Math 361S Lecture Notes
Numerical solution of ODEs

Holden Lee, Jeffrey Wong

April 1, 2020

Contents

1 Overview

In this section of the course we will derive methods to numerically solve ordinary differential equations (ODE's), analyze them, and gain experience with using them in practice. We'll apply what we have learned from interpolation, differentiation and integration. We will cover the following topics.

- **Basics**: We will focus on first-order ODE's, in standard form, and the problems we will consider are initial value problems (IVP's). How can we convert a higher-order ODE into a first-order ODE? How can we visualize the solution to an ODE?
- **Algorithms**: We will derive and analyze a variety of algorithms, such as forward and backward Euler, the family of Runge-Kutta methods, and multistep methods.
- **Convergence**: What is the relationship between local error and global error? How can we prove convergence? We will see that there are two ingredients: consistency and stability. How do we get quantitative estimates?
- **Stability**: Stability is one of the most important considerations in choosing an ODE solver. An important analysis is to find the region of stability for a numerical method. Stability is especially important for "stiff" ODEs.

  In practice, we will have to manage trade-offs between accuracy and stability.
- **Explicit vs. implicit methods**: Numerical methods can be classified as explicit and implicit. Implicit methods often have better stability properties, but require an extra step of solving non-linear equations using e.g., Newton's method.
- **Adaptive methods**: Similarly to integration, it is more efficient to vary the step size. To do this effectively, we need to derive methods with error estimates.
- **Using ODE solvers** in MATLAB and python: For example, `ode45` is an adaptive method in MATLAB that is a workhorse of solving ODE's, that often "just works." How do we use it? How can we choose which solver is appropriate for the problem? What are the tradeoffs?

See sections 7.1-7.3 of Moler's book or any standard text on ODEs for a review of ODEs. The bare minimum will be presented here. Essentially no ODE theory is required to solve ODEs numerically, but the theory does provide important intuition, so it will greatly enhance your understanding of the numerics.

2 Basics

In this section, you will learn:

- What is the **standard form** of a first-order ODE? How can we convert a higher-order ODE into a first-order ODE?
- What is an **initial value problem (IVP)** vs. a **boundary value problem (BVP)**? For IVPs, when do solutions exist and when are they unique?
- How does solving ODEs compare to integration? Integration can be viewed as a special case of solving an ODE.
- **Visualization**: Plot vector fields and trajectories.
- **Sensitivity**: Derive a bound on how solutions with differing initial conditions diverge.

2.1 First-order ODE; Initial value problems

We consider an ODE in the following standard form:

$$y'(t) = \frac{dy}{dt} = f(t, y), \quad t \in [a, b], \quad y(t) \in \mathbb{R}^d$$

where f is a function $f : [a, b] \times \mathbb{R}^d \to \mathbb{R}^d$. Here t is thought of as the independent variable, which can be time but does not have to be. Time is a useful analogy since it suggests the direction (forward in time) in which the ODE is to be solved. For simplicity, we will often consider the 1-D case when $y(t) \in \mathbb{R}$ ($y(t)$ is **scalar**)[1], but the theory extends to the general case. We say that this ODE is **first-order** because the highest derivative is first-order. Note the ODE does not have a unique solution until we impose some more conditions.

We will focus on solving **initial value problems (IVPs)** in the form

$$y'(t) = f(t, y), \quad t \in [a, b], \quad y(t) \in \mathbb{R}^d \tag{2.1a}$$

$$y(a) = y_0. \tag{2.1b}$$

The equation (2.1a) is the ODE for $y(t)$ and (2.1b) is the **initial condition**. We seek a function $y(t)$ that satisfies (2.1a) for all t in the given interval and whose value at a is y_0.

The following is a fundamental theorem about existence and uniqueness for ODE's.

Theorem 2.1. If $f : [a, b] \times \mathbb{R}^d \to \mathbb{R}^d$ is continuously differentiable, then in a neighborhood $[a, a + \varepsilon)$ around a, the solution to (2.1a)–(2.1b) exists and is unique.

Note that the solution may not exist for all $t \in [a, b]$ because the solution may diverge. An example is $y'(t) = y^2$, $y(0) = \frac{1}{c}$, which has the solution $y(t) = -\frac{1}{t-c}$ for $t < c$.

For our purposes, we will attempt to construct numerical solutions where the actual solution exists, so the theory is just there to ensure that the problem to solve is well-defined.

Throughout, we will further assume that f has partial derivatives of all orders required for any derivation (mostly for Taylor expansions).

2.2 Converting a higher-order ODE to a first-order ODE

If we have a nth-order ODE involving x, x',..., $x^{(n)}$, in the form

$$x^{(n)} = F(t, x, x', \ldots, x^{(n-1)}),$$

we can rewrite it as a first-order ODE as follows. First, define

$$y_1 = x, \quad y_2 = x', \ldots \quad y_n = x^{(n-1)}.$$

We can rewrite the nth-order ODE as a first-order ODE

$$\begin{aligned} y_1' &= y_2 \\ &\vdots \quad \vdots \\ y_{n-1}' &= y_n \\ y_n' &= F(t, y_1, \ldots, y_{n-1}). \end{aligned}$$

[1] We will often abuse notation slightly by writing $y \in \mathbb{R}$.

If $y(t) \in \mathbb{R}^d$, then we can collect $y_1, \ldots, y_n$ into a vector $\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \end{pmatrix} \in \mathbb{R}^{nd}$. Then letting $\mathbf{F}(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ \vdots \\ F(t, y_1, \ldots, y_{n-1}) \end{pmatrix}$, we have

$$\mathbf{y}' = \mathbf{F}(t, \mathbf{y})$$

To specify initial conditions for this problem, we need to specify the value of $y_1, \ldots, y_n$ at a, i.e., we need to specify the values of $y(a), y'(a), \ldots, y^{(n-1)}(a)$.

Example 2.2: For example, the ODE governing the motion of a pendulum (without air resistance, etc.) is

$$\theta'' = \frac{d^2\theta}{dt^2} = -g \sin(\theta)$$

where θ is the angle from the negative vertical axis and g is the gravitational constant. The initial conditions $\theta(0)$ and $\theta'(0)$ would give the initial position and velocity. The corresponding first-order ODE is

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -g \sin(y_1). \end{aligned}$$

This is in the form (2.1a)–(2.1b) with $y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$ and $f(t, y) = \begin{pmatrix} y_2 \\ -g \sin(y_1) \end{pmatrix}$.

The fact that we can rewrite higher-order ODE's as first-order ODE's means that it suffices to derive methods for first-order ODE's. Note that the standard ODE solvers for MATLAB require you to input a first-order ODE in standard form, so you will need to carry out this transformation before using it.

Problem 2.3: Write the ODE for the van der Pol oscillator

$$\frac{d^2}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0$$

as a first-order ODE in standard form.

2.3 Other types of differential equations

If conditions are given at more than one point, then the problem is a **boundary value problem (BVP)**. For an ODE, where the independent variable t is 1-dimensional, this means that conditions are given on both $y(a)$ and $y(b)$.

One common case of this is that for a second-order ODE, rather than giving the initial conditions $y(a) = y_0$ and $y'(a) = y'_0$, we are given the boundary conditions

$$y(a) = y_0 \qquad\qquad y(b) = y_1.$$

Unlike for IVPs, there is no simple existence and uniqueness theorem like Theorem 2.1. BVPs tend to be more challenging to solve numerically than IVPs, so we will not consider them here.

Finally, differential equations that involve more than one independent variable (and derivatives in those variables) are **partial differential equations (PDEs)**.

2.4 Comparison to integration

Integration is a special case of solving an ODE. To see this, note that by the fundamental theorem of calculus, the integral $F(t) = \int_a^t f(s)\,ds$ satisfies the ODE

$$F'(t) = f(t), \qquad\qquad F(a) = 0.$$

This is the special case when the function f depends only on a.

We can also make the comparison by looking at the integral form of an ODE. If

$$y'(t) = f(t, y), \qquad\qquad y(a) = y_0,$$

then $y(t)$ satisfies

$$y(t) = y_0 + \int_a^t f(s, y(s))\,ds.$$

Note two key differences with numerical integration:

1. The integrand $f(s, y(s))$ depends on the function value $y(s)$. This means that *any error in the current function value $y(s)$ propagates*. In contrast, for numerical integration, the errors on the different intervals are *independent*. This makes getting good error bounds for ODEs more challenging. Indeed, we have seen (Lemma 2.8) that error can accumulate *exponentially*.

2. When solving an ODE, we often want the *entire trajectory* $y(t)$, rather than just the value $y(b)$.

2.5 Visualization using vector fields

Slope fields are a good way to visualize the solution to an ODE.

Suppose we are given a scalar ODE ($y \in \mathbb{R}$)

$$y' = f(t, y).$$

A solution $(t, y(t))$ forms a curve in the (t, y) plane. The ODE tells us the direction of the curve at any given point, since

$$y(t) \text{ is parallel to } (1, y') = (1, f).$$

In this sense, solutions to the ODE follow the "slope field", which is the vector field

$$(1, f(t, y))$$

in the (t, y) plane. To find a solution to the IVP starting at (t_0, y_0), we may follow the slope field to construct the curve; this is the basis of the simplest numerical method that is detailed in the next section.

The MATLAB function for plotting vector fields is `quiver`.[2]

Example 2.4: An example of plotting the vector field for $y' = 0.5y$ on $[0, 2] \times [0, 4]$ is given below. Here, `t` and `y` contain the t and y-coordinates of the grid points, and `u`, `v` are the components of the vector field at those grid points. Make sure that `u`, `v` are defined as componentwise functions applied to `t`, `y` (for example, using `.*` for componentwise multiplication). The solution, $y = \frac{1}{2}e^{t^2/2}$ is also plotted.

```
[t,y] = meshgrid(0:0.2:2,0:0.5:4);
u = ones(size(t));
v = t.*y;

hold on
% Set window
xlim([0,2.2])
ylim([-0.25,4.25])
% Plot vector field
quiver(t,y,u,v, 'b')
% Also plot trajectory
T = linspace(0,2);
Y = 0.5*exp(T.^2/2);
plot(T,Y,'r');
```

Another case we can easily visualize is where $y \in \mathbb{R}^2$ and the ODE is **autonomous**, that is, not depending on t:

$$y' = f(y), \qquad y \in \mathbb{R}^2.$$

[2] Documentation: https://www.mathworks.com/help/matlab/ref/quiver.html

We can instead plot the vector field given by $f : \mathbb{R}^2 \to \mathbb{R}^2$, i.e., with the vector $f(y) \in \mathbb{R}^2$ at $y \in \mathbb{R}^2$.

