{r}(t) := \boldsymbol{u}'(t).$$

Then $\boldsymbol{r}$ is a polynomial of degree $\nu - 1$. We represent $\boldsymbol{r}$ in the Lagrange basis associated with the collocation points $\{c_j\}_{j=1}^{\nu}$, mapped to the time interval $[t_n, t_n + h]$. Namely, we write

$$\boldsymbol{r}(t) = \sum_{\ell=1}^{\nu} \boldsymbol{w}_\ell\, \ell_\ell\left(\frac{t - t_n}{h}\right), \qquad \ell_\ell(\tau) := \prod_{j \neq \ell} \frac{\tau - c_j}{c_\ell - c_j} = \frac{q_\ell(\tau)}{q_\ell(c_\ell)}.$$

By construction of the Lagrange basis, we have the interpolation property

$$\ell_\ell(c_j) = \delta_{\ell j}.$$

Therefore, evaluating at the collocation points $t = t_n + c_j h$ yields

$$\boldsymbol{r}(t_n + c_j h) = \sum_{\ell=1}^{\nu} \boldsymbol{w}_\ell\, \ell_\ell(c_j) = \boldsymbol{w}_j.$$

On the other hand, by the collocation conditions (10),

$$\boldsymbol{r}(t_n + c_j h) = \boldsymbol{u}'(t_n + c_j h) = \boldsymbol{f}\big(t_n + c_j h, \boldsymbol{u}(t_n + c_j h)\big).$$

Hence,

$$\boldsymbol{w}_j = \boldsymbol{f}\big(t_n + c_j h, \boldsymbol{u}(t_n + c_j h)\big), \qquad j = 1, 2, \dots, \nu.$$

Next, we recover $\boldsymbol{u}$ by integrating $\boldsymbol{r}$. For any $t \in [t_n, t_n + h]$, we have

$$\begin{aligned}
\boldsymbol{u}(t) &= \boldsymbol{u}(t_n) + \int_{t_n}^{t} \boldsymbol{r}(s)\, ds \\
&= \boldsymbol{y}_n + \sum_{\ell=1}^{\nu} \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{u}(t_n + c_\ell h)\big) \int_{t_n}^{t} \ell_\ell\left(\frac{s - t_n}{h}\right) ds \\
&= \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{u}(t_n + c_\ell h)\big) \int_0^{(t-t_n)/h} \ell_\ell(\tau)\, d\tau \\
&= \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{u}(t_n + c_\ell h)\big) \int_0^{(t-t_n)/h} \frac{q_\ell(\tau)}{q_\ell(c_\ell)}\, d\tau.
\end{aligned}$$

In particular, evaluating at the collocation points $t = t_n + c_j h$ yields

$$\boldsymbol{u}(t_n + c_j h) = \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{u}(t_n + c_\ell h)\big) \int_0^{c_j} \frac{q_\ell(\tau)}{q_\ell(c_\ell)}\, d\tau, \qquad j = 1, 2, \ldots, \nu.$$

We now define the stage values by

$$\boldsymbol{\xi}_j := \boldsymbol{u}(t_n + c_j h), \qquad j = 1, 2, \ldots, \nu.$$

Then the above relation can be written as

$$\boldsymbol{\xi}_j = \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{\xi}_\ell\big) \int_0^{c_j} \frac{q_\ell(\tau)}{q_\ell(c_\ell)}\, d\tau.$$

Defining

$$a_{j\ell} := \int_0^{c_j} \frac{q_\ell(\tau)}{q_\ell(c_\ell)}\, d\tau,$$

we recover the Runge–Kutta stage equations

$$\boldsymbol{\xi}_j = \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} a_{j\ell}\, \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{\xi}_\ell\big), \qquad j = 1, 2, \ldots, \nu.$$

Finally, evaluating $\boldsymbol{u}$ at $t = t_n + h$ gives

$$\boldsymbol{y}_{n+1} = \boldsymbol{u}(t_n + h) = \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{u}(t_n + c_\ell h)\big) \int_0^{1} \frac{q_\ell(\tau)}{q_\ell(c_\ell)}\, d\tau.$$

Using again the definition $\boldsymbol{\xi}_\ell = \boldsymbol{u}(t_n + c_\ell h)$ and defining

$$b_\ell := \int_0^{1} \frac{q_\ell(\tau)}{q_\ell(c_\ell)}\, d\tau,$$

we obtain the Runge–Kutta update

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{\ell=1}^{\nu} b_\ell\, \boldsymbol{f}\big(t_n + c_\ell h, \boldsymbol{\xi}_\ell\big).$$

This shows that the collocation method (10) takes the standard implicit Runge–Kutta form with coefficients $\{a_{j\ell}\}$ and $\{b_\ell\}$ as defined above. This completes the proof.

□

The lemma shows that any collocation method can be expressed as a Runge–Kutta method, but it does not address how the collocation points c_j should be chosen. Once the points c_j are fixed, the coefficients b_j and $a_{j,i}$ are determined automatically by the above formulas. The next theorem provides guidance on the choice of the c_j.

Theorem 3.2. *Suppose that*

$$\int_0^1 q(\tau)\,\tau^j\,d\tau = 0, \quad j = 0, 1, 2, \ldots, m-1,$$

for some $m \in \{0, 1, 2, \ldots, \nu\}$. *Then the collocation method* (10) *is of order* $\nu + m$. *In particular, if* $m = \nu$, *then the collocation method is of order* 2ν.

Proof. See the textbook for the proof. □

Together, Lemma 3.1 and Theorem 3.2 show how to construct implicit Runge–Kutta methods that achieve 2ν convergence, thereby providing a constructive proof of Theorem 3.1.

Example of collocation IRK when $\nu = 2$. Let us consider the case when $\nu = 2$. The specific method can be obtained in two steps.

- **Step 1: Use Theorem 3.2 to determine c_j.**
  Since $\nu = 2$, we have
  $$q(t) = (t - c_1)(t - c_2).$$
  Choosing $m = \nu = 2$ in Theorem 3.2, we require
  $$\int_0^1 (t - c_1)(t - c_2)\,dt = 0,$$
  $$\int_0^1 (t - c_1)(t - c_2)\,t\,dt = 0.$$
  Evaluating these integrals gives the system
  $$\frac{1}{3} - \frac{1}{2}(c_1 + c_2) + c_1 c_2 = 0,$$
  $$\frac{1}{4} - \frac{1}{3}(c_1 + c_2) + \frac{1}{2} c_1 c_2 = 0.$$
  Solving for c_1 and c_2, we obtain
  $$c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}, \qquad c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}.$$
  Note that c_1 and c_2 are also the Gauss-Legendre quadrature nodes.
- **Step 2: Use Lemma 3.1 to compute the Butcher coefficients b_j and a_{ji}.**
  For $\nu = 2$, we have
  $$q(t) = (t - c_1)(t - c_2), \qquad q_1(t) = t - c_2, \qquad q_2(t) = t - c_1,$$
  and from Step 1
  $$c_{1,2} = \frac{1}{2} \mp \frac{\sqrt{3}}{6}, \qquad c_1 - c_2 = -\frac{\sqrt{3}}{3}, \qquad c_2 - c_1 = \frac{\sqrt{3}}{3}.$$

Hence

$$\frac{q_1(t)}{q_1(c_1)} = \frac{t-c_2}{c_1-c_2} = -\sqrt{3}\,(t-c_2), \qquad \frac{q_2(t)}{q_2(c_2)} = \frac{t-c_1}{c_2-c_1} = \sqrt{3}\,(t-c_1).$$

Weights. Using $\int_0^1 (t-c)\,dt = \frac{1}{2} - c$,

$$b_1 = \int_0^1 \frac{q_1(t)}{q_1(c_1)}\,dt = -\sqrt{3}\Big(\frac{1}{2} - c_2\Big) = \frac{1}{2}, \qquad b_2 = \int_0^1 \frac{q_2(t)}{q_2(c_2)}\,dt = \sqrt{3}\Big(\frac{1}{2} - c_1\Big) = \frac{1}{2}.$$

Stage matrix. Using $\int_0^c (t-c_2)\,dt = \frac{c^2}{2} - c_2 c$ and $\int_0^c (t-c_1)\,dt = \frac{c^2}{2} - c_1 c$, we obtain

$$\begin{aligned}
a_{11} &= \int_0^{c_1} \frac{q_1(t)}{q_1(c_1)}\,dt = -\sqrt{3}\Big(\frac{c_1^2}{2} - c_2 c_1\Big) = \frac{1}{4},\\
a_{12} &= \int_0^{c_1} \frac{q_2(t)}{q_2(c_2)}\,dt = \sqrt{3}\Big(\frac{c_1^2}{2} - c_1^2\Big) = -\sqrt{3}\,\frac{c_1^2}{2} = \frac{1}{4} - \frac{\sqrt{3}}{6},\\
a_{21} &= \int_0^{c_2} \frac{q_1(t)}{q_1(c_1)}\,dt = -\sqrt{3}\Big(\frac{c_2^2}{2} - c_2^2\Big) = \sqrt{3}\,\frac{c_2^2}{2} = \frac{1}{4} + \frac{\sqrt{3}}{6},\\
a_{22} &= \int_0^{c_2} \frac{q_2(t)}{q_2(c_2)}\,dt = \sqrt{3}\Big(\frac{c_2^2}{2} - c_1 c_2\Big) = \frac{1}{4},
\end{aligned}$$

where we used the identities

$$c_1^2 = \frac{2-\sqrt{3}}{6}, \qquad c_2^2 = \frac{2+\sqrt{3}}{6}, \qquad c_1 c_2 = \frac{1}{6}.$$

Combining Step 1 and Step 2, we obtain the Butcher tableau:

$$\begin{array}{c|cc}
\frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6}\\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4}\\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}$$

This is also called the two-stage Gauss-Legendre implicit Runge-Kutta method, which is of order 4 and achieves the optimal 2ν order for $\nu = 2$.

Example. Consider again

$$\boldsymbol{y}' = \boldsymbol{y}(1-\boldsymbol{y}), \qquad \boldsymbol{y}(0) = \frac{1}{10}.$$

We want to predict the behavior of $\boldsymbol{y}$ on $[0, T]$ with $T = 10$.

- Matlab code of RK4 (explicit) and IRK2 (Gauss-Legendre)