Example 2.5: We plot the vector field corresponding to Example 2.2 on $[-\pi/2, \pi/2] \times [-2, 2]$. For simplicity, $g = 1$ here.

```
[x,y] = meshgrid(-pi/2:pi/8:pi/2,-2:0.5:2);
u = y;
v = -sin(x);

xlim([-1.7,1.7])
ylim([-2.2,2.2])
quiver(x,y,u,v,'b')
```

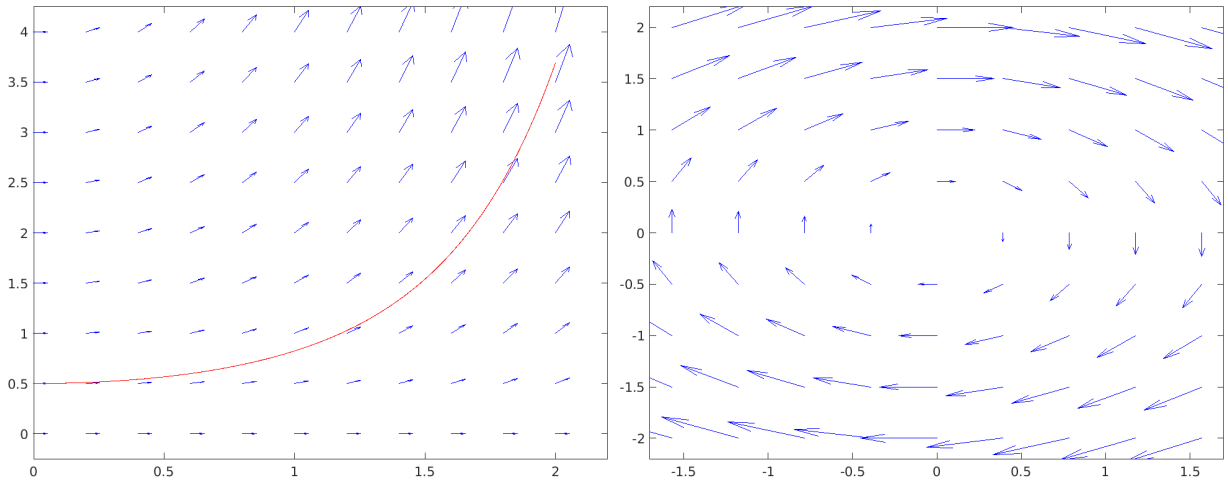


Figure 1: Outputs of Examples 2.4 and 2.5

Problem 2.6: For different values of μ, plot the vector field for the van der Pol oscillator in Problem 2.3.

2.6 Sensitivity

The slope field gives geometric intuition for some important concepts for numerical methods, such as the notion of **sensitivity**.

Sensitivity means: if $y(t)$ gets perturbed by some amount Δy at time t_0, how far apart are the original and perturbed trajectories after some time? Put another way, how sensitive is the ODE to changes in the initial condition?

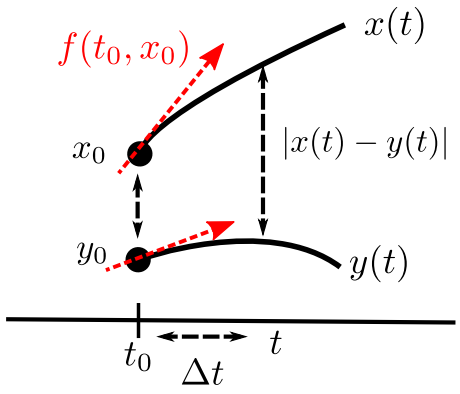


Figure 2: Sensitivity

Suppose we have two solutions $x(t)$ and $y(t)$ to the same ODE,

$$y' = f(t, y), \quad y(0) = y_0,$$

$$x' = f(t, x), \quad x(0) = x_0.$$

We will use a Lipschitz bound on f to derive a bound on the difference $z = y - x$.

Definition 2.7: Let $f : [a, b] \times \mathbb{R} \to \mathbb{R}$ be a function. We say that $f(t, y)$ is L**-Lipschitz** in y (or the **Lipschitz constant** in y is at most L) if

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \text{ for all } t \in [a, b] \text{ and } y_1, y_2 \in \mathbb{R}. \tag{2.2}$$

When f is differentiable, this is equivalent to

$$\max_{t\in[a,b], y\in\mathbb{R}} \left|\frac{\partial f}{\partial y}(t, y)\right| \leq L. \tag{2.3}$$

To see that (2.3) implies (2.2) note that by the mean value theorem, for some $\xi \in [y_1, y_2]$,

$$f(t, y_1) - f(t, y_2) = \frac{\partial f}{\partial y}(t, \xi)(y_1 - y_2),$$

and take the absolute value of both sides.[3]

Going back to the problem, if $f(t, y)$ is L-Lipschitz in y, then the difference $z = y - x$ satisfies

$$|z'(t)| = |f(t, y(t)) - f(t, x(t))| \leq L|y(t) - x(t)| = L|z(t)|. \tag{2.4}$$

To obtain a bound on $z(t)$ given $z(0)$, we use the following useful lemma.

Lemma 2.8 (Grönwall's Lemma)**.** Let $u \in C^1([0, t])$. If $u'(s) \leq Lu(s)$ for each $s \in [0, t]$, then $u(t) \leq u(0)e^{Lt}$.

[3]This proof only works in 1 dimension. For $y \in \mathbb{R}^d$, (2.3) becomes $\max_{t\in[a,b], y\in\mathbb{R}^d} |\nabla_y f(t, y)| \leq L$. We can conclude (2.2) by $f(t, y_1) - f(t, y_2) = \int_0^1 \langle \nabla_y f\,(t, y_1 + s(y_2 - y_1))\,, y_2 - y_1\rangle\, ds$.

Proof. We show that $u(t)e^{-Lt}$ is decreasing. This follows from the product rule and the assumption $u' \le Lu$:

$$\frac{d}{dt}(ue^{-Lt}) = \frac{du}{dt}e^{-Lt} - Lue^{-Lut} = \left(\frac{du}{dt} - Lu\right)e^{-Lt} \le 0.$$

Thus $u(t)e^{-Lt} \le u(0)$, giving the conclusion. □

Applying Lemma 2.8 to u and $-u$, we get the following from (2.4):

$$|z'| \le L|z| \implies |z(t)| \le |z(0)|e^{Lt}$$

Thus L (the maximum of the variation of f with respect to y) is the exponential rate, at worst, at which the two solutions can move apart. We have proved the following.

Theorem 2.9 (Sensitivity of ODE)**.** Suppose

$$\begin{aligned} y' &= f(t,y), & y(0) &= y_0, \\ x' &= f(t,x), & x(0) &= x_0. \end{aligned}$$

are solutions to the same ODE with different initial conditions, where $f \in C^1([0,t] \times \mathbb{R})$. If $f(t,y)$ is L-Lipschitz in y, then

$$|x(t) - y(t)| \le e^{Lt}|x_0 - y_0|.$$

The idea of sensitivity is very useful: The different initial conditions can come from some "natural" perturbation, but they can also be error that is built up from previous steps of a numerical algorithm.

However, the bound in Theorem 2.9 is sometimes pessimistic. Taking absolute values discards information about the sign, so if $z' \approx -Lz$ then the bound is the same, even though z then decays exponentially. This is shown in the figure.

Problem 2.10 (Generalization of Lemma 2.8)**:** Prove the following.
Let $u \in C^1([0,t])$, and suppose $a \in C([0,t])$ is nonnegative. If $u'(s) \le a(s)u(s)$ for each $s \in [0,t]$, then $u(t) \le u(0)e^{\int_0^t a(s)\,ds}$.

3 Euler's method

In this section we derive and analyze the simplest numerical method for ODEs, **(forward) Euler's method**; we will also briefly consider the backward Euler's method. Through analyzing Euler's method, we introduce general concepts which are useful for understanding and analyzing all kinds of numerical methods. This includes:

- **Local truncation error**.

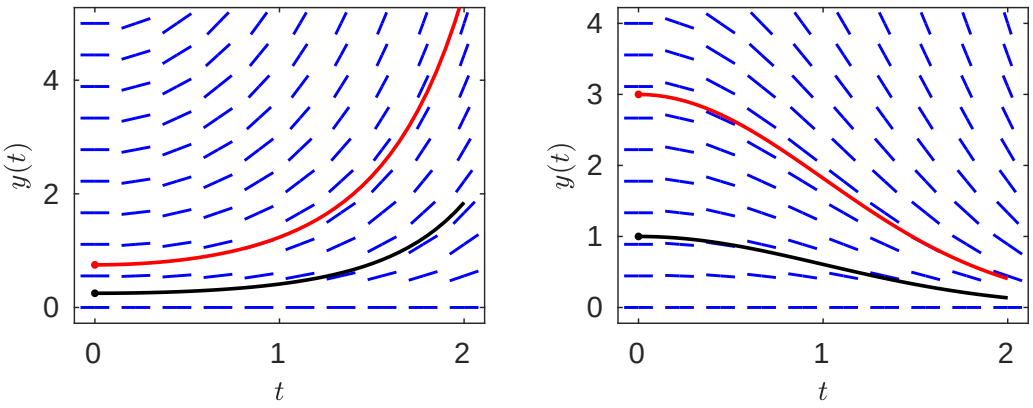


Figure 3: Sketch of the difference in two solutions that start at nearby points (t_0, x_0) and (t_0, y_0) and numerical examples for $y' = ty$ and $y' = -ty$.

- Proving **convergence** (for **global error**): After getting a bound for the local truncation error, use induction to give a quantitative bound for the global error.
- The **order** of a method.

This gives a general framework to analyze numerical methods. The goal is to be able to go through this analysis for other (better!) numerical methods as well. In the next section, we will elaborate on this framework.

We summarize our notation in the following box. They will be explained more as they are introduced.

Summary of notation in this section:

- $t_0, \ldots, t_N$: the points where the approximate solution is defined
- $y_j = y(t_j)$: the exact solution to the IVP
- $\tilde{y}_j$ the approximation at t_j
- τ_j: truncation error in obtaining $\tilde{y}_j$
- h or Δt: the "step size" $t_j - t_{j-1}$ (if it is constant); otherwise h_j or Δt_j

We seek a numerical solution $y(t)$ to the scalar IVP

$$y' = f(t, y), \qquad y(a) = y_0$$

(with $y \in \mathbb{R}$) up to a time $t = b$. The approximation will take the form of values $\tilde{y}_j$ defined on a grid

$$a = t_0 < t_1 < \cdots < t_N = b$$

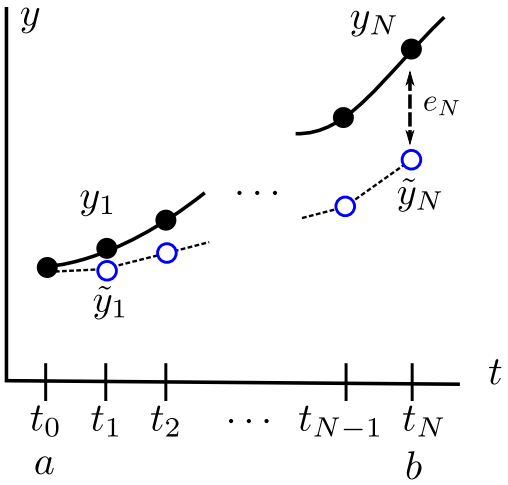


Figure 4: Numerical solution of an IVP forward in time from $t = a$ to $t = b$. The actual values are $y_1, \ldots, y_N$ and the estimated values are $\widetilde{y}_1, \ldots, \widetilde{y}_N$.

such that

$$\tilde{y}_j \approx y(t_j).$$

For convenience, denote by y_j the **exact solution** at t_j and let the "error" at each point be

$$e_j = y_j - \tilde{y}_j.$$

It will be assumed that we have a free choice of the t_j's. The situation is sketched in Figure 4.