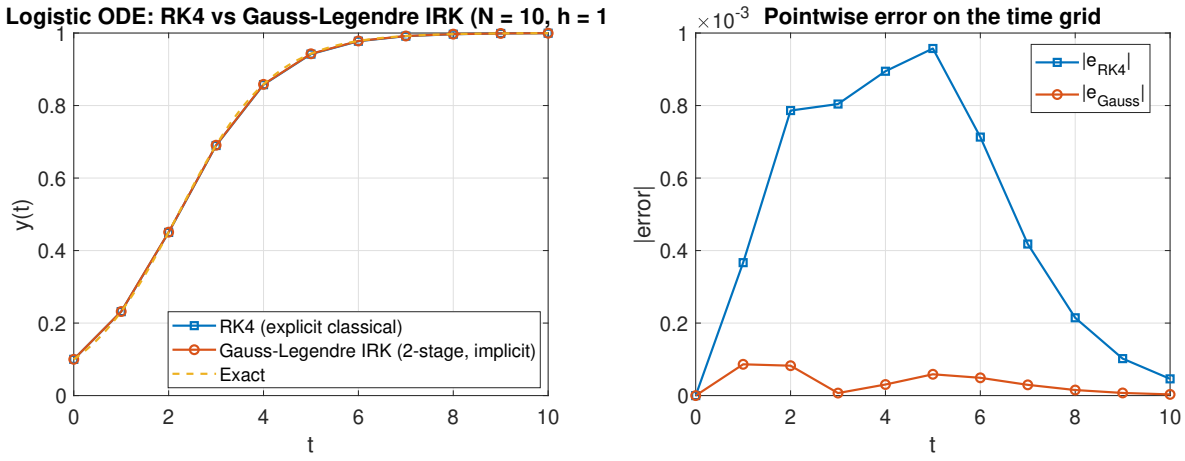
```
% Compare RK4 (explicit) vs Gauss-Legendre IRK (2-stage, implicit) for
% y' = y(1 - y),  y(0) = 1/10
clear; clc; close all

% ---------- Global font settings ----------
```

```
set(groot, 'defaultAxesFontSize', 14);
set(groot, 'defaultTextFontSize', 14);
set(groot, 'defaultLegendFontSize', 12);

% Parameters
y0 = 1/10;
T  = 10;
N  = 10;           % try 10, 20, 50, 100, ...
h  = T/N;

% Grid
t = linspace(0, T, N+1).';

% RHS and exact solution
f = @(tt, yy) yy .* (1 - yy);
y_exact = @(tt) 1 ./ (1 + 9*exp(-tt));

% Derivative of f wrt y (for Newton)
fy = @(tt, yy) 1 - 2*yy;

% Storage
y_rk4  = zeros(N+1,1); y_rk4(1)  = y0;
y_irk  = zeros(N+1,1); y_irk(1)  = y0;

% ---- Gauss-Legendre IRK (2-stage, order 4) coefficients ----
s3 = sqrt(3);

c1 = 1/2 - s3/6;
c2 = 1/2 + s3/6;

a11 = 1/4;
a12 = 1/4 - s3/6;
a21 = 1/4 + s3/6;
a22 = 1/4;

b1  = 1/2;
b2  = 1/2;

% Newton settings
newton_tol   = 1e-12;
newton_maxit = 25;

% Time stepping
for n = 1:N
    tn = t(n);

    % ---- RK4 (classical explicit) ----
    yn = y_rk4(n);
    k1 = f(tn, yn);
    k2 = f(tn + h/2, yn + (h/2)*k1);
    k3 = f(tn + h/2, yn + (h/2)*k2);
    k4 = f(tn + h,   yn + h*k3);
    y_rk4(n+1) = yn + (h/6)*(k1 + 2*k2 + 2*k3 + k4);
```

```
    % ---- Gauss-Legendre IRK (2-stage implicit) ----
    yn = y_irk(n);

    t1 = tn + c1*h;
    t2 = tn + c2*h;

    % Unknown stage values Y1, Y2 solve:
    % Y1 = yn + h*(a11*f(t1,Y1) + a12*f(t2,Y2))
    % Y2 = yn + h*(a21*f(t1,Y1) + a22*f(t2,Y2))

    % Initial guess: both stages start at yn
    Y = [yn; yn];

    for it = 1:newton_maxit
        Y1 = Y(1); Y2 = Y(2);

        f1 = f(t1, Y1);
        f2 = f(t2, Y2);

        % Residual F(Y)=0
        F = [ Y1 - yn - h*(a11*f1 + a12*f2);
              Y2 - yn - h*(a21*f1 + a22*f2) ];

        if norm(F, inf) < newton_tol
            break
        end

        % Jacobian J = dF/d[Y1;Y2]
        df1 = fy(t1, Y1);
        df2 = fy(t2, Y2);

        J = [ 1 - h*a11*df1,    -h*a12*df2;
             -h*a21*df1,        1 - h*a22*df2 ];

        dY = -J \ F;
        Y  = Y + dY;

        if norm(dY, inf) < newton_tol
            break
        end
    end

    Y1 = Y(1); Y2 = Y(2);
    y_irk(n+1) = yn + h*(b1*f(t1, Y1) + b2*f(t2, Y2));
end

% Exact on same grid (for error)
y_ex = y_exact(t);

% Errors
e_rk4 = y_rk4 - y_ex;
e_irk = y_irk - y_ex;

err_rk4_inf = norm(e_rk4, inf);
```

```
err_irk_inf = norm(e_irk, inf);
err_rk4_rms = norm(e_rk4, 2) / sqrt(numel(e_rk4));
err_irk_rms = norm(e_irk, 2) / sqrt(numel(e_irk));

fprintf('T = %.2f, N = %d, h = %.4g\n', T, N, h);
fprintf('RK4 :  ||e||_inf = %.3e,  RMS = %.3e\n', err_rk4_inf, err_rk4_rms);
fprintf('Gauss IRK (2-stage): ||e||_inf = %.3e,  RMS = %.3e\n', err_irk_inf, err_irk_rms);

% ---------- Plot solutions ----------
t_fine = linspace(0, T, 1000).';

figure;
plot(t, y_rk4, 's-', 'LineWidth', 1.5); hold on;
plot(t, y_irk, 'o-', 'LineWidth', 1.5);
plot(t_fine, y_exact(t_fine), '--', 'LineWidth', 1.5);
grid on;
xlabel('t');
ylabel('y(t)');
legend('RK4 (explicit classical)', 'Gauss-Legendre IRK (2-stage, implicit)', 'Exact', ...
       'Location', 'best');
title(sprintf('Logistic ODE: RK4 vs Gauss-Legendre IRK (N = %d, h = %.4g)', N, h));

% ---------- Plot pointwise errors ----------
figure;
plot(t, abs(e_rk4), 's-', 'LineWidth', 1.5); hold on;
plot(t, abs(e_irk), 'o-', 'LineWidth', 1.5);
grid on;
xlabel('t');
ylabel('|error|');
legend('|e_{RK4}|', '|e_{Gauss}|', 'Location', 'best');
title('Pointwise error on the time grid');
```

- Results



We see that, even with only two stages, the Gauss–Legendre IRK method can be more accurate than the classical explicit RK4 method (which uses four stages) for this test problem and step size. This is not guaranteed to hold in general. Still, it highlights a key advantage of implicit methods: they can deliver high accuracy with fewer stages.

4 Multistep methods

For the Euler method, the trapezoidal rule, or the Runge-Kutta methods we have studied, these methods advance the solution one step at a time: the new state $\boldsymbol{y}_i$ is determined entirely by the previous state $\boldsymbol{y}_{i-1}$. Methods with this property are called *one-step methods*.

A common guiding idea in numerical analysis is that, when additional information is available and used effectively, one can often achieve better results (e.g., higher accuracy). Multistep methods follow this idea by incorporating a short history of previously computed states. Schematically,

$$\text{(Multistep method)} \qquad (\boldsymbol{y}_{i-m}, \boldsymbol{y}_{i-m+1}, \ldots, \boldsymbol{y}_{i-2}, \boldsymbol{y}_{i-1}) \longrightarrow \boldsymbol{y}_i.$$

That is, instead of using only $\boldsymbol{y}_{i-1}$, a multistep method uses several past values $\boldsymbol{y}_{i-1}$, $\boldsymbol{y}_{i-2}$, ..., $\boldsymbol{y}_{i-s}$ to compute the next state $\boldsymbol{y}_i$.

A general s-step linear multistep method can be written in the form

$$\sum_{m=0}^{s} a_m \, \boldsymbol{y}_{i-s+m} = h \sum_{m=0}^{s} b_m \, \boldsymbol{f}(t_{i-s+m}, \boldsymbol{y}_{i-s+m}), \tag{11}$$

where $\{a_m\}_{m=0}^{s}$ and $\{b_m\}_{m=0}^{s}$ are given coefficients.

Why multistep? Before moving on, let us briefly explain why multistep methods remain valuable even though Runge–Kutta (RK) methods are available. Indeed, RK methods can achieve arbitrarily high order, and implicit RK schemes can be very stable (we will discuss the topic of stability later).

A key advantage of multistep methods is *efficiency*. They follow a classical *computational philosophy*: by storing a small amount of history (past solution values and/or past right-hand-side evaluations), one can reduce the computational work per time step. In particular, many multistep methods attain high-order accuracy while requiring only *one new* evaluation of the right-hand side $\boldsymbol{f}(t, \boldsymbol{y})$ per step (after an initialization phase). This can be especially beneficial when evaluating $\boldsymbol{f}$ is expensive, for instance when the ODE system arises from a high-order spatial discretization of a PDE.

To illustrate the idea, consider a three-step explicit multistep method of the form

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\Big[a \, \boldsymbol{f}(t_{i-3}, \boldsymbol{y}_{i-3}) + b \, \boldsymbol{f}(t_{i-2}, \boldsymbol{y}_{i-2}) + c \, \boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1})\Big],$$

which advances the solution from time level $i-1$ to i. Assume we have stored the recent history

$$\begin{aligned} &\boldsymbol{y}_{i-3}, \ \boldsymbol{y}_{i-2}, \ \boldsymbol{y}_{i-1}, \\ &\boldsymbol{f}_{i-3}, \ \boldsymbol{f}_{i-2}, \ \boldsymbol{f}_{i-1}, \end{aligned}$$

where $\boldsymbol{f}_j := \boldsymbol{f}(t_j, \boldsymbol{y}_j)$ for $j = i-3, i-2, i-1$. Then computing $\boldsymbol{y}_i$ requires only a linear combination of already-available values:

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\big(a \, \boldsymbol{f}_{i-3} + b \, \boldsymbol{f}_{i-2} + c \, \boldsymbol{f}_{i-1}\big).$$

After that, we perform exactly one new function evaluation,

$$\boldsymbol{f}_i := \boldsymbol{f}(t_i, \boldsymbol{y}_i), \qquad \text{(the only new evaluation needed at step } i\text{)},$$

and update the history by discarding the oldest entry:

$$\cancel{\boldsymbol{y}_{i-3}},\ \boldsymbol{y}_{i-2},\ \boldsymbol{y}_{i-1},\ \boldsymbol{y}_i,$$
$$\cancel{\boldsymbol{f}_{i-3}},\ \boldsymbol{f}_{i-2},\ \boldsymbol{f}_{i-1},\ \boldsymbol{f}_i.$$

Repeating this procedure advances the solution with only one new evaluation of $\boldsymbol{f}$ per step. The price for this efficiency is the need to store and maintain the short history table above.

Convergence of multistep. Let us now investigate the accuracy of multistep methods. We say that the method (11) is of order $p \geq 1$ if, for any sufficiently smooth exact solution $\boldsymbol{y}(t)$, it satisfies

$$\sum_{m=0}^{s} a_m\, \boldsymbol{y}(t+mh) - h \sum_{m=0}^{s} b_m\, \boldsymbol{y}'(t+mh) = \mathcal{O}(h^{p+1}), \qquad h \to 0. \tag{12}$$

The above relation is obtained by formally replacing the numerical values $\boldsymbol{y}_{i-s+m}$ in (11) with the exact solution values $\boldsymbol{y}(t+mh)$ (letting $t = t_{i-s}$) and then examining the resulting local truncation error.