3.1 Forward Euler's method

Assume for simplicity that the step size (for time)

$$h = t_j - t_{j-1},$$

is constant. This is not necessary, but is convenient.

Suppose that we have the exact value of $y(t)$. To get $y(t+h)$ from $y(t)$, expand in a Taylor series and use the ODE to simplify the derivatives:

$$\begin{aligned} y(t+h) &= y(t) + hy'(t) + O(h^2) \\ &= y(t) + hf(t,y) + O(h^2). \end{aligned}$$

This gives, for any point t_j, the formula

$$y(t_j + h) = y(t_j) + hf(t_j, y(t_j)) + \tau_{j+1} \tag{3.1}$$

where τ_{j+1} is the **local truncation error** defined below. We could derive a formula, but the important thing is that

$$\tau_{j+1} = O(h^2).$$

Dropping the error in (3.1) and iterating this formula, we get the **(forward) Euler's method**:

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_j, \widetilde{y}_j). \tag{3.2}$$

Algorithm 3.1 (Forward Euler's method)**:** The forward Euler's method for solving the IVP

$$y' = f(t, y), \qquad y(a) = y_0$$

is given by

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_j, \widetilde{y}_j).$$

The initial point is, ideally,

$$\widetilde{y}_0 = y_0$$

since the initial value (at t_0) is given. However, in practice, there may be some initial error in this quantity as well.

Definition 3.2 (Local truncation error, or LTE)**:** The **local truncation error** τ_{j+1} is the error incurred in obtaining $\widetilde{y}_{j+1}$ *when the previous data* $y_j = \widetilde{y}_j$ *is known exactly*:

$$y_{j+1} = \widetilde{y}_{j+1} + \tau_{j+1}$$

In other words, it is the amount by which the exact solution fails to satisfy the equation given by the numeric method.

The LTE is "local" in the sense that it does not include errors created at previous steps. For example, for Euler's method,

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_j, \widetilde{y}_j)$$

The LTE is the error between this and the true value of $\widetilde{y}_{j+1}$ when $\widetilde{y}_j = y_j$:

$$\begin{aligned} y_{j+1} &= \widetilde{y}_j + hf(t_j, \widetilde{y}_j) + \tau_{j+1} \\ &= y_j + hf(t_j, y_j) + \tau_{j+1}. \end{aligned}$$

! Notice that the total error is **not** just the sum of the truncation errors, because when starting the numeric method with y_0, at step j f is evaluated at the approximation $\widetilde{y}_j$, and $\widetilde{y}_j \neq y_j$. The truncation error will propagate through the iteration, as a careful analysis will show.

3.2 Convergence

Suppose we use Euler's method to generate an approximation

$$(t_j, \widetilde{y}_j), \quad j = 0, \ldots, N$$

to the solution $y(t)$ in the interval $[a, b]$ (with $t_0 = a$ and $t_N = b$). The "error" in the approximation that matters in practice is the **global error**

$$E = \max_{0 \le j \le N} |y_j - \widetilde{y}_j| = \max_{0 \le j \le N} |e_j|$$

where

$$e_j = y_j - \widetilde{y}_j$$

is the error at t_j. This is a measure of how well the approximation $\widetilde{y}_j$ agrees with the true solution over the whole interval.[4]

Definition 3.3: The method is **convergent** if the global error approaches 0 if h approaches 0.

More precisely, we would like to show that, given an interval $[a, b]$, the global error has the form

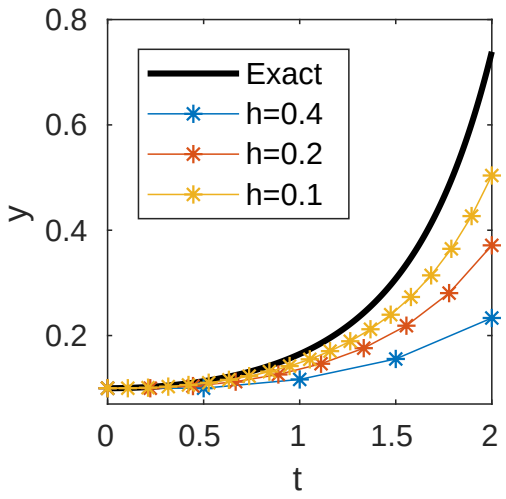
$$\max_{0 \le j \le N} |e_j| = O(h^p)$$

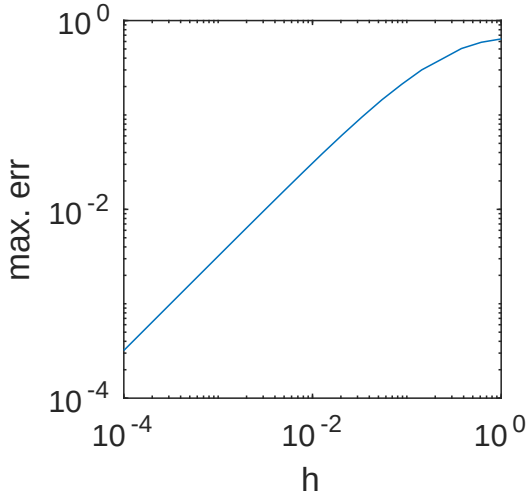
for some integer p, the **order** of the approximation.

As an example, consider

$$y' = ty, \qquad y(0) = 0.1$$

which has exact solution $y(t) = 0.1e^{t^2}$. Below, we plot some approximations for various time-steps h; on the right is the max. error in the interval. The log-log plot has a slope of 1, indicating the error should be $O(h)$.





[4]Note that this is not precisely true since the approximation is not defined for all t; we would need to interpolate and that would have its own error bounds. But in practice we typically consider error at the points where the approximation is computed.

The definition of convergent means that the approximation as a piecewise-defined function, e.g. piecewise linear, converges to $y(t)$ as $h \to 0$. Since the points get arbitrarily close together as $h \to 0$, the distinction between "max error at the t_j's" and "max error as functions" is not of much concern here.

3.3 Euler's method: error analysis

The details of the proof are instructive, as they will illustrate how error propagates in the "worst case". Assume that, as before, we have a fixed step size $h = t_j - t_{j-1}$, points

$$a = t_0 < t_1 < \cdots < t_N = b$$

and seek a solution in $[a, b]$.

We proceed in two steps: bounding the local truncation error, and then using tracking the accumulation of the local error to bound the global error.

3.3.1 Local truncation error

The bound for the local truncation error is a direct consequence of Taylor's formula.

Lemma 3.4. Suppose $t_{j+1} = t_j + h$, and the solution $y(t)$ to the scalar IVP

$$y' = f(t, y), \qquad y(t_j) = y_j$$

satisfies $\max_{[t_j, t_{j+1}]} |y''| \leq M$. Let the truncation error τ_{j+1} for Euler's method be defined by

$$y(t_{j+1}) = y(t_j) + hf(t_j, y_j) + \tau_{j+1}.$$

Then

$$|\tau_{j+1}| \leq \frac{Mh^2}{2}.$$

Proof. By Taylor's formula with remainder,

$$y(t_{j+1}) = y(t_j) + hy'(t_j) + \frac{h^2}{2} y''(\xi)$$

for some $\xi \in [t_j, t_{j+1}]$. Now use the fact that $y'(t_j) = f(t_j, y_j)$ and $|y''(\xi)| \leq M$ to conclude the bound. □

3.3.2 From local to global error

Given a bound on the local truncation errors, we can obtain a bound on the global error.

Lemma 3.5 (Euler's method: from local to global error)**.** Suppose that $f(t, y)$ is L-Lipschitz in y, and let τ_j be the local truncation error at step j. Let $\tilde{y}_j$ be the result of applying Euler's method (3.2) starting at $(t_0 = a, \tilde{y}_0)$. Then

$$|e_j| \leq e^{L(t_j - t_0)} |e_0| + \left(\frac{e^{L(t_j - t_0)} - 1}{L} \right) \frac{\max |\tau_k|}{h} \tag{3.3}$$

$$\max_{0 \leq j \leq n} |e_j| \leq e^{L(b-a)} |e_0| + \left(\frac{e^{L(b-a)} - 1}{L} \right) \frac{\max |\tau_k|}{h}. \tag{3.4}$$

where τ_k is the local truncation error and $e_j = y_j - \tilde{y}_j$ is the error at t_j.

In particular, $\max \tau_j = O(h^2)$ as $h \to 0$, and $e_0 = 0$ (no error in y_0) then

$$\max_{0 \le j \le n} |e_j| = O(h) \tag{3.5}$$

as $h \to 0$, where the constant in the $O(\cdot)$ depends on the interval size and L but not on h. Note also that the amplification of the initial error e_0 is similar to in Theorem 2.9.

Combining Lemmas 3.4 and 3.5, we obtain the following.

Theorem 3.6 (Convergence of Euler's method)**.** Suppose:

1. The actual solution $y(t)$ satisfies $\max_{[a,b]} |y''| \le M$.
2. $f(t, y)$ is L-Lipschitz in y.

Let $\tilde{y}_j$ be the result of applying Euler's method (3.2) starting at $(t_0 = a, \tilde{y}_0 = y_0)$. Then

$$|e_j| \le \frac{Mh}{2L}[e^{L(t_j - t_0)} - 1] \tag{3.6}$$

$$\max_{0 \le j \le n} |e_j| \le \frac{Mh}{2L}[e^{L(b-a)} - 1]. \tag{3.7}$$

where $e_j = y_j - \tilde{y}_j$ is the error at t_j.

! The global error is $O(h)$ as $h \to 0$, but the O-notation hides a large constant that is *exponential* in $L(b-a)$, and which is independent of h.
As with the bound on sensitivity, this bound can be quite pessimistic.

Proof of Lemma 3.5. To start, recall that from the definition of the truncation error and the formula,

$$y_{j+1} = y_j + hf(t_j, y_j) + \tau_{j+1} \tag{3.8}$$

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_j, \widetilde{y}_j) \tag{3.9}$$

Subtracting (3.9) from (3.8) gives

$$e_{j+1} = e_j + h\Big(f(t_j, y_j) - f(t_j, \widetilde{y}_j)\Big) + \tau_{j+1}.$$

Because f is L-Lipschitz in y, we have $h|f(t_j, y_j) - f(t_j, \widetilde{y}_j)| \le h(L|y_j - \widetilde{y}_j|) = Lh|e_j|$. By the triangle inequality,

$$|e_{j+1}| \le (1 + Lh)|e_j| + |\tau_{j+1}|.$$

Iterating, we get

$$|e_1| \le (1 + Lh)|e_0| + |\tau_1|,$$
$$|e_2| \le (1 + Lh)^2|e_0| + (1 + Lh)|\tau_1| + |\tau_2|$$

and in general

$$|e_j| \le (1+Lh)^j|e_0| + (1+Lh)^{j-1}\tau_1 + \cdots + (1+Lh)\tau_{j-1} + \tau_j$$
$$= (1+Lh)^j|e_0| + \sum_{k=1}^{j}(1+Lh)^{j-k}|\tau_k|.$$

Bounding each $|\tau_k|$ by the maximum and evaluating the sum,

$$|e_j| \le (1+Lh)^j|e_0| + (\max|\tau_k|)\,\frac{(1+Lh)^j - 1}{Lh}.$$

Now we want to take the maximum over j, so the RHS must be written to be independent of j. To fix this problem, we use the crude estimate

$$1 + Lh \le e^{Lh}$$

to obtain

$$|e_j| \le e^{Ljh}|e_0| + \left(\frac{e^{Ljh}-1}{Lh}\right)\max|\tau_k|$$

But $jh = t_j - t_0 \le b - a$ (equal when $j = N$) so

$$|e_j| \le e^{L(b-a)}|e_0| + \left(\frac{e^{L(b-a)}-1}{L}\right)\frac{\max|\tau_k|}{h}.$$

Taking the maximum over j (note that the RHS is independent of j) we get (3.7). □

Proof of Theorem 3.6. By Lemma 3.4, the local truncation errors satisfy $|\tau_j| \le \frac{Mh^2}{2}$. Hence $\frac{\max|\tau_k|}{h} \le \frac{Mh}{2}$. Plug this into Lemma 3.5. □

3.4 Interpreting the error bound

A few observations on what the error bound tells us:

- The LTE introduced at each step grows at a rate of e^{Lt} at worst.
- An initial error (in y_0) also propagates in the same way.
- The LTE is $O(h^2)$ and $O(1/h)$ steps are taken, so the total error is $O(h)$; the propagation does not affect the order of the error on a finite interval.

Note that the factor L and $(1 + Lh)$ are method dependent; for other methods, the factors may be other expressions related to L.

Our two examples

$$\text{(a) } y' = ty, \qquad \text{(b) } y' = -ty$$

illustrate the propagation issue. As with actual solutions, the error and a numerical solution (or two nearby numerical solutions), can grow like e^{Lt} at worst. Indeed, for (a), the error grows in this way; the error bound is good here.

However, for (b), the numerical solutions actually converge to the true solution as t increases; in fact we have that the error behaves more like e^{-Lt}. But the error bound cannot distinguish between the two cases, so it is pessimistic for (b).

In either case, the global error over $[a, b]$ is $O(h)$ as $h \to 0$.

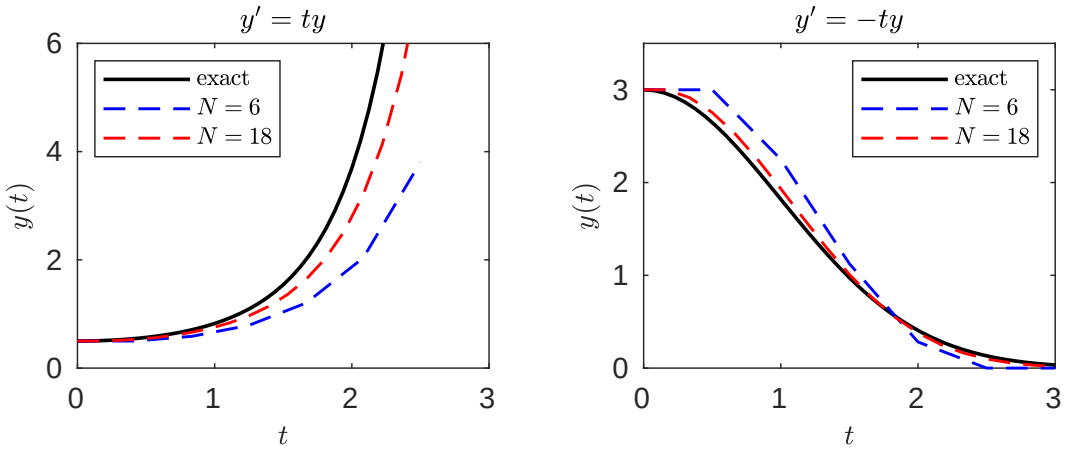


Figure 5: Numerical solutions to $y' = ty$ and $y' = -ty$ with different values of N; note the behavior of the error as t increases.

3.5 Order

The **order** p of a numerical method for an ODE is the order of the **global** truncation error. Euler's method, for instance, has order 1 since the global error is $O(h)$.

Definition 3.7 (Order)**:** A numerical method with timestep h is said to be convergent with **order** p if, on an interval $[a, b]$,

$$\max_{0 \le j \le n} |\widetilde{y}_j - y(t_j)| = O(h^p) \text{ as } h \to 0.$$

The $1/h$ factor is true for (most) other methods, so as a rule of thumb,

$$\text{LTE} = O(h^{p+1}) \implies \text{global error} = O(h^p).$$

The interpretation here is that to get from a to b we take $\sim 1/h$ steps, so the error is on the order of the number of steps times the error at each step, $(1/h) \cdot O(h^{p+1}) = O(h^p)$. The careful analysis shows that the order is not further worsened by the *propagation* of the errors.

! Some texts define the LTE with an extra factor of h so that it lines up with the global error, in which case the rule is that the LTE and global error have the same order. For this reason it is safest to say that the error is $O(h^p)$ rather than to use the term "order p", but either is fine in this class.

3.6 Backward Euler's method

The forward Euler's method is an **explicit** method: the approximation $\widetilde{y}_{j+1}$ can be evaluated directly in terms of $\widetilde{y}_j$ and other known quantities. In contrast, a **implicit** method is one

where we are only given a (nonlinear) equation that is satisfied by $\widetilde{y}_{j+1}$, and we have to solve for $\widetilde{y}_{j+1}$.

The simplest example of an implicit method is backward Euler, which has the iteration

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_{j+1}, \widetilde{y}_{j+1}).$$

Note the function is evaluated at $\widetilde{y}_{j+1}$ rather than $\widetilde{y}_j$ as in the forward Euler's method. To solve for $\widetilde{y}_{j+1}$, we can iterate Newton's method until convergence; $\widetilde{y}_j$ makes for a good initial guess.

Using a "backward" Taylor expansion around t_{j+1} rather than around t_j, you can prove that the LTE is also $O(h^2)$.

This seems more complicated; why would you want to use the backward method? It turns out that it has better stability properties; we will come back to this point.

Algorithm 3.8 (Backward Euler's method)**:** The backward Euler's method is given by

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_{j+1}, \widetilde{y}_{j+1}). \tag{3.10}$$

Here, you can solve for $\widetilde{y}_{j+1}$ using Newton's method with $\widetilde{y}_j$ as an initial guess.

If Newton's method is used, the code must know f and $\partial f/\partial y$. The function in Matlab may be written, for instance, in the form

```
[T,Y] = beuler(f,fy,[a b],y_0,h)
```

where `f,fy` are both functions of t and y. At step j, We would like to set $\widetilde{y}_{j+1}$ equal to the zero of

$$g(z) = z - \widetilde{y}_j - hf(t_{j+1}, z).$$

We compute

$$g'(z) = 1 - hf_y(t_{j+1}, z)$$

so the Newton iteration is

$$z_{k+1} = z_k - \frac{g(z_k)}{g'(z_k)}.$$

This is iterated until convergence; then $\widetilde{y}_{j+1}$ is set to the resulting z.

Problem 3.9: Prove an analogue of Theorem 3.6 for the backward Euler's method. You can assume that the numeric solution $\widetilde{y}_{j+1}$ obtained from (3.10) is exact.

4 Consistency, stability, and convergence

Convergence is the property any good numerical method for an ODE must have. However, as the proof shows, it takes some effort (and is harder for more complicated methods). To

better understand it, we must identify the key properties that guarantee convergence and how they can be controlled.

This strategy has two steps: showing **consistency** and **stability**. Both are necessary for convergence. We'll look at each of these notions in turn, and give an informal "theorem" which says that they are sufficient for convergence. Finally, we'll consider an example of what can go wrong: a method which seems reasonable at first glance but is disastrously unstable.

As before, we solve an IVP in an interval $[a, b]$ starting at $t_0 = a$ with step size h.

4.1 Consistency

A method is called **consistent** if

$$\text{the LTE at each step is } o(h) \text{ as } h \to 0.$$

That is,

$$\lim_{h \to 0} \frac{\tau_j}{h} = 0 \text{ for all } j.$$

To check consistency, we may assume **the result of the previous step is exact** (since this is how the LTE is defined). This is a benefit, as there is no need to worry about the accumulation of errors at earlier steps.

Example (Checking consistency): Euler's method,

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_j, \widetilde{y}_j)$$

is consistent since the truncation error is

$$\tau_{j+1} = y(t_{j+1}) - y(t_j) - hf(t_j, y(t_j)) = \frac{h^2}{2} y''(\xi_j)$$

where ξ_j lies between t_j and t_{j+1}. For each j, the error is $O(h^2)$ as $h \to 0$. From the point of view of consistency, j is fixed so $y''(\xi_j)$ is just some constant; we do not need a uniform bound on y'' that is true for all j.

4.2 Stability

In constrast, **stability** refers to the sensitivity of solutions to initial conditions. We derived a bound on stability in Theorem 2.9.

We would like to have a corresponding notion of "numerical" stability.

Definition 4.1 (Zero stability)**:** Suppose $\{y_n\}$ and $\{z_n\}$ are approximate solutions to

$$y' = f(t, y), \qquad y(t_0) = y_0 \tag{4.1}$$

in $[a, b]$. If it holds that

$$|y_n - z_n| \leq C|y_0 - z_0| + O(h^p) \tag{4.2}$$

where C is *independent* of n, then the method is called **zero stable**.