The method (11) is completely determined by the coefficient sets $\{a_m\}_{m=0}^{s}$ and $\{b_m\}_{m=0}^{s}$. It is therefore convenient to encode the method by the associated characteristic polynomials

$$\rho(w) := \sum_{m=0}^{s} a_m w^m, \qquad \sigma(w) := \sum_{m=0}^{s} b_m w^m.$$

Theorem 4.1 (Order characterization via ρ and σ)**.** *The s-step multistep method (11) is of order $p \geq 1$ if and only if*

$$\rho(w) - \sigma(w) \ln w = \mathcal{O}\big(|w-1|^{p+1}\big), \qquad w \to 1.$$

Proof. Assume that $\boldsymbol{y}$ is analytic on an interval containing $[t,\, t+sh]$ so that it admits a convergent Taylor expansion there. For each $m = 0, 1, \ldots, s$, we have

$$\boldsymbol{y}(t+mh) = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{y}^{(k)}(t)\, m^k h^k.$$

Substituting this expansion into the local truncation error expression (12) gives

$$\begin{aligned}
\mathcal{O}(h^{p+1}) &= \sum_{m=0}^{s} a_m\, \boldsymbol{y}(t+mh) - h \sum_{m=0}^{s} b_m\, \boldsymbol{y}'(t+mh) \\
&= \sum_{m=0}^{s} a_m \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{y}^{(k)}(t)\, m^k h^k - h \sum_{m=0}^{s} b_m \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{y}^{(k+1)}(t)\, m^k h^k \\
&= \sum_{m=0}^{s} a_m\, \boldsymbol{y}(t) + \sum_{k=1}^{\infty} \frac{1}{k!} \left(\sum_{m=0}^{s} a_m m^k \right) h^k \boldsymbol{y}^{(k)}(t) - \sum_{k=1}^{\infty} \frac{1}{(k-1)!} \left(\sum_{m=0}^{s} b_m m^{k-1} \right) h^k \boldsymbol{y}^{(k)}(t) \\
&= \sum_{m=0}^{s} a_m\, \boldsymbol{y}(t) + \sum_{k=1}^{\infty} \frac{1}{k!} \left(\sum_{m=0}^{s} a_m m^k - k \sum_{m=0}^{s} b_m m^{k-1} \right) h^k \boldsymbol{y}^{(k)}(t).
\end{aligned}$$

Therefore, the method has order p (i.e., its local truncation error is $\mathcal{O}(h^{p+1})$) if and only if

$$\sum_{m=0}^{s} a_m = 0, \tag{13a}$$

$$\sum_{m=0}^{s} a_m m^k - k \sum_{m=0}^{s} b_m m^{k-1} = 0, \qquad k = 1, 2, \ldots, p. \tag{13b}$$

Next let $w = e^z$. Then $z \to 0$ if and only if $w \to 1$. Using $e^{zm} = \sum_{k=0}^{\infty} \frac{m^k z^k}{k!}$, we compute

$$\begin{aligned}
\rho(e^z) - z\sigma(e^z) &= \sum_{m=0}^{s} a_m e^{zm} - z \sum_{m=0}^{s} b_m e^{zm} \\
&= \sum_{m=0}^{s} a_m \left(\sum_{k=0}^{\infty} \frac{m^k z^k}{k!} \right) - \sum_{m=0}^{s} b_m \left(\sum_{k=1}^{\infty} \frac{m^{k-1} z^k}{(k-1)!} \right) \\
&= \sum_{m=0}^{s} a_m + \sum_{k=1}^{\infty} \frac{1}{k!} \left(\sum_{m=0}^{s} a_m m^k - k \sum_{m=0}^{s} b_m m^{k-1} \right) z^k.
\end{aligned}$$

Hence (13) holds if and only if

$$\rho(e^z) - z\sigma(e^z) = \mathcal{O}(z^{p+1}), \qquad z \to 0.$$

Finally, since $w = e^z$ implies $z = \ln w$, we obtain

$$\rho(w) - \sigma(w) \ln w = \mathcal{O}\big((\ln w)^{p+1}\big), \qquad w \to 1.$$

Moreover, since

$$\ln w = (w-1) + \mathcal{O}\big((w-1)^2\big) \qquad \text{as } w \to 1,$$

it follows that

$$(\ln w)^{p+1} = \mathcal{O}\big(|w-1|^{p+1}\big).$$

This proves the theorem. □

We next consider specific multistep methods and study their properties.

Adams-Bashforth method. From the ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, we have

$$\boldsymbol{y}(t_i) = \boldsymbol{y}(t_{i-1}) + \int_{t_{i-1}}^{t_i} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau)) \mathrm{d}\tau.$$

The idea behind *Adams-Bashforth* (AB) method is to approximate the integral $\int_{t_{i-1}}^{t_i} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau)) \mathrm{d}\tau$ by the past states' values $\boldsymbol{f}(t_{i-s}, \boldsymbol{y}_{i-s})$, $\boldsymbol{f}(t_{i-s+1}, \boldsymbol{y}_{i-s+1})$, ..., $\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1})$. To be more specific, we have

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h \sum_{j=0}^{s-1} b_j \boldsymbol{f}(t_{i-s+j}, \boldsymbol{y}_{i-s+j}), \tag{14}$$

where $h := t_i - t_{i-1}$ is the time step size, and b_j are fixed coefficients to be determined such that the scheme is high-order accurate.

The scheme (14) is called s-step Adams-Bashforth (AB) method. Let us examine some examples of Adams-Bashforth methods for $s = 1$, $s = 2$, and $s = 3$.

- When $s = 1$, we recover our old friend explicit Euler method, which is

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1}).$$

  Note that $b_0 = 1$ in this case.

- When $s = 2$, we have

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\left[-\frac{1}{2}\boldsymbol{f}(t_{i-2}, \boldsymbol{y}_{i-2}) + \frac{3}{2}\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1})\right]. \tag{15}$$

  Note that $b_0 = -1/2$ and $b_1 = 3/2$ in this case.

- When $s = 3$, we have

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + h\left[\frac{5}{12}\boldsymbol{f}(t_{i-3}, \boldsymbol{y}_{i-3}) - \frac{4}{3}\boldsymbol{f}(t_{i-2}, \boldsymbol{y}_{i-2}) + \frac{23}{12}\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1})\right] \tag{16}$$

  Note that $b_0 = \frac{5}{12}$, $b_1 = -\frac{4}{3}$, and $b_2 = \frac{23}{12}$ in this case.

Question: how are the parameters b_j are obtained for $s = 2$ or 3? We will use Theorem 4.1.

For $s = 2$, rearranging the AB2 scheme (15), we have

$$-\boldsymbol{y}_{i-1} + \boldsymbol{y}_i = h\left[-\frac{1}{2}\boldsymbol{f}(t_{i-2}, \boldsymbol{y}_{i-2}) + \frac{3}{2}\boldsymbol{f}(t_{i-1}, \boldsymbol{y}_{i-1})\right].$$

Thus, $a_0 = 0$, $a_1 = -1$, and $a_2 = 1$. In addition, $b_0 = -1/2$, $b_1 = 3/2$ and $b_2 = 0$.

Let $\xi = w - 1$. Then,

$$\rho(w) = -w + w^2 = \xi + \xi^2,$$
$$\sigma(w) = -1/2 + 3/2w = 1 + \frac{3}{2}\xi.$$

On the other hand, we have

$$\ln(w) = \xi - \frac{1}{2}\xi^2 + \frac{1}{3}\xi^3 + \mathcal{O}(|\xi|^4).$$

Therefore,

$$\rho(w) - \sigma(w)\ln w = (\xi + \xi^2) - (1 + \frac{3}{2}\xi)\left(\xi - \frac{1}{2}\xi^2 + \frac{1}{3}\xi^3 + \mathcal{O}(|\xi|^4)\right) = \mathcal{O}(|\xi|^3).$$

Thus, the AB2 method is second-order.

Similarly, we can verify that AB3 method (16) is third-order.

Backward differentiation formulas. Starting from the ODE $\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t))$, backward differentiation formulas (BDF) approximate the derivative $\boldsymbol{y}'(t_i)$ by a backward finite-difference formula using the past values $\boldsymbol{y}_i, \boldsymbol{y}_{i-1}, \ldots, \boldsymbol{y}_{i-s}$. The resulting scheme has the form

$$\sum_{m=0}^{s} a_m \, \boldsymbol{y}_{i-s+m} = h \, \boldsymbol{f}(t_i, \boldsymbol{y}_i), \qquad a_s \neq 0, \tag{17}$$

which is implicit in $\boldsymbol{y}_i$. Since we can multiply the scheme by any non-zero constant, an equivalent way to write the scheme is

$$\boldsymbol{y}_i - h \, a_s^{-1} \boldsymbol{f}(t_i, \boldsymbol{y}_i) = - \sum_{m=0}^{s-1} a_s^{-1} a_m \, \boldsymbol{y}_{i-s+m}. \tag{18}$$

Note that we will need to solve the above equation to obtain $\boldsymbol{y}_i$.

By contrast, recall that the s-step Adams–Bashforth (AB) method has the form

$$\boldsymbol{y}_i - \boldsymbol{y}_{i-1} = h \sum_{j=0}^{s-1} b_j \, \boldsymbol{f}(t_{i-s+j}, \boldsymbol{y}_{i-s+j}),$$

which is explicit since it only involves past values of $\boldsymbol{f}$.

More generally, recall that a linear multistep method can be written as

$$\sum_{m=0}^{s} a_m \, \boldsymbol{y}_{i-s+m} = h \sum_{m=0}^{s} b_m \, \boldsymbol{f}(t_{i-s+m}, \boldsymbol{y}_{i-s+m}).$$

If we summarize the coefficients in the following matrix:

$$\begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_s \\ b_0 & b_1 & b_2 & \cdots & b_s \end{pmatrix}.$$

Then, for Adams-Bashforth methods, one has

$$\text{AB} \qquad \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & -1 & 1 \\ b_0 & b_1 & b_2 & \cdots & b_{s-2} & b_{s-1} & 0 \end{pmatrix}.$$

For BDF methods, one has

$$\text{BDF} \qquad \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{s-2} & a_{s-1} & a_s \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}.$$

Some examples of BDF schemes are

- When $s = 1$, we recover the implicit Euler method

$$\boldsymbol{y}_i = \boldsymbol{y}_{i-1} + \boldsymbol{f}(t_i, \boldsymbol{y}_i).$$

- When $s = 2$, we have

$$\boldsymbol{y}_i - \frac{4}{3}\boldsymbol{y}_{i-1} + \frac{1}{3}\boldsymbol{y}_{i-2} = \frac{2}{3} h \boldsymbol{f}(t_i, \boldsymbol{y}_i).$$

- When $s = 3$, we have

$$\boldsymbol{y}_i - \frac{18}{11}\boldsymbol{y}_{i-1} + \frac{9}{11}\boldsymbol{y}_{i-2} - \frac{2}{11}\boldsymbol{y}_{i-3} = \frac{6}{11}h\boldsymbol{f}(t_i, \boldsymbol{y}_i).$$

We next show how the coefficients of the BDF methods can be derived. In fact, there is a closed-form expression for the characteristic polynomial $\rho(w)$ of the s-step BDF method.

Lemma 4.1. *For the s-step BDF method with order s, the characteristic polynomial is*

$$\rho(w) = \sum_{j=1}^{s} \frac{1}{j}\, w^{s-j}(w-1)^j.$$

Proof. We use the order characterization from Theorem 4.1:

$$\rho(w) - \sigma(w)\ln w = \mathcal{O}(|w-1|^{s+1}), \qquad w \to 1.$$

For a BDF method, $\sigma(w) = w^s$, and $\rho(w) = \sum_{j=0}^{s} a_j w^j$.

Set $v = w^{-1}$. Then $w \to 1$ is equivalent to $v \to 1$, and multiplying the order condition by v^s gives

$$v^s\rho(v^{-1}) + \ln v = \mathcal{O}(|v-1|^{s+1}), \qquad v \to 1.$$

Next we use the Taylor expansion of the logarithm about $v = 1$:

$$\ln v = \sum_{j=1}^{s} \frac{(-1)^{j-1}}{j}(v-1)^j + \mathcal{O}(|v-1|^{s+1}).$$

Since $v^s\rho(v^{-1}) = \sum_{j=0}^{s} a_j v^{s-j}$ is a polynomial of degree at most s in v, the relation above implies that its Taylor expansion about $v = 1$ must cancel the polynomial part of $\ln v$. Therefore,

$$v^s\rho(v^{-1}) = -\sum_{j=1}^{s} \frac{(-1)^{j-1}}{j}(v-1)^j = \sum_{j=1}^{s} \frac{(-1)^j}{j}(v-1)^j.$$

Substituting back $v = w^{-1}$ gives

$$\rho(w) = w^s \sum_{j=1}^{s} \frac{(-1)^j}{j}(w^{-1}-1)^j = \sum_{j=1}^{s} \frac{(-1)^j}{j} w^{s-j}(1-w)^j = \sum_{j=1}^{s} \frac{1}{j} w^{s-j}(w-1)^j,$$

which completes the proof. □

5 Stability

In the previous sections we studied various numerical methods for ODEs, including explicit and implicit Euler, the trapezoidal rule, Runge–Kutta methods, and multistep methods such as Adams–Bashforth and BDF.

A primary focus so far has been the *order* of a method, which measures how fast the local truncation error goes to zero as $h \to 0$. A method of order p has local truncation error $\mathcal{O}(h^{p+1})$; this property is referred to as *consistency*.

However, consistency alone does not guarantee convergence. Another equally important ingredient is *stability*. Roughly speaking, stability controls whether small perturbations (from roundoff, data noise, or previous steps) remain small or get amplified by the numerical method. As we will see, a method can be accurate locally but still fail to converge if it is unstable.