Note that the best we can hope for is $C = e^{L(t-t_0)}$ since the numerical method will never be more stable than the actual IVP. In what follows, we will try to determine the right notions of stability for the numerical method.

As written, the stability condition is not easy to check. However, one can derive easy to verify conditions that imply zero stability. We have the following informal result.

Proposition: A "typical" numerical method is **zero stable** in the sense (4.2) if it is numerically stable when used to solve the trivial ODE

$$y' = 0.$$

Here "typical" includes any of the methods we consider in class (like Euler's method) and covers most methods for ODEs one encounters in practice. "Numerical stability" means that a perturbation at some step will not cause the solution to blow up. Numerical stability for $y' = 0$ is much easier to check than the original definition of zero stability.

4.3 General convergence theorem

With some effort, one can show that this notion of stability is exactly the minimum required for the method to converge.

Theorem 4.2 (Convergence theorem, informal (Dahlquist))**.** A "typical" numerical method converges if it is **consistent** and **zero stable**. Moreover, it is true that

$$LTE = O(h^{p+1}) \implies \text{global error} = O(h^p).$$

This assertion was proven for Euler's method directly. Observe that the theorem lets us verify two simple conditions (easy to prove) to show that a method converges (hard to prove).

Example: convergence by theorem The Backward Euler method (to be studied in detail later) is

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf(t_{j+1}, \widetilde{y}_{j+1}).$$

Consistency: The local truncation error is defined by

$$y_{j+1} = y_j + hf(t_j + h, y_{j+1}) + \tau_{j+1}.$$

Expanding around $t_j + h$ and using the ODE we get

$$y_j = y_{j+1} - y'(t_{j+1})h + O(h^2) = y_{j+1} - hf(t_j + h, y_{j+1}) + O(h^2).$$

Plugging this in to the formula yields $\tau_{j+1} = O(h^2)$ so the method is consistent.

Zero stability: This part is trivial. When the ODE is $y' = 0$,

$$\widetilde{y}_{j+1} = \widetilde{y}_j$$

which is clearly numerically stable.

The theorem then guarantees that the method is convergent, and that the order of convergence is 1 (the global error is $O(h)$).

4.4 Example of an unstable method

Obviously, any method we propose should be consistent (i.e. the truncation error is small enough). As the theorem asserts, consistency is not enough for convergence! A simple example illustrates what can go wrong when a method is consistent but not stable.

Euler's method can be derived by replacing y' in the ODE with a forward difference:

$$\frac{y(t+h) - y(t)}{h} = y' = f(t, y).$$

One might hope, then, that a more accurate method can be obtained by using a second-order forward difference

$$y'(t) = \frac{-y(t+2h) + 4y(t+h) - 3y(t)}{2h} + O(h^2).$$

Plugging this in, we obtain the method

$$-\widetilde{y}_{j+2} = -4\widetilde{y}_{j+1} + 3\widetilde{y}_j + 2hf(t_j, \widetilde{y}_j) \tag{4.3}$$

which is consistent with an $O(h^3)$ LTE. However, this method is not zero stable!

It suffices to show numerical instability for the trivial ODE $y' = 0$. The iteration reduces to

$$\widetilde{y}_{j+2} = 4\widetilde{y}_{j+1} - 3\widetilde{y}_j.$$

Plugging in $y_j = r^j$ we get a solution when

$$r^2 - 4r + 3 = 0 \implies r = 1,\ 3$$

so the general solution is[5]

$$\widetilde{y}_j = a + b \cdot 3^j.$$

If initial values are chosen so that

$$\widetilde{y}_0 = \widetilde{y}_1$$

then $y_j = y_0$ for all j with exact arithmetic. However, if there are any errors ($\widetilde{y}_0 \neq \widetilde{y}_1$) then $b \neq 0$ and $|\widetilde{y}_j|$ willl grow exponentially. Thus, the method is unstable, and is not convergent.

Obtaining a second order method therefore requires a different approach.

5 Runge-Kutta methods

In this section the most popular general-purpose formulas are introduced, which can be constructed to be of any order.

- How might we derive a higher-order method? Considering the Taylor expansion, a first idea is to include higher-order derivatives. This leads us to **Taylor's method.**
- However, Taylor's method is inconvenient as it involves derivatives of f. A better way to get higher-order is to use **Runge-Kutta methods**, which use evaluations of f at multiple points to achieve higher-order error. We'll first consider the second-order **explicit midpoint method** or **modified Euler method**, and then the classic **RK-4 method**.
- Finally, we'll look at **implicit Runge-Kutta methods**, and derive the second-order **trapezoidal method**.

5.1 Setup: one step methods

Euler's method is the simplest of the class of **one step** methods, which are methods that involve only y_j and intermediate quantities to compute the next value y_{j+1}. In contrast, **multi-step methods** use more than the previous point, e.g.

$$y_{j+1} = y_j + h(a_1 y_{j-1} + a_2 y_{j-2})$$

which will be addressed later.

Definition 5.1 (One step method)**:** A general explicit one step method has the form

$$\widetilde{y}_{j+1} = \widetilde{y}_j + h\psi(t_j, \widetilde{y}_j) \tag{5.1}$$

where ψ is some function we can evaluate at (t_j, y_j). The truncation error is defined by

$$y_{j+1} = y_j + h\psi(t_j, y_j) + \tau_{j+1}. \tag{5.2}$$

[5]See Appendix C for a review of solving these recurrences.

To improve on the accuracy of Euler's method with a one-step method, we may try to include higher order terms to get a. To start, write (5.2) as

$$\underbrace{y_{j+1}}_{\text{LHS}} = \underbrace{y_j + h\psi(t_j, y_j)}_{\text{RHS}} + \tau_{j+1}$$

For a p-th order method, we want the LHS to equal the RHS up to $O(h^{p+1})$. Now expand the LHS in a Taylor series around t_j:

$$\text{LHS: } y_{j+1} = y(t_j) + hy'(t_j) + \frac{h^2}{2}y''(t_j) + \cdots$$

A p-order formula is therefore obtained by taking

$$\psi(t_j, y_j) = y'(t_j) + \frac{h}{2}y''(t_j) + \cdots + \frac{h^{p-1}}{p!}y^{(p-1)}(t_j).$$

The key point is that the derivatives of $y(t)$ at can be expressed in terms of f and its partial derivatives - which we presumably know. Simply differentiate the ODE $y' = f(t, y(t))$ in t, being **careful with the chain rule**. If $G(t, y)$ is any function of t and y evaluated on the solution $y(t)$ then

$$\frac{d}{dt}(G(t, y(t))) = G_t + G_y y'(t) = G_t + fG_y.$$

with subscripts denoting partial derivatives and G_t etc. evaluated at $(t, y(t))$.

It follows that

$$y'(t) = f(t, y(t)),$$
$$y''(t) = f_t + f_y f,$$
$$y'''(t) = (f_t + f_y f)' = f_{tt} + f_{ty}f + \cdots \text{ (see HW)}.$$

In operator form,

$$y^{(p)} = \left(\frac{\partial}{\partial t} + f\frac{\partial}{\partial y}\right)^{p-1} f.$$

Taylor's method: The p-th order one-step formula

$$y_{j+1} = y_j + hy'(t_j) + \frac{h^2}{2}y''(t_j) + \cdots + \frac{h^{p+1}}{(p+1)!}y^{(p+1)}(t_j) + O(h^{p+1}).$$

Note that $y', y'', \ldots$ are replaced by formulas involving f and its partials by repeatedly differentiating the ODE.

This method is generally not used due to the convenience of the (more or less equivalent) Runge-Kutta methods.

5.2 A better way: Explicit Runge-Kutta methods

Taylor's method is inconvenient because it involves derivatives of f. Ideally, we want a method that needs to know $f(t, y)$ and nothing more.

The key observation is that the choice of ψ in Taylor's method is not unique. We can replace ψ with **anything else that has the same order error**. The idea of a **Runge-Kutta method** is to replace the expression with function evaluations at "intermediate" points involving computable values starting with $f(t_j, y_j)$.

5.2.1 Explicit midpoint rule (Modified Euler's method)

Let us illustrate this by deriving a second-order one-step method of the form

$$\underbrace{y_{j+1}}_{\text{LHS}} = \underbrace{y_j + w_1 h f_1 + w_2 h f_2}_{\text{RHS}} + O(h^3).$$

where

$$\begin{aligned} f_1 &= f(t_j, y_j), \\ f_2 &= f(t_j + h/2, y_j + h\beta f_1) \end{aligned}$$

and w_1, w_2, β are constants to be found.

Aside (integration): You may notice that this resembles an integration formula using two points; this is not a coincidence since

$$y' = f(t, y) \implies y_{j+1} - y_j = \int_{t_j}^{t_{j+1}} f(t, y(t))\, dt$$

so we are really estimating the integral of $f(t, y(t))$ using points at t_j and $t_{j+1/2}$. The problem is more complicated than just integrating $f(t)$ because the argument depends on the unknown $y(t)$, so that also has to be approximated.

To find the coefficients, expand everything in a Taylor series, keeping terms up to order h^2:

$$\begin{aligned} \text{LHS} &= y_j + h y_j' + \frac{h^2}{2} y_j'' + O(h^3) \\ &= y_j + hf + \frac{h^2}{2}(f_t + f f_y) + O(h^3) \end{aligned}$$

where f etc. are all evaluated at (t_j, y_j). For the f_i's, we only need to expand f_2 (using Taylor's Theorem A.1 for multivariate functions):

$$\begin{aligned} hf_2 &= hf + \frac{h^2}{2} f_t + h^2 f_y(\beta f_1) + O(h^3) \\ &= hf + \frac{h^2}{2} f_t + \beta h^2 f f_y + O(h^3). \end{aligned}$$

Plugging this into the RHS gives

$$\begin{aligned}\text{RHS} &= y_j + h(w_1+w_2)f + \frac{w_2h^2}{2}f_t + w_2\beta h^2 f f_y + O(h^3)\\ \text{LHS} &= y_j + hf \qquad\qquad + \frac{h^2}{2}(f_t + f f_y) \qquad\quad + O(h^3)\end{aligned}$$

Comparing the LHS/RHS are equal up to $O(h^3)$ if

$$w_1 + w_2 = 1, \quad w_2 = 1, \quad w_2\beta = \frac{1}{2}$$

which gives

$$w_1 = 0, \quad w_2 = 1, \quad \beta = \frac{1}{2}.$$

We have therefore obtained the formula

$$y_{j+1} = y_j + hf_2 + O(h^3),$$

$$f_1 = f(t_j, y_j), \quad f_2 = f(t_j + h/2, y_j + hf_1/2).$$

This gives the **explicit midpoint rule** or the **modified Euler's method**.

Algorithm 5.2 (Explicit midpoint rule, or modified Euler's method)**:** This is a second-order method given by

$$\widetilde{y}_{j+1} = \widetilde{y}_j + hf_2,$$

$$f_1 = f(t_j, \widetilde{y}_j), \quad f_2 = f(t_j + h/2, \widetilde{y}_j + hf_1/2).$$

Remark (integration connection): In this case one can interpret the formula as using the midpoint rule to estimate

$$y(t_{j+1}) = y(t_n) + \int_{t_j}^{t_{j+1}} f(t, y(t))\, dt \approx y(t_j) + hf(t_j + h/2, y(t_j + h/2))$$

but using Euler's method to estimate the midpoint value:

$$y(t_j + h/2) \approx y_n + \frac{h}{2}f(t_j, y_j).$$

In fact, when $f = f(t)$, the method reduces exactly to the composite midpoint rule for $\int f(t)\, dt$. However, in general, the various intermediate quantities do not have a clear interpretation, and the formulas can appear somewhat mysterious. Deriving methods from integration formulas is done in a different way (multistep methods).

5.2.2 Higher-order explicit methods

The modified Euler method belongs in the class of **Runge-Kutta** methods.

Definition 5.3 (Explicit RK method)**:** A general **explicit Runge-Kutta (RK) method** uses m "substeps" or "stages" to get from y_j to y_{j+1} and has the form

$$\begin{aligned} f_1 &= f(t_j, y_j) \\ f_2 &= f(t_j + c_2 h, y_j + h a_{21} f_1) \\ f_3 &= f(t_j + c_3 h, y_j + h a_{31} f_1 + h a_{32} f_2) \\ &\vdots \\ f_m &= f(t_j + c_m h, y_j + h a_{m1} f_1 + \cdots + h a_{m,m-1} f_{m-1}) \\ y_{j+1} &= y_j + h(w_1 f_1 + \cdots + w_m f_m), \end{aligned}$$

where $c_i = \sum_{k=1}^{i-1} a_{ik}$ (in order for the time to correspond to the y-estimates). Each f_j is an evaluation of f at a y-value obtained as y_j plus a linear combination of the previous f_i's. The "next" value y_{j+1} is a linear combination of all the f_i's.

- The best possible local truncation error is $O(h^{p+1})$ where $p \leq m$. For each p, the system is underdetermined and has a family of solutions (see HW for the $p = 2$ case).
- Unfortunately, it is not true that $p = m$. That is, to get a high order method - fifth order and above - we need more substeps per iteration than the order.
- Deriving RK methods past third order is quite tedious and a mess of algebra, since the system for the coefficients is non-linear and the Taylor series expansions become complicated.

Thankfully, just about every useful set of coefficients - at least for general purpose methods - has been calculated already, so in practice one can just look them up. They are typically arranged in the **Butcher Tableau**, defined by

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ c_m & a_{m1} & \cdots & a_{mm} \\ \hline & w_1 & \cdots & w_m \end{array}$$

Note that for an explicit method, the "matrix" A in the table only has nonzeros in the strictly lower triangular part.

Algorithm 5.4 (The classical RK-4 method)**:** One four stage method of note is the classical "RK-4" method

$$\begin{aligned} f_1 &= f(t_n, y_n) \\ f_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hf_1\right) \\ f_3 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hf_2\right) \\ f_4 &= f(t_n + h, y_n + hf_3) \\ y_{n+1} &= y_n + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4). \end{aligned}$$

This method has a good balance of efficiency and accuracy (only four function evaluations per step, and $O(h^5)$ LTE). The method would be a good first choice for solving ODEs, except that there is a more popular variant that is better for error estimation, the Runge-Kutta-Fehlberg method. (The formula is rather hairy so I will not copy it here, see e.g. http://maths.cnam.fr/IMG/pdf/RungeKuttaFehlbergProof.pdf.)

5.3 Implicit methods

The explicit RK methods are nice because we simply iterate to compute successive values, ending up with y_{j+1}. That is, all quantities in the formula can be computed explicitly.

However, we could also include y_{j+1} in the formula as well. For a one step method, the formula would take the form

$$y_{j+1} = y_j + \psi(t_j, y_j, y_{j+1}) + \tau_{j+1}.$$

We have already seen an example of this, the backward Euler's method.

Nothing is different in the theory; the truncation error is calculated in the same way and the same remarks on convergence apply. In practice, however, more work is required because y_{j+1} is defined implicitly in the formula:

$$\widetilde{y}_{j+1} = \widetilde{y}_j + \psi(t_j, \widetilde{y}_j, \widetilde{y}_{j+1}).$$

Suppose we have computed values up to $\widetilde{y}_j$. Define

$$g(z) = z - \widetilde{y}_j - \psi(t_{j+1}, \widetilde{y}_j, z).$$

Then $\widetilde{y}_{j+1}$ is a root of $g(z)$ (which is computable for any z). Thus $\widetilde{y}_{j+1}$ can be computed by applying Newton's method (ideally) or some other root-finder to $g(z)$.

Practical note: The obvious initial guess is $\widetilde{y}_j$, which is typically close to the root. If h is small, then $\widetilde{y}_j$ is almost guaranteed to be close to the root, and moreover

$$\widetilde{y}_{j+1} \to \widetilde{y}_j \text{ as } h \to 0.$$

Thus, if Newton's method fails to converge, h can be reduced to make it work. Since the initial guess is close, quadratic convergence ensures that the Newton iteration will only take a few steps to achieve very high accuracy - so each step is only a few times more work than an equally accurate explicit method.

You may wonder why we would bother with an implicit method when the explicit methods are more efficient per step; the reason is that they have other desirable properties to be explored in the next section. For some problems, implicit methods can use much larger h values than explicit ones.

5.3.1 Example: deriving the trapezoidal method

Here we derive a second-order method that uses $f(t_j, y_j)$ and $f(t_{j+1}, y_{j+1})$. The formula is

$$y_{j+1} = y_j + hw_1 f(t_j, y_j) + hw_2 f(t_{j+1}, y_{j+1}) + \tau_{j+1}.$$

First, note that for the RHS, we only need to expand y_{j+1} up to an $O(h^2)$ error. Using

$$y_{j+1} = y_j + hf + O(h^2),$$

we have that

$$f(t_{j+1}, y_{j+1}) = f(t_j + h, y_j + A)$$

where $A = hf + O(h^2)$. Since $A^2 = O(h^2)$, the result is

$$\begin{aligned}
\text{RHS} &= hw_1 f + hw_2 f(t_j + h, y_j + A) \\
&= hw_1 f + hw_2 \Big(f + hf_t + f_y A + O(A^2) \Big) \\
&= h(w_1 + w_2) f + w_2 h^2 \Big(f_t + f f_y \Big) + O(h^3).
\end{aligned}$$

Comparing to the LHS,

$$\text{LHS} = y_{j+1} = y_j + hf + \frac{h^2}{2}(f_t + f f_y) + O(h^3)$$

we find that $w_1 = w_2 = 1/2$. This gives

$$y_{j+1} = y_j + \frac{h}{2}(f_j + f_{j+1}) + O(h^3)$$

where $f_j = f(t_j, y_j)$ and $f_{j+1} = f(t_{j+1}, y_{j+1})$.

Algorithm 5.5 (Implicit Trapezoidal rule)**:** This is a second-order method given by

$$\widetilde{y}_{j+1} = \widetilde{y}_j + \frac{h}{2}(f_j + f_{j+1})$$

$$f_j = f(t_j, \widetilde{y}_j), \quad f_{j+1} = f(t_{j+1}, \widetilde{y}_{j+1}).$$

Note that when $f = f(t)$, the formula reduces to the composite trapezoidal rule.

5.4 Summary of Runge-Kutta methods

A general (possibly explicit) Runge-Kutta method is in the following form.

Definition 5.6 (General RK method)**:** A general **Runge-Kutta (RK) method** uses m "substeps" or "stages" to get from y_j to y_{j+1} and has the form

$$\begin{aligned} f_1 &= f\left(t_j + c_1 h, y_j + h\sum_{k=1}^{m} a_{1k} f_k\right) \\ &\vdots \\ f_m &= f\left(t_j + c_m h, y_j + h\sum_{k=1}^{m} a_{mk} f_k\right) \\ y_{j+1} &= y_j + h(w_1 f_1 + \cdots + w_m f_m), \end{aligned}$$

where $c_i = \sum_{k=1}^{m} a_{ik}$ (in order for the time to correspond to the y-estimates). Each f_j is an evaluation of f at a y-value obtained as y_j plus a linear combination of the previous f_i's. The "next" value y_{j+1} is a linear combination of all the f_i's.

As before, they can be arranged in the tableau

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ c_m & a_{m1} & \cdots & a_{mm} \\ \hline & w_1 & \cdots & w_m \end{array}$$

The formulas we discussed, and several others, are given on the next page.

Problem 5.7: Based on the tableau, write out the formulas for the explicit trapezoidal and the implicit midpoint rules. Show that they are second-order accurate.

Name	Order	Tableau
Forward Euler	1	$\begin{array}{c\|c} 0 & 0 \\ \hline & 1 \end{array}$
Explicit Midpoint/ Modified Euler	2	$\begin{array}{c\|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$
Explicit Trapezoidal	2	$\begin{array}{c\|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$
RK-4	4	$\begin{array}{c\|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$
Backward Euler	1	$\begin{array}{c\|c} 1 & 1 \\ \hline & 1 \end{array}$
Implicit Midpoint	2	$\begin{array}{c\|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$
Implicit Trapezoidal	2	$\begin{array}{c\|cc} 0 & 0 & 0 \\ 1 & 0 & 1 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$

Figure 6: Summary of RK methods

6 Absolute stability and stiff equations

6.1 An example of a stiff problem

Convergence (and zero stability and consistency) are only the bare minimum required for a method to work. This does not, however, guarantee that the method works *well*, just that it works when h is small enough. To motivate the discussion, consider the ODE

$$y' = -20(y - \sin t) + \cos t, \quad y(0) = 1 \tag{6.1}$$

for $t \in [0, 3]$. The solution is

$$y(t) = e^{-20t} + \sin t.$$

At early times (see Figure 6.5), there is a rapidly decaying transient; then it settles down and looks like $\sin t$ (the "long term" solution). To obtain a reasonable solution for small t (up to about $1/20$), we need to pick h small enough that the approximation captures the variations in $y(t)$. To get a qualitatively correct solution, h must be small enough to resolve the solution features.

However, once t is large enough ($\gg \frac{1}{20}$) the initial transient e^{-20t} disappears, and we are left with the much smoother part $y(t) = \sin t$. We would like to use larger time steps to compute the solution.

What happens if Euler's method is used? The situation is shown in Figure 6.5. Some observations:

- If $h > 1/10$, then the numerical solution oscillates and diverges (the amplitude grows exponentially). If $h = 1/10$ exactly, the oscillations stay bounded.
- If $h < 1/10$, the iteration suddenly settles down and behaves correctly.