To illustrate this, consider the following two-step multistep method:

$$2y_{i-2} - 3y_{i-1} + y_i = h\left[-\frac{5}{12}f(t_{i-2}, y_{i-2}) - \frac{5}{3}f(t_{i-1}, y_{i-1}) + \frac{13}{12}f(t_i, y_i)\right].$$

One can verify by Theorem 4.1 that its local truncation error is $\mathcal{O}(h^3)$, so the method is second-order accurate.

Now apply this method to the trivial ODE

$$y'(t) = 0, \qquad y(0) = 1,$$

whose exact solution is $y(t) \equiv 1$. Since $f \equiv 0$, the scheme reduces to the recurrence

$$y_i = 3y_{i-1} - 2y_{i-2}.$$

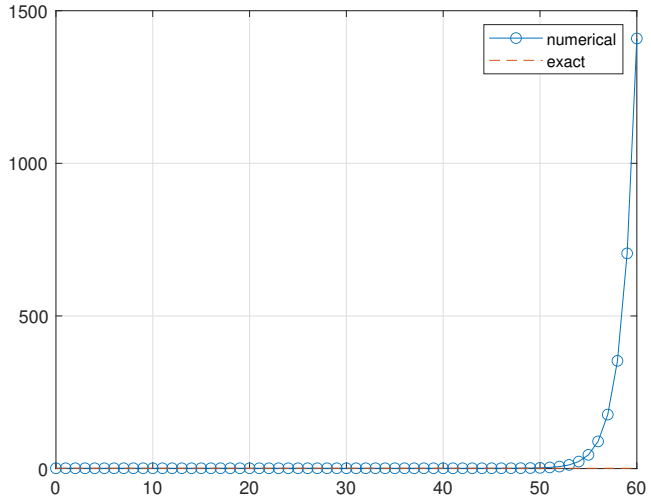
We choose the initial values

$$y_0 = 1, \qquad y_1 = 1 + 10^{-15},$$

where y_1 is perturbed by a very small error from the exact solution.

The resulting iteration can be computed in Matlab:

```
N  = 60;
y  = zeros(N+1,1);
n  = (0:N)';

y(1) = 1;
y(2) = 1 + 1e-15;

for k = 3:N+1
    y(k) = 3*y(k-1) - 2*y(k-2);
end

plot(n,y,'o-'); hold on;
plot(n,ones(size(n)),'--');
legend('numerical','exact');
grid on;
```



We observe that, despite the equation being trivial and the method having second-order accuracy, the numerical solution quickly diverges from the exact solution. A tiny perturbation is exponentially amplified. This shows that consistency alone does not guarantee convergence; stability is essential.

5.1 Zero-stability and root condition

The idea of zero-stability is, in simple terms, that the numerical method is stable when applied to the differential equation $\boldsymbol{y}'(t) = 0$.

For example, the multistep method

$$2y_{i-2} - 3y_{i-1} + y_i = h\left[-\frac{5}{12}f(t_{i-2}, y_{i-2}) - \frac{5}{3}f(t_{i-1}, y_{i-1}) + \frac{13}{12}f(t_i, y_i)\right]$$

that we studied above does *not* satisfy zero-stability.

To see this, let us directly study zero-stability for a general s-step linear multistep method

$$\sum_{m=0}^{s} a_m \, \boldsymbol{y}_{i-s+m} = h \sum_{m=0}^{s} b_m \, \boldsymbol{f}(t_{i-s+m}, \boldsymbol{y}_{i-s+m}).$$

To check zero-stability, we apply the method to the equation $\boldsymbol{y}'(t) = 0$, for which $\boldsymbol{f} \equiv 0$. The scheme then reduces to the homogeneous recurrence

$$\sum_{m=0}^{s} a_m \, \boldsymbol{y}_{i-s+m} = 0,$$

Thus, zero-stability depends only on the coefficients $\{a_m\}_{m=0}^{s}$ and amounts to determining whether this recurrence relation is stable.

To analyze this recurrence, we look for solutions of the form $\boldsymbol{y}_i = \xi^i$, where $\xi \in \mathbb{C}$. Substituting into the recurrence gives

$$\sum_{m=0}^{s} a_m \, \xi^m = 0.$$

This motivates the definition of the characteristic polynomial we have seen before:

$$\rho(\xi) = \sum_{m=0}^{s} a_m \, \xi^m.$$

The quantities ξ for which $\rho(\xi) = 0$ are precisely the modes governing the behavior of the recurrence.

If $|\xi| < 1$, then $\xi^i \to 0$ as $i \to \infty$, so the corresponding mode decays and is stable. If $|\xi| > 1$, then $|\xi^i| \to \infty$, and the mode grows exponentially, leading to instability.

If $|\xi| = 1$ and ξ is a simple root, then $|\xi^i| = 1$, and the corresponding mode is bounded. However, if $|\xi| = 1$ and ξ is a multiple root, then the recurrence contains solutions of the form

$$\boldsymbol{y}_i = p(i) \, \xi^i,$$

where $p(i)$ is a polynomial whose degree is one less than the multiplicity of ξ. In this case, $|p(i)\xi^i|$ grows without bound as $i \to \infty$, so the method is unstable.

We therefore arrive at the following fundamental result.

Theorem 5.1 (Root condition for zero-stability)**.** *A linear multistep method is zero-stable if and only if all roots ξ of the characteristic polynomial $\rho(\xi) = 0$ satisfy*

$$|\xi| \leq 1, \quad \textit{and any root with } |\xi| = 1 \textit{ is simple.}$$

Remark 5.1. *Note that for all Runge-Kutta methods and Adams-Bashforth (AB) methods, when applied to $\boldsymbol{y}'(t) = 0$ (so that $\boldsymbol{f} \equiv 0$), the scheme reduces to the simple recurrence*

$$\boldsymbol{y}_i - \boldsymbol{y}_{i-1} = 0.$$

Thus, they are automatically zero-stable.

Now let us examine the the previously considered multistep method

$$2y_{i-2} - 3y_{i-1} + y_i = h \left[-\frac{5}{12} f(t_{i-2}, y_{i-2}) - \frac{5}{3} f(t_{i-1}, y_{i-1}) + \frac{13}{12} f(t_i, y_i) \right]$$

Its root condition is

$$2 - 3\xi + \xi^2 = 0.$$

Thus, we have $\xi_1 = 1$ or $\xi_2 = 2$. Since $|\xi_2| = 2 > 1$, this scheme is not zero-stable.

Recall that for BDF methods, the characteristic polynomial ρ is determined by Lemma 4.1, since we enforce convergence of order s. It is therefore natural to ask whether this ρ satisfies the root condition. It turns out that the root condition holds only for $1 \leq s \leq 6$. Consequently, we have the following theorem.

Theorem 5.2. *The BDF method* (17) *is zero-stable if and only if* $1 \le s \le 6$.

To prove the theorem, recall from Lemma (4.1) that the characteristic polynomial of the s-step BDF method is

$$\rho(w) = \sum_{j=1}^{s} \frac{1}{j}\, w^{s-j}(w-1)^j.$$

Therefore, by the Theorem 5.1 (root condition for zero-stability), it suffices to determine for which s all roots of $\rho(w)$ lie in the closed unit disk $\{|w| \le 1\}$, with any roots on the unit circle being simple.

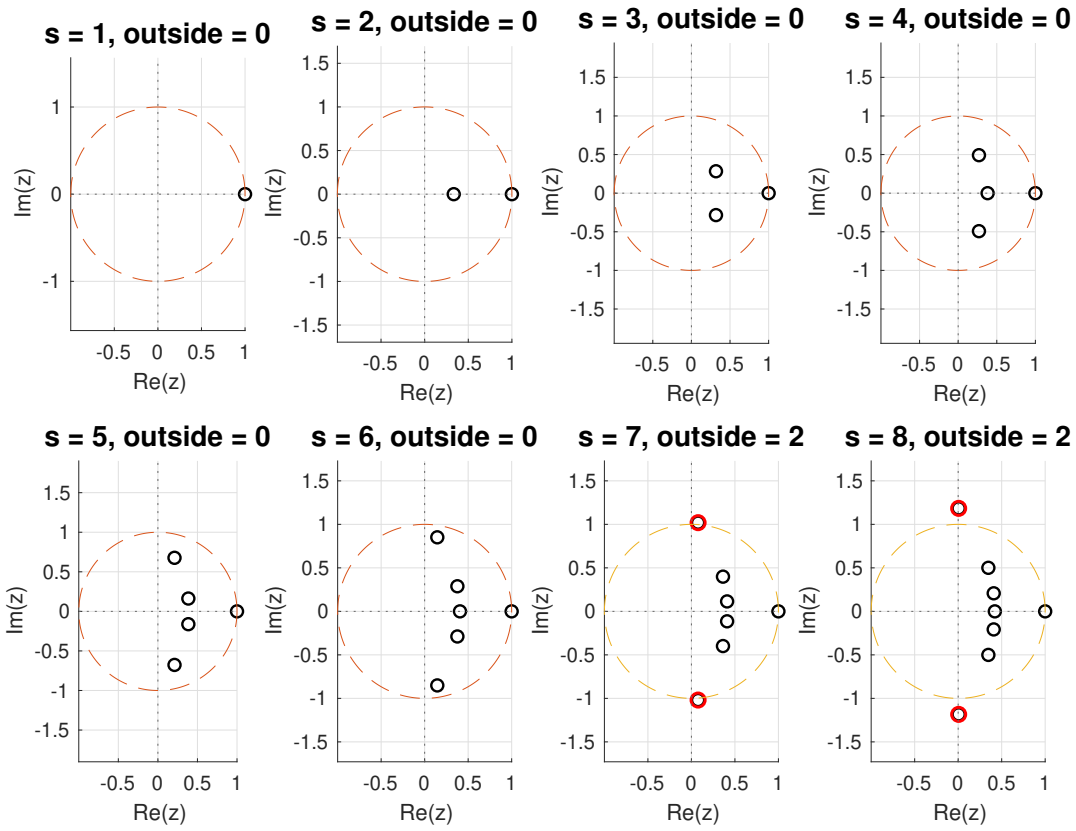
Let us plot the roots for $s = 1, \ldots, 8$.

- Code

```
clear; clc; close all;

syms w

Smax = 8;
tol = 1e-10;

figure('Color','w');
tiledlayout(2,4,'TileSpacing','compact','Padding','compact');

th = linspace(0, 2*pi, 400);
ucx = cos(th); ucy = sin(th);

for s = 1:Smax

    % Build rho_s(w) in its original (unexpanded) form
    p = 0;
    for j = 1:s
        p = p + (1/j) * w^(s-j) * (w - 1)^j;
    end

    % Convert to coefficient vector (descending powers)
    c = sym2poly(expand(p));

    % Compute roots
    r = roots(c);

    % Plot roots
    nexttile; hold on;
    plot(real(r), imag(r), 'ko', 'MarkerSize', 6, 'LineWidth', 1.2);

    % Highlight roots outside the unit disk
    outside = abs(r) > 1 + tol;
    plot(real(r(outside)), imag(r(outside)), 'ro', 'MarkerSize', 7, 'LineWidth', 1.5);

    % Unit circle for reference
    plot(ucx, ucy, '--');
    xline(0, ':'); yline(0, ':');
    axis equal; grid on;
```

```
    title(sprintf('s = %d, outside = %d', s, sum(outside)), ...
        'FontSize', 14, 'FontWeight', 'bold');
    xlabel('Re(w)'); ylabel('Im(w)');
end
```

- Results – roots of $\rho(w)$ for s-step BDF methods



Fortunately, the range $1 \leq s \leq 6$ is good enough in practice, since one rarely needs a multistep method with $s > 6$.

5.2 Linear stability region and A-stability

While *zero-stability* studies the stability of a numerical method when $f \equiv 0$, *linear stability* examines the behavior of the method when it is applied to the linear test equation

$$y' = \lambda y, \qquad t \geq 0, \tag{19}$$

where $\lambda \in \mathbb{C}$. The exact solution for (19) is

$$y(t) = ce^{\lambda t},$$

where c is determined by the initial condition. We focus on the case $\Re(\lambda) \leq 0$, since when $\Re(\lambda) > 0$ the exact solution itself grows exponentially.