There is a threshold of $h^* = 1/10$; the step size must be less than h^* to have a qualitatively correct solution. Note that this is a separate issue from convergence, which only guarantees that the error is $O(h)$ for sufficiently small h.

The requirement that $h < 1/10$ is a **stability constraint**: the step size must be below a certain threshold to avoid numerical instability. (Even *in the absence* of truncation error, a larger step size would cause the solution to diverge.)

The threshold depends on the problem. An ODE that has such a constraint is called **stiff**. Our first "practical" definition is as follows: [6]

[6]The definition of stiffness and example are borrowed from Ascher & Petzold, *Computer methods for ordinary differential equations*.

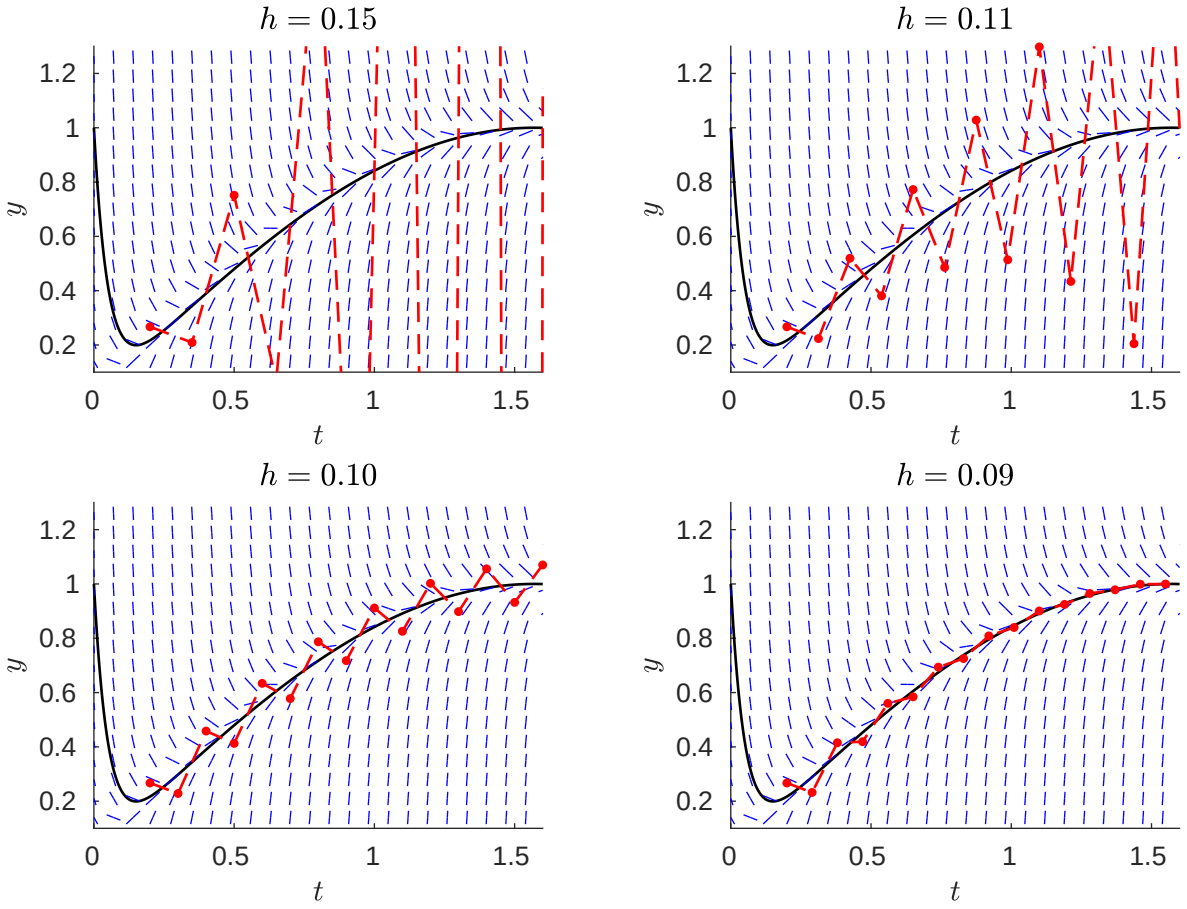


Figure 7: Euler's method applied to the stiff IVP (6.1) and the slope field (blue lines).

Definition 6.1 (Stiffness, definition I)**:** An IVP in some interval $[a, b]$ is called **stiff** if Euler's method requires a much smaller time step h to be stable than is needed to represent the solution accurately.

We will see shortly that "Euler's method" can be replaced by "typical explicit method".

Note that this is a practical definition in that the meaning of 'accurate" and "much smaller" depends on what is needed. The ODE from the example is not stiff in $[0, 0.01]$ by this definition since $h < 1/10$ for accuracy, but is stiff in $[0, 3]$. On the interval $[0, 3]$, to represent the solution accurately means representing the $\sin t$ part accurately, and we know that it is possible to obtain the curve $\sin t$ with much larger step size (e.g., if the $-20(y-\sin t)$ part were removed from the ODE, and we just had $y' = \cos t$, we can use fairly large step size and still obtain $y \approx \sin t$).

At the present, the "stability" constraint remains mysterious; we will develop some theory shortly to explain it.

A geometric interpretation

Nearby solutions with some other initial condition rapidly converge to $y(t)$ (if the solution is perturbed, it will quickly return). The numerical method should, but doesn't, always behave the same way.

The figure (Figure 6.5) is revealing here, recalling that Euler's method follows the vector field. Observe that if h is not small enough, Euler's method will overshoot due to the large slope. This process will continue, causing aggressive oscillations that will diverge. If h is at a certain value, then the oscillations will stay bounded.

Below this value, the overshoot is not severe and the method provides a reasonable approximation.

Observation: An IVP with solution $y(t)$ (in an interval) is stiff if $y(t)$ is well-behaved but nearby trajectories to $y(t)$ vary rapidly compared to $y(t)$ itself.

This is *not* the same as instability of the IVP itself; the solution is in fact quite stable here. All solution curves, in fact, will approach $\sin t$ as $t \to \infty$.

6.2 The test equation: Analysis for Euler's method

A simple analysis explains the phenomenon. Define the "test equation"

$$y' = \lambda y, \qquad y(0) = y_0 \tag{6.2}$$

where λ is a (complex) number. The solution is just $y(t) = y_0 e^{\lambda t}$ so

- If $\mathrm{Re}(\lambda) < 0$ then $y(t) \to 0$ as $t \to \infty$
- If $\mathrm{Re}(\lambda) > 0$ then $y(t)$ grows exponentially.

A good numerical method should have the property that

$$\mathrm{Re}(\lambda) < 0 \implies \widetilde{y}_j \to 0 \text{ as } j \to \infty.$$

That is, if the true solution decays, the approximation should also decay (exponentially). In particular, it should not grow exponentially when the true solution does not.

What does the numerical method do? For Euler's method[7], we have

$$y_{n+1} = y_n + hf(t_n, y_n) = (1 + h\lambda)y_n.$$

The test equation is simple enough that the iterates have an exact formula:

$$y_n = (1 + h\lambda)^n y_0.$$

Now define a region R of the complex plane as the set of $h\lambda$ for which the iteration has $|y_n| \to 0$ as $n \to \infty$ (i.e. the set of $h\lambda$ such that Euler's method applied to (6.2) gives a sequence that converges to zero). Then

$$R = \{z \in \mathbb{C} : |1 + z| < 1\}.$$

This is a circle of radius 1 centered at $z = -1$. In particular, $|y_n| \to 0$ if

$$h < \frac{2}{|\lambda|}. \tag{6.3}$$

Moreover, if $h > 2/|\lambda|$ then $|y_n|$ will increase exponentially. This means that in order to have y_n decay exponentially when the exact solution does, the condition (6.3) must hold. Otherwise, y_n will grow exponentially even though it should go to zero.

Note that the $\lambda > 0$ is not as much of a concern here; if $\lambda > 0$ then $1 + h\lambda > 1$ for any positive h so y_n will grow exponentially as desired.

6.3 Analysis for a general ODE

For the first-order ODE

$$y' = f(t, y)$$

the behavior on the test equation tells us how the numerical method will behave locally. Consider a point (t_0, y_0), the solution $y(t)$ and a nearby solution

$$w = y + \epsilon u.$$

where ϵu is a small perturbation (so w starts at $w_0 = y_0 + \epsilon u_0$). Plugging into $w' = f(t, w)$,

$$y' + \epsilon u' = f(t, y + \epsilon u) = f(t, y) + \epsilon \frac{\partial f}{\partial y} u + O(\epsilon^2)$$

[7]For convenience, the tilde's have been dropped.

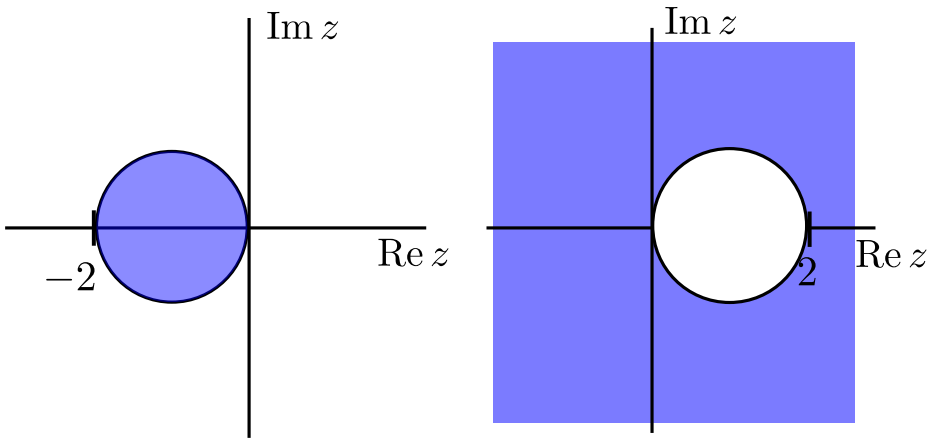


Figure 8: Sketches of the stability regions (shaded area) for Euler's method (left) and Backward Euler (right) in the complex plane.

so the perturbation u evolves according to

$$u' = \frac{\partial f}{\partial y} u + O(\epsilon^2) \approx \left(\frac{\partial f}{\partial y}(t_0, y_0) \right) u$$

near (t_0, y_0). That is, for nearby trajectories $w(t)$ to the solution $y(t)$, the difference will grow/decay exponentially at a rate $\frac{\partial f}{\partial y}$.

Key point: Locally, for a perturbed solution $w(t)$ to the ODE $y' = f(t, y)$, the difference will evolve like the test equation $u' = \lambda u$ with $\lambda = \partial f / \partial y$. This determines the local stability of the numerical method.