To illustrate the idea of linear stability region, consider the forward Euler method applied to (19):

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

This gives the explicit formula

$$y_i = c(1 + h\lambda)^i.$$

Thus, in order for the numerical solution not to grow without bound as $i \to \infty$, we require

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

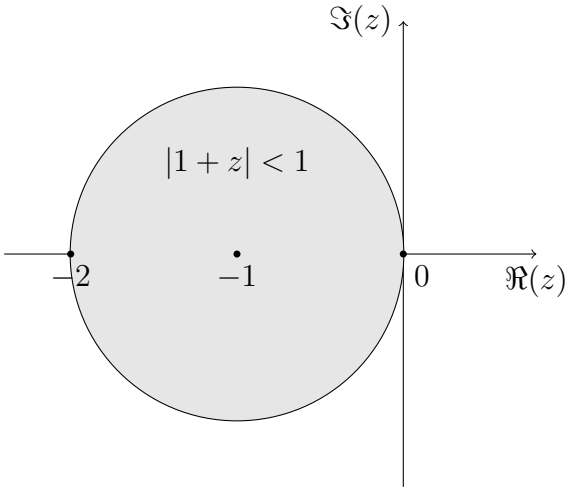
It is therefore natural to introduce the *stability function*

$$R(z) = 1 + z, \qquad z = h\lambda,$$

and to define the *linear stability region* of the Euler method by

$$\mathcal{D}_{\text{Euler}} = \{z \in \mathbb{C} : |R(z)| < 1\} = \{z \in \mathbb{C} : |1 + z| < 1\}.$$

This is the disk in the complex plane centered at -1 with radius 1.



Linear stability region can be used to guide our choice of the step size h for different λ. For instance, if $\lambda = -1$, then $z = h\lambda = -h$, and the stability condition

$$|1 - h| < 1$$

is equivalent to

$$0 \leq h < 2.$$

Thus, when solving $y' = -y$, the forward Euler method is stable only if the step size satisfies $h < 2$.

Usefulness of linear stability in the general case. Before moving on to examine the linear stability of other methods, let us first address a natural question that you may already have in mind:

Why do we study the test equation $y' = \lambda y$, when our real problem of interest is a possibly nonlinear system $\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y})$?

Suppose that at time t^* we have $\boldsymbol{y}(t^*) = \boldsymbol{y}^*$. Then, for t in a small neighborhood of t^*, the solution $\boldsymbol{y}(t)$ remains close to $\boldsymbol{y}^*$. Expanding $\boldsymbol{f}$ about $\boldsymbol{y}^*$ gives

$$\boldsymbol{f}(\boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{y}^*) + A(\boldsymbol{y} - \boldsymbol{y}^*) + o(\|\boldsymbol{y} - \boldsymbol{y}^*\|), \qquad A = \nabla \boldsymbol{f}(\boldsymbol{y}^*).$$

Thus, locally, the dynamics are governed by the linear system

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y}^*) + A(\boldsymbol{y} - \boldsymbol{y}^*).$$

This motivates us to study linear ODEs of the form

$$\boldsymbol{y}' = A\boldsymbol{y} + \boldsymbol{b},$$

where $\boldsymbol{b}$ is a constant forcing term. One can write the general solution of this equation as

$$\boldsymbol{y} = \boldsymbol{z} + \boldsymbol{y}_0,$$

where $\boldsymbol{y}_0$ is any particular solution satisfying $\boldsymbol{y}_0' = A\boldsymbol{y}_0 + \boldsymbol{b}$, and $\boldsymbol{z}$ is a general solution of the homogeneous system $\boldsymbol{z}' = A\boldsymbol{z}$. Thus, the problem reduces to studying the stability of the homogeneous system

$$\boldsymbol{z}' = A\boldsymbol{z}. \tag{20}$$

Let us remark that this argument serves only to motivate the study of the linearized system. It is far from sufficient to fully characterize the behavior of general nonlinear systems, which remains, to this day, a challenging and active area of research. In some special cases, however, the above heuristic of locally approximating the dynamics of an ODE by its linearization can be made rigorous; see, for instance, the Hartman–Grobman theorem.

Another reason for studying linear systems of the form (20) is that many important physical models are linear. Examples include linear wave equations (such as elastic and electromagnetic waves), the heat equation, and many others.

Now consider the linear ODE of the form (20):

$$\boldsymbol{y}' = A\boldsymbol{y}, \qquad A \in \mathbb{C}^{d\times d}.$$

If A is diagonalizable, then it admits an eigen-decomposition: there exist eigenpairs

$$A\boldsymbol{v}_\ell = \lambda_\ell \boldsymbol{v}_\ell, \qquad \ell = 1, \dots, d,$$

and any initial condition can be expanded in the corresponding eigenbasis. The exact solution can be written as

$$\boldsymbol{y}(t) = \sum_{\ell=1}^{d} c_\ell \, \boldsymbol{v}_\ell \, e^{\lambda_\ell t},$$

where the coefficients c_ℓ are determined by the initial condition.

Each eigenmode evolves independently, and its growth or decay is governed by the corresponding eigenvalue λ_ℓ. Therefore, both the continuous dynamics and the numerical method can be better understood by studying how they act on a single mode. This motivates the scalar model problem

$$y' = \lambda y,$$

which serves as the fundamental test equation for linear stability analysis.

Linear stability region for the trapezoidal method. Let us derive the linear stability region for the trapezoidal method. Letting $f = \lambda y$, recall that the trapezoidal method gives

$$\begin{aligned} y_i &= y_{i-1} + \frac{h}{2}\Big(f(t_{i-1}, y_{i-1}) + f(t_i, y_i)\Big) \\ &= y_{i-1} + \frac{h}{2}\lambda y_{i-1} + \frac{h}{2}\lambda y_i. \end{aligned}$$

Rearranging yields

$$\Big(1 - \frac{h}{2}\lambda\Big) y_i = \Big(1 + \frac{h}{2}\lambda\Big) y_{i-1}.$$

Thus,

$$y_i = R(z)\, y_{i-1}, \qquad R(z) = \frac{1 + z/2}{1 - z/2}, \qquad z := h\lambda.$$

Therefore, the stability function is

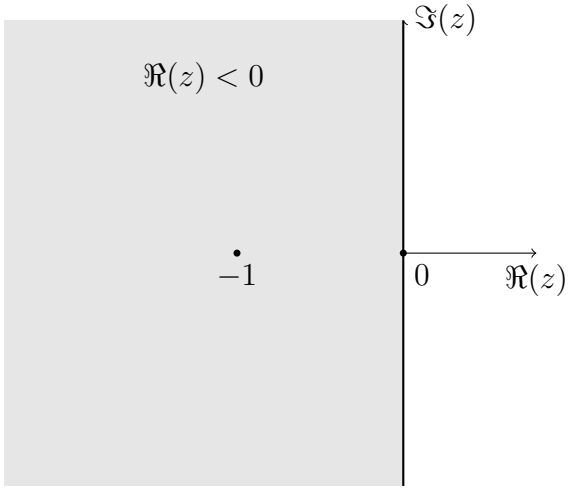
$$R(z) = \frac{1 + z/2}{1 - z/2}.$$

The linear stability region is

$$\mathcal{D}_{\text{trap}} = \left\{ z \in \mathbb{C} \,:\, \left| \frac{1 + z/2}{1 - z/2} \right| < 1 \right\}.$$

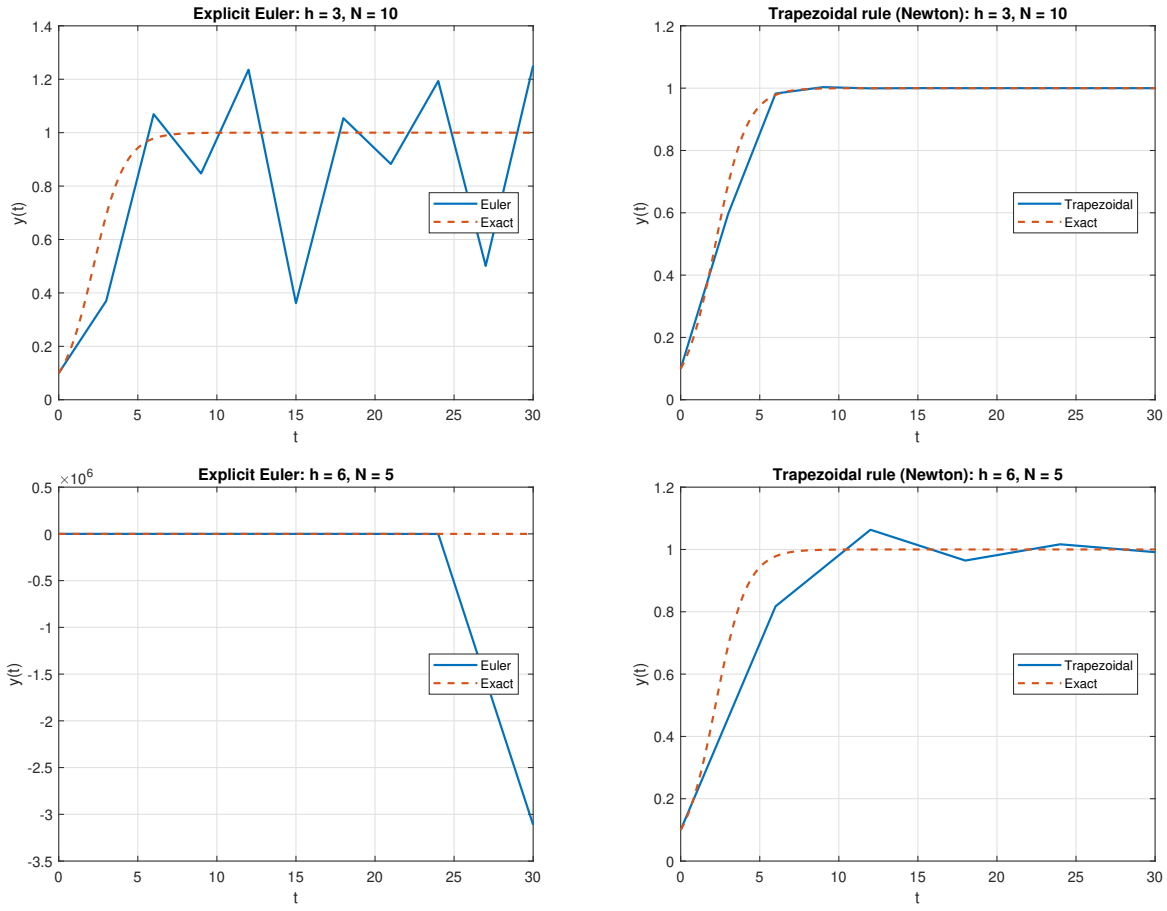
This corresponds to the open left half-plane in the complex plane, that is,

$$\mathcal{D}_{\text{trap}} = \{ z \in \mathbb{C} :\ \Re(z) < 0 \}.$$



Since $\Re(\lambda) < 0$ implies $\Re(z) = h\,\Re(\lambda) < 0$ for any $h > 0$, the method is stable for all step sizes. In this case, we say that the trapezoidal method is *A-stable*.

Example – Stability of Euler vs Trapezoidal method. Let us see some numerical examples, where we apply explicit Euler and Trapezoidal methods to the ODE $y' = y(1-y)$ with $y(0) = 0.1$ and $t \in [0, 30]$.



Linear stability region of Runge–Kutta methods. Applying a general ν-stage Runge–Kutta method to the ODE $y' = \lambda y$, we obtain

$$\xi_j = y_n + h\lambda \sum_{i=1}^{\nu} a_{j,i}\xi_i,$$

$$y_{n+1} = y_n + h\lambda \sum_{j=1}^{\nu} b_j \xi_j.$$

Let $z := h\lambda$ and define

$$\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_\nu)^\top, \quad A = \begin{pmatrix} a_{11} & \cdots & a_{1\nu} \\ \vdots & \ddots & \vdots \\ a_{\nu 1} & \cdots & a_{\nu\nu} \end{pmatrix}, \quad \boldsymbol{b} = (b_1, b_2, \ldots, b_\nu)^\top, \quad \mathbf{1} = (1, 1, \ldots, 1)^\top.$$

Then

$$\begin{aligned} \boldsymbol{\xi} &= (I - zA)^{-1}\mathbf{1}\, y_n, \\ y_{n+1} &= y_n + z\, \boldsymbol{b}^\top \boldsymbol{\xi}. \end{aligned}$$

Therefore,

$$y_{n+1} = \left(1 + z\, \boldsymbol{b}^\top (I - zA)^{-1}\mathbf{1}\right) y_n.$$

Thus, the stability function for the Runge–Kutta method is

$$R(z) = 1 + z\, \boldsymbol{b}^\top (I - zA)^{-1}\mathbf{1}.$$

Let us consider an example. Recall the RK4 method we introduced before:

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

For this method, we have

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{6} \end{pmatrix}, \qquad \mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

To compute $R(z) = 1 + z\, \boldsymbol{b}^\top (I - zA)^{-1}\mathbf{1}$, let us denote

$$\boldsymbol{v} := (I - zA)^{-1}\mathbf{1}, \qquad \text{i.e.} \qquad (I - zA)\boldsymbol{v} = \mathbf{1}.$$

Since

$$I - zA = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{z}{2} & 1 & 0 & 0 \\ 0 & -\frac{z}{2} & 1 & 0 \\ 0 & 0 & -z & 1 \end{pmatrix}$$

is lower triangular, we can solve for $\boldsymbol{v} = (v_1, v_2, v_3, v_4)^\top$ sequentially:

$$\begin{aligned} v_1 &= 1, \\ -\frac{z}{2} v_1 + v_2 = 1 \quad &\Rightarrow \quad v_2 = 1 + \frac{z}{2}, \\ -\frac{z}{2} v_2 + v_3 = 1 \quad &\Rightarrow \quad v_3 = 1 + \frac{z}{2} + \frac{z^2}{4}, \\ -z\, v_3 + v_4 = 1 \quad &\Rightarrow \quad v_4 = 1 + z + \frac{z^2}{2} + \frac{z^3}{4}. \end{aligned}$$

Therefore,

$$\begin{aligned}\boldsymbol{b}^\top (I - zA)^{-1}\mathbf{1} = \boldsymbol{b}^\top \boldsymbol{v} &= \frac{1}{6}v_1 + \frac{1}{3}v_2 + \frac{1}{3}v_3 + \frac{1}{6}v_4 \\ &= 1 + \frac{z}{2} + \frac{z^2}{6} + \frac{z^3}{24}.\end{aligned}$$

Plugging this into the formula for $R(z)$, we obtain

$$\begin{aligned}R(z) &= 1 + z\left(1 + \frac{z}{2} + \frac{z^2}{6} + \frac{z^3}{24}\right) \\ &= 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}.\end{aligned}$$

Hence, the linear stability region for RK4 is

$$\mathcal{D}_{\text{RK4}} = \left\{ z \in \mathbb{C} : \left| 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} \right| < 1 \right\}.$$

Recall that the linear stability region for RK1 (explicit Euler) is $|1 + z| < 1$. Thus RK4 is not only higher order (fourth vs. first), but it also has a substantially larger linear stability region than explicit Euler.

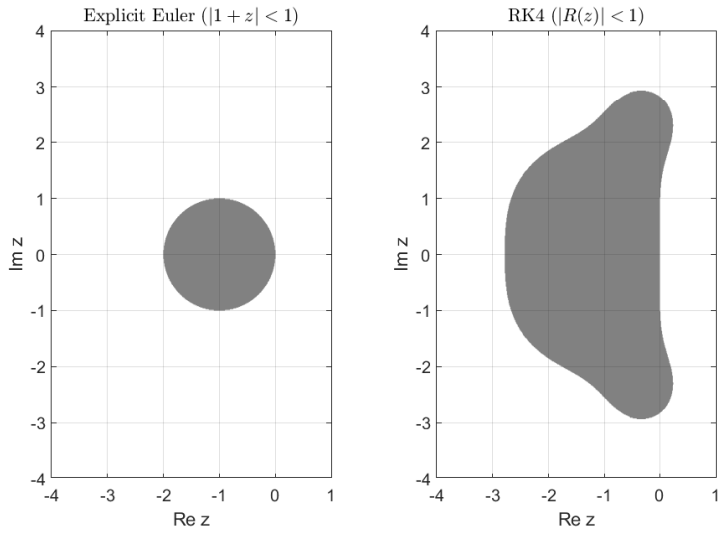


Figure 1: Visualization of the linear stability regions of explicit Euler and RK4.

Exercise. Derive and plot the linear stability regions for RK2 and RK3.

Linear stability region for Gauss-Legendre implicit Runge-Kutta. Recall that for implicit Runge–Kutta methods, we have learned about a special class called collocation IRK methods. Among them, we can choose the collocation points c_j such that the method has order 2ν, where ν is the number of stages. This method is also called the Gauss–Legendre IRK method since c_j are Gauss–Legendre quadrature points.

For these methods, we have the following result.

Theorem 5.3. *All Gauss–Legendre implicit RK methods are A-stable.*

Linear stability region for multistep methods. Let us first investigate the stability of the Adams–Bashforth method. Recall that the AB2 method is

$$y_i = y_{i-1} + h\left[-\frac{1}{2}f(t_{i-2}, y_{i-2}) + \frac{3}{2}f(t_{i-1}, y_{i-1})\right].$$

Note that $b_0 = -1/2$ and $b_1 = 3/2$ in this case.

Let $f = \lambda y$ and set $z := h\lambda$. Then

$$\begin{aligned} y_i &= y_{i-1} + h\left[-\frac{1}{2}\lambda y_{i-2} + \frac{3}{2}\lambda y_{i-1}\right] \\ &= \left(1 + \frac{3}{2}z\right) y_{i-1} - \frac{1}{2}z\, y_{i-2}. \end{aligned}$$

This is a linear difference equation. Seeking solutions of the form $y_i = \xi^i$, we obtain the characteristic equation

$$\xi^2 - \left(1 + \frac{3}{2}z\right)\xi + \frac{1}{2}z = 0.$$

Denote its two roots by $\xi_1(z)$ and $\xi_2(z)$. Then, the method is stable at a given $z \in \mathbb{C}$ if

$$|\xi_1(z)| < 1 \qquad \text{and} \qquad |\xi_2(z)| < 1.$$

Therefore, the linear stability region of AB2 is

$$\mathcal{D}_{\text{AB2}} := \left\{ z \in \mathbb{C} : \ \xi^2 - \left(1 + \frac{3}{2}z\right)\xi + \frac{1}{2}z = 0 \text{ has both roots strictly inside the unit disk} \right\}.$$

To plot the stability region, it is convenient to parameterize its boundary curve. Formally, points on the boundary correspond to the case where one root lies on the unit circle, i.e. $|\xi| = 1$. Let $\xi = e^{\mathrm{i}\theta}$ with $\theta \in [0, 2\pi]$. Solving the characteristic equation for z gives

$$\xi^2 - \left(1 + \frac{3}{2}z\right)\xi + \frac{1}{2}z = 0 \quad \Longrightarrow \quad \xi^2 - \xi - \frac{3}{2}z\,\xi + \frac{1}{2}z = 0 \quad \Longrightarrow \quad z = \frac{2(\xi^2 - \xi)}{3\xi - 1}.$$

Hence, the boundary of the stability region is parameterized by

$$z(\theta) = \frac{2\left(e^{2\mathrm{i}\theta} - e^{\mathrm{i}\theta}\right)}{3e^{\mathrm{i}\theta} - 1}, \qquad \theta \in [0, 2\pi].$$

Exercise. Derive the linear stability region for the AB3 method.

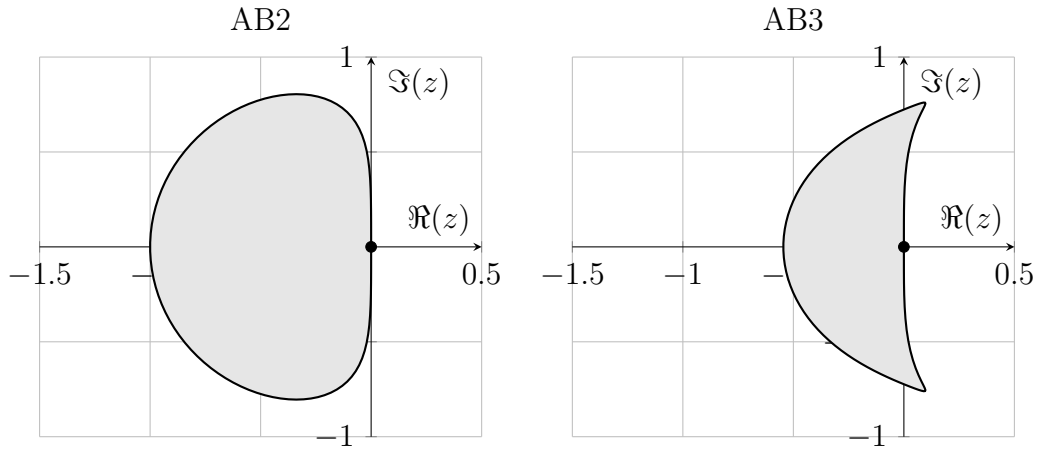


Figure 2: Linear stability regions of AB2 and AB3 (shaded).

Finally, let us consider the linear stability of the backward differentiation methods (BDF). Here we present the linear stability regions for BDF2, BDF3, and BDF4.

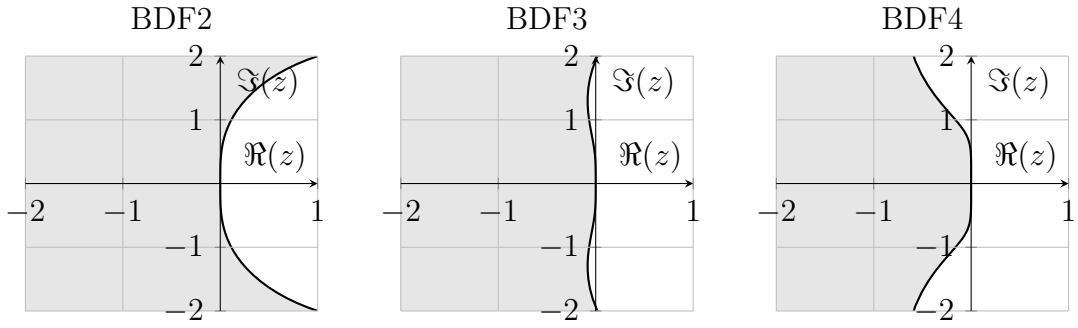


Figure 3: Linear stability regions of BDF2–BDF4 (shaded).

Note that for BDF3 and BDF4, their linear stability regions do not include the left complex half-plane. Thus, they are not A-stable. In fact, we have the following result about BDF methods.

Theorem 5.4. *The only A-stable BDF method is BDF2.*

6 Symplectic integrator

Consider a classical Hamiltonian system with positions $\boldsymbol{q}(t) \in \mathbb{R}^N$ and momenta $\boldsymbol{p}(t) \in \mathbb{R}^N$. The dynamics are governed by Hamilton's equations:

$$\dot{\boldsymbol{q}} = \frac{\partial \mathcal{H}(\boldsymbol{q}, \boldsymbol{p})}{\partial \boldsymbol{p}}, \qquad \dot{\boldsymbol{p}} = -\frac{\partial \mathcal{H}(\boldsymbol{q}, \boldsymbol{p})}{\partial \boldsymbol{q}}.$$

Here, the Hamiltonian $\mathcal{H}$ represents the total energy of the system. We can also write the above system in the ODE form (1) as

$$\begin{pmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{pmatrix} = \boldsymbol{f}\big(t, (\boldsymbol{q}, \boldsymbol{p})\big) = \begin{pmatrix} \frac{\partial \mathcal{H}(\boldsymbol{q},\boldsymbol{p})}{\partial \boldsymbol{p}} \\ -\frac{\partial \mathcal{H}(\boldsymbol{q},\boldsymbol{p})}{\partial \boldsymbol{q}} \end{pmatrix}.$$

Examples of Hamiltonian systems include planetary systems, frictionless pendulums, and classical molecular dynamics systems.

Hamiltonian systems satisfy several important structural properties. In particular, the total energy, represented by the Hamiltonian $\mathcal{H}$, is preserved in time. To see this, we take the time derivative of $\mathcal{H}(\boldsymbol{q}, \boldsymbol{p})$:

$$\begin{aligned} \frac{d\mathcal{H}(\boldsymbol{q},\boldsymbol{p})}{dt} &= \frac{\partial \mathcal{H}}{\partial \boldsymbol{q}} \frac{d\boldsymbol{q}}{dt} + \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} \frac{d\boldsymbol{p}}{dt} \\ &= \frac{\partial \mathcal{H}}{\partial \boldsymbol{q}} \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} - \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} \frac{\partial \mathcal{H}}{\partial \boldsymbol{q}} = 0. \end{aligned}$$

Thus, $\mathcal{H} \equiv C$ is constant in time.

In addition to energy conservation, Hamiltonian systems often preserve other physical quantities, such as linear momentum and angular momentum (exercises). It is therefore natural to ask whether numerical methods can be designed to respect these structural properties at the discrete level.

To illustrate the importance of energy- or Hamiltonian-preserving methods, we consider the canonical example of the n-body problem with Newtonian gravitation. We take the Hamiltonian in separable form,

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) := K(\boldsymbol{p}) + V(\boldsymbol{q}),$$

where the kinetic energy is given by

$$K(\boldsymbol{p}) := \frac{1}{2} \sum_{i=1}^{n} \frac{\|\boldsymbol{p}_i\|^2}{m_i},$$

and $V(\boldsymbol{q})$ denotes the potential energy. For Newtonian gravitation, we define

$$V(\boldsymbol{q}) := -G \sum_{1 \le i < j \le n} \frac{m_i m_j}{\|\boldsymbol{q}_i - \boldsymbol{q}_j\|},$$

where G is a constant, and m_i represents the mass of the i-th body.

Example. Consider a two-body system with $m_1 = 1$ and $m_2 = 10$ under Newtonian gravitation. Let $G = 1$ (non-dimensional units). We prescribe the initial positions and velocities

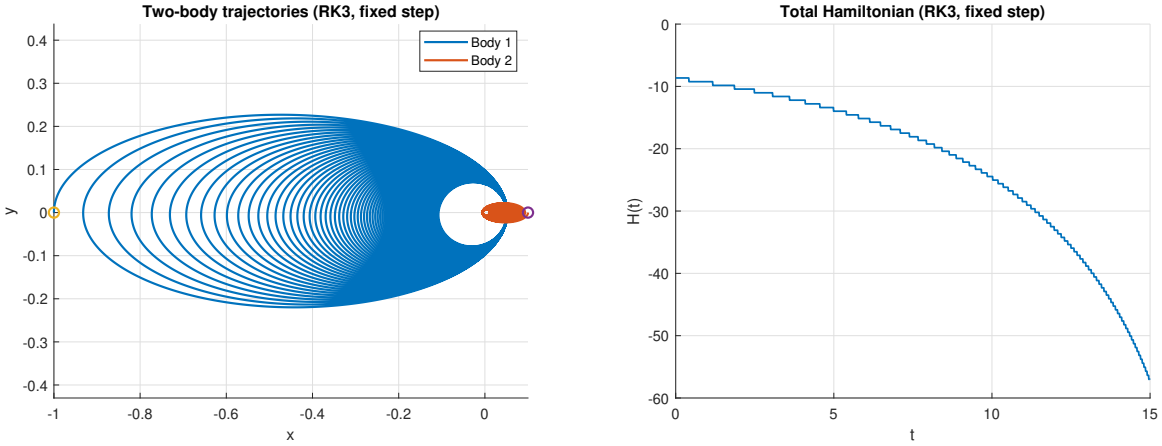
$$\boldsymbol{q}_1(0) = (-1, 0), \quad \boldsymbol{q}_2(0) = (0.1, 0), \qquad \boldsymbol{v}_1(0) = (0,\, 0.9), \quad \boldsymbol{v}_2(0) = \Big(0,\, -\frac{m_1}{m_2}\, 0.9\Big),$$

so that the total linear momentum satisfies $m_1\boldsymbol{v}_1(0) + m_2\boldsymbol{v}_2(0) = \mathbf{0}$ and the center of mass is (approximately) at rest.

The equations of motion are

$$\dot{\boldsymbol{q}}_i = \boldsymbol{v}_i, \qquad \dot{\boldsymbol{v}}_1 = -Gm_2\frac{\boldsymbol{q}_1 - \boldsymbol{q}_2}{\|\boldsymbol{q}_1 - \boldsymbol{q}_2\|^3}, \qquad \dot{\boldsymbol{v}}_2 = -Gm_1\frac{\boldsymbol{q}_2 - \boldsymbol{q}_1}{\|\boldsymbol{q}_2 - \boldsymbol{q}_1\|^3}.$$

We integrate these ODEs over the time interval $t \in [0, T]$ and plot the resulting trajectories using a third-order Runge–Kutta (RK3) scheme.



We observe that the total energy decreases over time, and the two bodies eventually spiral toward collision. This behavior contradicts the true physics of the system, in which the total energy should remain constant. This motivates the development of numerical methods for Hamiltonian ODE systems that preserve the Hamiltonian, and possibly other structural invariants, over long-time integration.

In what follows, we consider several so-called *symplectic* time-stepping methods. We will not go into the full theory of symplectic integrators; instead, we emphasize their key practical feature for Hamiltonian dynamics: they typically exhibit near-conservation of invariants (most notably the energy) over long time intervals, without systematic drift.

Symplectic Euler method. The first symplectic method we consider is the *symplectic Euler method*, defined by

$$\begin{cases} \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\nabla_p\mathcal{H}(\boldsymbol{q}_n, \boldsymbol{p}_{n+1}), \\ \boldsymbol{p}_{n+1} &= \boldsymbol{p}_n - h\nabla_q\mathcal{H}(\boldsymbol{q}_n, \boldsymbol{p}_{n+1}), \end{cases} \quad \text{or} \quad \begin{cases} \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\nabla_p\mathcal{H}(\boldsymbol{q}_{n+1}, \boldsymbol{p}_n), \\ \boldsymbol{p}_{n+1} &= \boldsymbol{p}_n - h\nabla_q\mathcal{H}(\boldsymbol{q}_{n+1}, \boldsymbol{p}_n). \end{cases}$$

These schemes can be viewed as mixed implicit–explicit methods. In general, one needs to solve a nonlinear system at each time step. For example, in the left formulation, the update requires solving for $\boldsymbol{p}_{n+1}$.

In certain cases, however, the method reduces to a fully explicit scheme. A particularly important example is when the Hamiltonian is *separable*, that is,

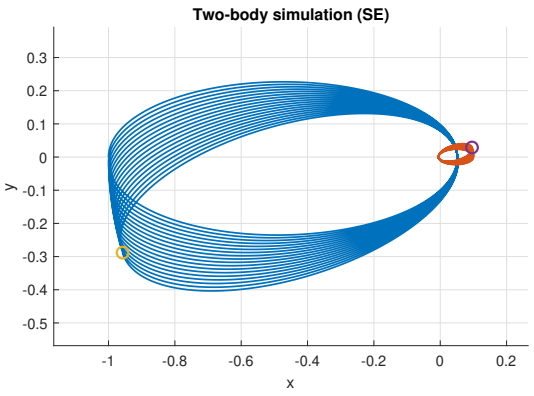
$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = K(\boldsymbol{p}) + V(\boldsymbol{q}).$$

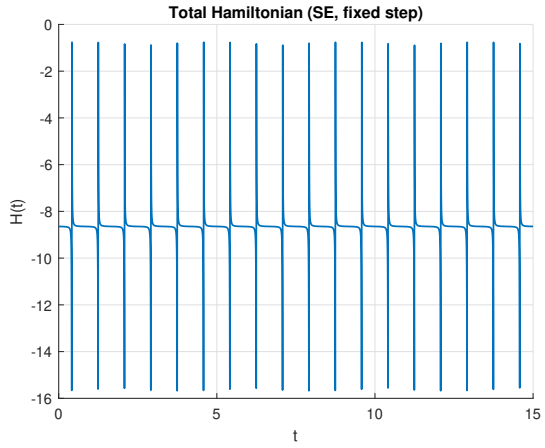
In this case, the symplectic Euler method becomes

$$\begin{cases} \boldsymbol{p}_{n+1} &= \boldsymbol{p}_n - h\nabla V(\boldsymbol{q}_n), \\ \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\nabla K(\boldsymbol{p}_{n+1}), \end{cases} \quad \text{or} \quad \begin{cases} \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\nabla K(\boldsymbol{p}_n), \\ \boldsymbol{p}_{n+1} &= \boldsymbol{p}_n - h\nabla V(\boldsymbol{q}_{n+1}). \end{cases} \tag{21}$$

Fortunately, many Hamiltonian systems of practical interest are separable, including n-body problems with Newtonian gravitational potential.

Let us revisit the two-body example below. The following numerical results are obtained using the symplectic Euler method.





One can see that, even though the total energy is not preserved exactly, but it oscillates around a constant value and does not exhibit long-term drift.

Verlet method. Another popular and widely used symplectic method is the *Verlet method*. It can be interpreted as a composition of two symplectic Euler steps: first advancing the system by a half time step using one formulation, and then advancing another half time step using the complementary formulation.

We again restrict attention to the separable Hamiltonian

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = K(\boldsymbol{p}) + V(\boldsymbol{q}).$$

From t_n to $t_{n+1/2}$, we apply the left symplectic Euler scheme (21):

$$\begin{aligned} \boldsymbol{p}_{n+1/2} &= \boldsymbol{p}_n - \frac{h}{2}\nabla V(\boldsymbol{q}_n), \\ \boldsymbol{q}_{n+1/2} &= \boldsymbol{q}_n + \frac{h}{2}\nabla K(\boldsymbol{p}_{n+1/2}). \end{aligned}$$

Then, from $t_{n+1/2}$ to t_{n+1}, we apply the right symplectic Euler scheme:

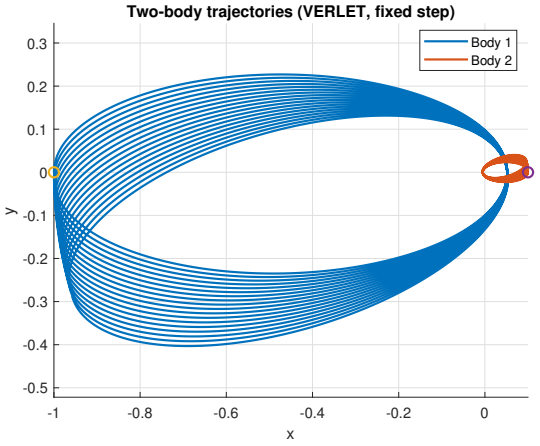
$$\begin{aligned} \boldsymbol{q}_{n+1} &= \boldsymbol{q}_{n+1/2} + \frac{h}{2}\nabla K(\boldsymbol{p}_{n+1/2}), \\ \boldsymbol{p}_{n+1} &= \boldsymbol{p}_{n+1/2} - \frac{h}{2}\nabla V(\boldsymbol{q}_{n+1}). \end{aligned}$$

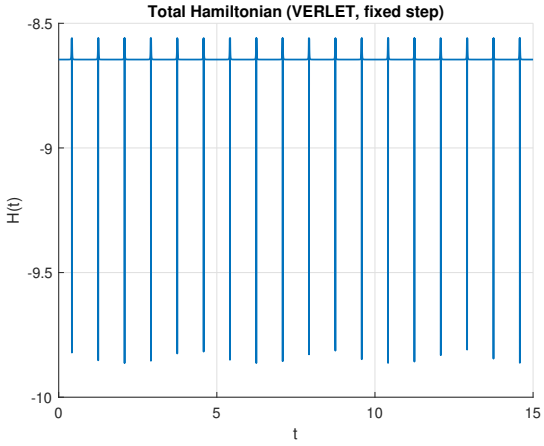
Combining these updates yields the standard *(velocity) Verlet method*:

$$\begin{aligned}\boldsymbol{p}_{n+1/2} &= \boldsymbol{p}_n - \frac{h}{2}\nabla V(\boldsymbol{q}_n),\\ \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\,\nabla K(\boldsymbol{p}_{n+1/2}),\\ \boldsymbol{p}_{n+1} &= \boldsymbol{p}_{n+1/2} - \frac{h}{2}\nabla V(\boldsymbol{q}_{n+1}).\end{aligned}$$

In words, we first take a half step for the momentum to obtain $\boldsymbol{p}_{n+1/2}$. We then use this intermediate momentum to update the position from $\boldsymbol{q}_n$ to $\boldsymbol{q}_{n+1}$. Finally, we complete the step by taking another half step for the momentum, using the updated position $\boldsymbol{q}_{n+1}$. This method is also referred to as the *Störmer–Verlet method*.

The following numerical results are obtained using the Verlet method. One can see that, although the total energy is not preserved exactly, it remains bounded and oscillates around a nearly constant value without systematic drift.





Moreover, compared with the symplectic Euler method, the energy oscillations are noticeably smaller, consistent with the higher-order accuracy of Verlet. Indeed, the symplectic Euler method is first-order accurate, whereas the Verlet method is second-order accurate.

Owing to its simplicity, symplecticity, and second-order accuracy, the Verlet method is among the most widely used symplectic integrators for Hamiltonian ODE systems.

What does symplectic mean? We have seen that both symplectic Euler and the Verlet method preserve certain structural properties of Hamiltonian systems. We now make precise what "symplectic" means.

To motivate the discussion, let us first consider perhaps the simplest Hamiltonian system with scalar position q and momentum p:

$$\dot{q} = \frac{\partial \mathcal{H}(q,p)}{\partial p}, \qquad \dot{p} = -\frac{\partial \mathcal{H}(q,p)}{\partial q}.$$

Now suppose that at $t = 0$ we are given a probability density $\rho_0(q,p)$ describing uncertainty in the initial phase point (q,p). Let $\rho(t,q,p)$ denote the density transported by the

Hamiltonian dynamics. The phase-space velocity field is

$$\boldsymbol{v}(q,p) := \begin{pmatrix} \frac{\partial \mathcal{H}(q,p)}{\partial p} \\ -\frac{\partial \mathcal{H}(q,p)}{\partial q} \end{pmatrix}.$$

It can be shown that the evolution of the probability density is governed by the *Liouville equation*:

$$\partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho) = 0, \qquad \nabla = (\partial_q, \partial_p),$$

with initial condition $\rho(q,p,0) = \rho_0(q,p)$. To see how the equation can be obtained. Let $\{\boldsymbol{z}_i(t)\}_{i=1}^N \subset \mathbb{R}^d$ be mass points with masses $m_i > 0$ transported by the velocity field

$$\dot{\boldsymbol{z}}_i(t) = \boldsymbol{v}(\boldsymbol{z}_i(t)).$$

Introduce the empirical density

$$\rho(\boldsymbol{z},t) := \sum_{i=1}^N m_i\, \delta(\boldsymbol{z} - \boldsymbol{z}_i(t))\,,$$

so that for any smooth test function φ with compact support, we have,

$$\langle \rho(\cdot,t), \varphi \rangle := \int_{\mathbb{R}^d} \varphi(\boldsymbol{z}) \rho(\boldsymbol{z},t)\, d\boldsymbol{z} = \sum_{i=1}^N m_i\, \varphi(\boldsymbol{z}_i(t)).$$

Differentiating in time and using the chain rule gives

$$\frac{d}{dt}\langle \rho, \varphi \rangle = \sum_{i=1}^N m_i\, \nabla\varphi(\boldsymbol{z}_i(t)) \cdot \dot{\boldsymbol{z}}_i(t) = \sum_{i=1}^N m_i\, \nabla\varphi(\boldsymbol{z}_i(t)) \cdot \boldsymbol{v}(\boldsymbol{z}_i(t),t).$$

Using the definition of ρ again, this may be rewritten as

$$\frac{d}{dt}\langle \rho, \varphi \rangle = \int_{\mathbb{R}^d} \nabla\varphi(\boldsymbol{z}) \cdot \boldsymbol{v}(\boldsymbol{z},t)\, \rho(\boldsymbol{z},t)\, d\boldsymbol{z}.$$

Since φ has compact support, integration by parts yields

$$\int_{\mathbb{R}^d} \nabla\varphi \cdot (\boldsymbol{v}\rho)\, d\boldsymbol{z} = -\int_{\mathbb{R}^d} \varphi\, \nabla \cdot (\boldsymbol{v}\rho)\, d\boldsymbol{z},$$

and therefore

$$\frac{d}{dt}\int_{\mathbb{R}^d} \varphi\, \rho\, d\boldsymbol{z} = -\int_{\mathbb{R}^d} \varphi\, \nabla \cdot (\boldsymbol{v}\rho)\, d\boldsymbol{z}.$$

Equivalently,

$$\int_{\mathbb{R}^d} \varphi \Big(\partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho) \Big)\, d\boldsymbol{z} = 0 \qquad \forall\, \varphi \in C_c^\infty(\mathbb{R}^d).$$

Hence, in the distributional (weak) sense,

$$\partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho) = 0.$$

A direct computation shows

$$\nabla \cdot \boldsymbol{v} = \partial_q\left(\frac{\partial \mathcal{H}}{\partial p}\right) + \partial_p\left(-\frac{\partial \mathcal{H}}{\partial q}\right) = \frac{\partial^2 \mathcal{H}}{\partial q\, \partial p} - \frac{\partial^2 \mathcal{H}}{\partial p\, \partial q} = 0.$$

Thus, the velocity field is divergence-free, and therefore the flow preserves phase-space area. This geometric intuition motivates the stronger notion of symplecticity introduced next.

A symplectic map is a map that preserves oriented areas in phase space. To make this precise, consider two vectors

$$\boldsymbol{a} = (a_q, a_p)^\top, \qquad \boldsymbol{b} = (b_q, b_p)^\top$$

based at a point (q, p). The (scalar) cross product

$$\boldsymbol{a} \times \boldsymbol{b} := a_q b_p - a_p b_q$$

represents the signed (oriented) area of the parallelogram spanned by $\boldsymbol{a}$ and $\boldsymbol{b}$ in the (q, p)-plane.

Let $\Psi : \mathbb{R}^2 \to \mathbb{R}^2$ be a smooth map and denote by $D\Psi(\boldsymbol{z})$ its Jacobian at $\boldsymbol{z} = (q, p)$. Locally, Ψ is dominated by the linear map $D\Psi(\boldsymbol{z})$, under which the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are mapped to $D\Psi(\boldsymbol{z})\boldsymbol{a}$ and $D\Psi(\boldsymbol{z})\boldsymbol{b}$, forming a new parallelogram. If

$$(D\Psi(\boldsymbol{z})\boldsymbol{a}) \times (D\Psi(\boldsymbol{z})\boldsymbol{b}) = \boldsymbol{a} \times \boldsymbol{b} \quad \text{for all } \boldsymbol{z} \in \mathbb{R}^2,\ \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^2,$$

or equivalently

$$(D\Psi(\boldsymbol{z}))^\top J\, D\Psi(\boldsymbol{z}) = J, \qquad J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

then the oriented area of every infinitesimal parallelogram is preserved. Such a map is called *symplectic*.

This notion extends to Hamiltonian systems with N degrees of freedom. Writing $\boldsymbol{z} = (\boldsymbol{q}, \boldsymbol{p}) \in \mathbb{R}^{2N}$ and

$$J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix},$$

a map $\Psi : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ is called symplectic if

$$(D\Psi(\boldsymbol{z}))^\top J\, D\Psi(\boldsymbol{z}) = J \quad \text{for all } \boldsymbol{z} \in \mathbb{R}^{2N}.$$

Hamiltonian flows are symplectic maps and therefore preserve the geometric structure of phase space. A one-step numerical method $\boldsymbol{z}_{n+1} = \Psi_h(\boldsymbol{z}_n)$ for a Hamiltonian system is called symplectic if its update map Ψ_h is symplectic for every step size h. Such methods preserve the geometry structure of the exact Hamiltonian flow and typically exhibit good long-time behavior.

The symplectic Euler method is one of the simplest symplectic integrators. For a separable Hamiltonian $\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + V(\boldsymbol{q})$, recall that the method reads $\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + V(\boldsymbol{q})$:

$$\boldsymbol{p}_{n+1} = \boldsymbol{p}_n - h\nabla V(\boldsymbol{q}_n), \qquad \boldsymbol{q}_{n+1} = \boldsymbol{q}_n + h\nabla T(\boldsymbol{p}_{n+1}).$$

This can be decomposed as two consecutive maps:

$$\Phi_V(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q},\ \boldsymbol{p} - h\nabla V(\boldsymbol{q})),$$
$$\Phi_T(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q} + h\nabla T(\boldsymbol{p}),\ \boldsymbol{p}).$$

Thus the full update map is $\Psi_h = \Phi_T \circ \Phi_V$.

Denote the Hessians

$$H_V(\boldsymbol{q}) = \nabla^2 V(\boldsymbol{q}), \qquad H_T(\boldsymbol{p}) = \nabla^2 T(\boldsymbol{p}).$$

Then, the Jacobians of the two steps are

$$D\Phi_V = \begin{pmatrix} I & 0 \\ -hH_V(\boldsymbol{q}) & I \end{pmatrix}, \qquad D\Phi_T = \begin{pmatrix} I & hH_T(\boldsymbol{p}) \\ 0 & I \end{pmatrix}.$$

We can verify that both Φ_T and Φ_V are symplectic:

$$(D\Phi_V)^\top J D\Phi_V = \begin{pmatrix} I & -hH_V \\ 0 & I \end{pmatrix}\begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}\begin{pmatrix} I & 0 \\ -hH_V & I \end{pmatrix} = J,$$
$$(D\Phi_T)^\top J D\Phi_T = \begin{pmatrix} I & 0 \\ hH_T & I \end{pmatrix}\begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}\begin{pmatrix} I & hH_T \\ 0 & I \end{pmatrix} = J,$$

Since the composition of symplectic maps is symplectic, the full update $\Psi_h = \Phi_T \circ \Phi_V$ is symplectic:

$$(D\Psi_h(\boldsymbol{z}))^\top J\, D\Psi_h(\boldsymbol{z}) = J.$$

7 Research project: numerical simulation of simplified planetary systems

In this project you will simulate a simplified gravitational multi-body system, such as Sun–Earth–Moon or Sun–Jupiter–Saturn, using Newton's law of gravitation. You may work in 2D, and you may use nondimensionalization or appropriate scaling to avoid excessively large or small numerical values.

1. **Method comparison.** Implement and compare several time-stepping methods, for example: Euler, Trapezoidal (implicit), Runge–Kutta methods, and symplectic methods (e.g., symplectic Euler or Verlet). Test multiple step sizes and (when applicable) different orders. Compare their performance in *long-time* simulations using diagnostics such as: energy error, and errors in linear and angular momentum. (These quantities are conserved in the ODE level.)
2. **Sensitivity analysis.** Apply a small perturbation to the initial condition (for instance, a 1%–5% change in position or velocity), and observe how the trajectory changes over time. Study how this sensitivity depends on the numerical method, step size, and (when applicable) the order of the method. Discuss whether qualitative changes (e.g., escape/collision) appear to be physical effects of the model or numerical artifacts.

Deliverables: A report and a 10–12 minute presentation summarizing your findings. You may use any tools you find helpful (including AI), but you must understand and be able to explain your implementation and results. Questions will be asked during your presentation.