Indeed, this matches the observed behavior from the motivating example where $\frac{\partial f}{\partial y} = -20$. Solutions that are nearby to $y(t)$ want to be pushed onto $y(t)$ exponentially fast (with rate 20) as shown by the vector field in Figure 6.5. The numerical method must therefore behave well on the test equation $y' = -20y$ in order to be stable.

The test equation analysis then yields a new, more precise identification of stiffness:

Stiffness (definition II:) A first order ODE is stiff if $|\frac{\partial f}{\partial y}|$ is large in the sense that $1/|\frac{\partial f}{\partial y}|$ is much less than the typical width of a change in $y(t)$.

Geometrically, this typically means that nearby trajectories to $y(t)$ converge rapidly to $y(t)$ compared to the variation in $y(t)$ itself.

6.4 Stability regions

Now consider a method that produces a sequence y_n of approximations. The stability constraint turns out to depend only on the location of the product $h\lambda$.

Definition 6.2 (stability region)**:** The **region of absolute stability** R is the set of complex numbers $z \in \mathbb{C}$ with $z = h\lambda$ such that for the test equation

$$y' = \lambda y$$

the approximation $\{y_n\}$ with a step size h converges to zero as $n \to \infty$.

The **interval of absolute stability** is the real portion of R (that is, the same definition but λ is real).

Note: Typically, the part of R that matters is the region where $\mathrm{Re}(\lambda) < 0$.

We found that the region of absolute stability for Euler's method is

$$R = \{z : |1 + z| < 1\}$$

and its interval of absolute stability is $(-2, 0)$.

For the backward Euler method

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

we get

$$y_{n+1} = \frac{1}{(1 - h\lambda)} y_n$$

so $|y_n| \to 0$ when

$$|1 - h\lambda| > 1.$$

The region of absolute stability is then the set

$$R = \{z \in \mathbb{C} : |1 - z| > 1\}$$

which is the complement of a circle centered at $z = 1$ of radius 1 (see Figure 6.3). In particular, R contains the entire half plane $\{z < 0\}$, which means that if $\mathrm{Re}\lambda < 0$ then $|y_n| \to 0$ for any positive value of h.

6.5 A-stability and L-stability; Stability in practice

So, what does this analysis say about what methods to use for stiff equations? Observe that if R contains all points with $\mathrm{Re}(z) < 0$ then there is no stability constraint, because $h\lambda \in R$ for all λ's with negative real part.

If the interval contains $(-\infty, 0)$, then $h\lambda \in \mathbb{R}$ for all real λ's, which is often still good enough. Such a method can be applied to a stiff equation without any constraint on h for stability: it will always be true that the approximation stays bounded when it should.

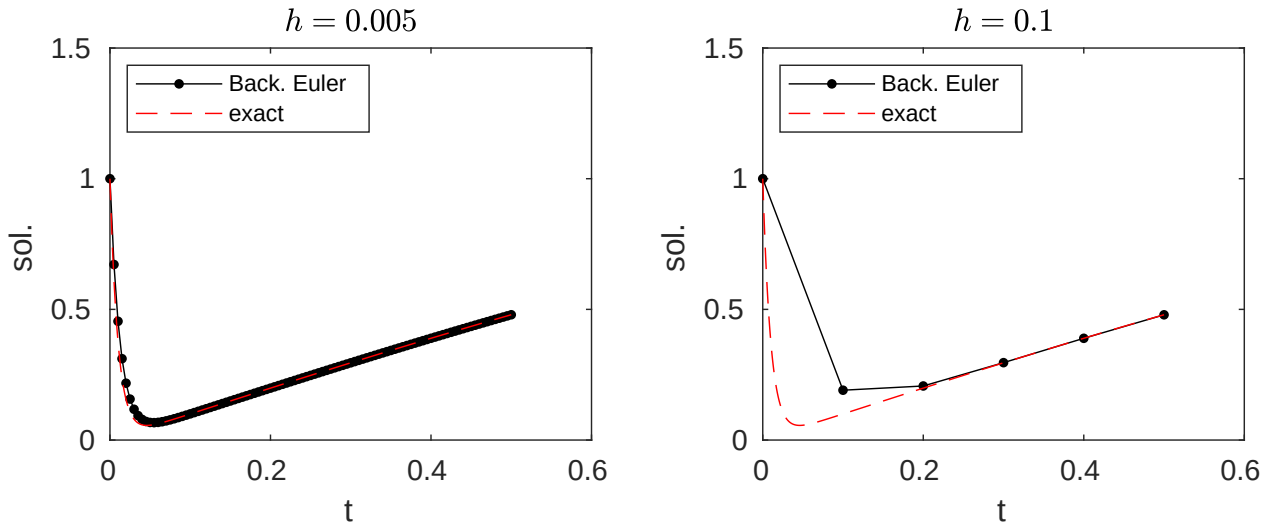


Figure 9: Backward Euler to the stiff IVP (6.4). The approximation stays bounded for all h, although for the initial transient, h must be taken smaller ($h \sim 1/200$) to be accurate.

Definition 6.3: A method for which R contains all of $\{\mathrm{Re}(z) < 0\}$ is called **A-stable**. For such a method, there is no stability condition required when integrating a stiff equation.

For example, the backward Euler method is A-stable. Indeed, if it is used on the example problem, the approximation will always be reasonable, even when $h > 1/10$. Below, Euler's method and Backward Euler are used to solve

$$y' = -100(y - \sin t) + \cos t, \quad y(0) = 1. \tag{6.4}$$

Backwards Euler does fine in the stiff interval, but Euler's method would require $h < 1/50$: the stiffness imposes a severe constraint here and there is a clear winner away from the initial transient.

The trapezoidal method is also A-stable (see homework). However, there are a few points worth noting here:

- Not all implicit methods are A-stable, or even have intervals of absolute stability containing $(-\infty, 0)$. Only certain classes of implicit methods are good for stiff problems.
- All explicit methods have finite intervals of absolute stability $(-a, 0)$ and the value of a is typically not that large. The regions R for RK methods up to order 4 are shown in Figure 6.5. The stability constraint for RK-4 is more or less the same as for Euler's method.

There is much more to the story than just absolute stability. Many other notions of stability exist, describing other constraints and finer properties of numerical solutions. For

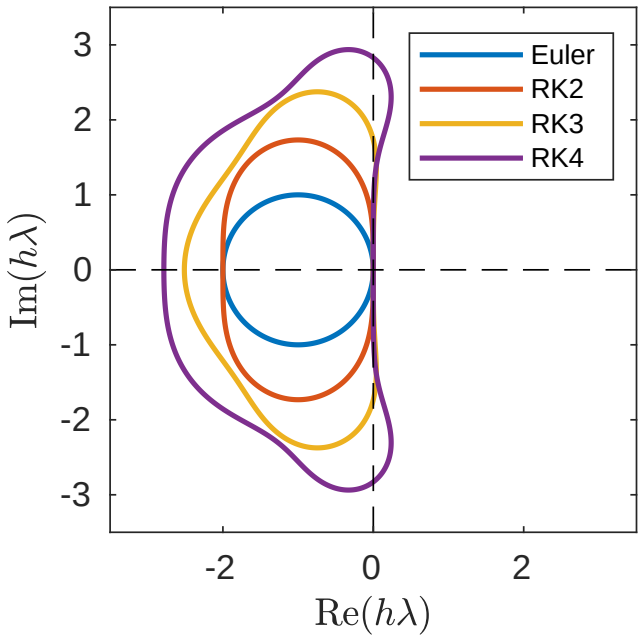


Figure 10: Stability regions (the area inside the curve) for RK methods up to order 4. Note that all RK methods of the same order have the same stability region.

example, we may want it to be true that

$$\text{the rate at which } |y_n| \to 0 \text{ increases as } \mathrm{Re}(\lambda) \to -\infty.$$

For instance, if $\lambda = -1000$ we would want the approximation to decay to zero much faster than if $\lambda = -10$. Backward Euler has this property (called L-stability or "stiff decay") since

$$|y_{n+1}| = \frac{1}{|1 - h\lambda|}|y_n|$$

so as $\mathrm{Re}\lambda \to -\infty$, the coefficient goes to zero in magnitude. The property is nice because it means that *fast-decaying transients are damped out quickly* by the method.

6.6 Absolute stability for systems

For systems of ODEs, the analysis for absolute stability must be modified slightly. The definition of the region of absolute stability is the same as before; we apply the method to the scalar test equation.

In principle, one might want to consider the linear system

$$\mathbf{y}' = A\mathbf{y}$$

instead. However, if A is diagonalizable, then $A = V\Lambda V^{-1}$ where Λ is the diagonal matrix of eigenvalues, and setting $\mathbf{x} = V^{-1}y$ we get the decoupled equations

$$x_i' = \lambda_i x_i$$

for each component of $\mathbf{x}$. Thus it suffices to know how the method behaves on the scalar test equation to know how it behaves on the "system" version.

For a general system

$$\mathbf{y}' = \mathbf{F}(t, \mathbf{y}),$$

and a perturbation $\mathbf{w} = \mathbf{y} + \epsilon\mathbf{u}$ near (t_0, y_0), the difference $\mathbf{u}$ evolves according to

$$\mathbf{u}' \approx (D\mathbf{F}(t_0, \mathbf{y}_0))\mathbf{u}$$

where $D\mathbf{F}$ is the Jacobian of $\mathbf{F}$ (with respect to $\mathbf{y}$). Letting

$$\lambda^* = \text{eigenvalue with most negative real part of } D\mathbf{F}.$$

it follows that the stability constraint should be

$$h\lambda^* \in R.$$

That is, the numerical stability constraint is determined by the component that decays fastest (the "most stiff" component). As a trivial example,

$$x' = -5x$$
$$y' = -100y$$

using Euler's method has the stability constraint $-100h \in (-2, 0) \implies h < 1/50$ because the y-component is stiff. Note that in general, "component" means in the eigenvector basis of $D\mathbf{F}$, so it cannot typically be seen by just looking at the equations in the system individually.

The "stiffness ratio" is the ratio of the largest/smallest real parts (in absolute value) of the Jacobian; if this ratio is large, the system is typically stiff since it has some components that change slowly and some that change very fast (e.g. the ratio is 20 in the example above).

7 Adaptive time stepping

Notice that one step methods use only the values at the previous time step n to update to time $n + 1$. For this reason, once we are at $n + 1$, we are free to pick a new value of h. It is desirable to have an algorithm that can adjust the time step h to ensure that the local truncation error remains below a certain tolerance:

$$|\tau_i| < \epsilon \text{ for all } i.$$

To do so, we need an estimate for τ_i at each step, and a way to select a step size ϵ that will ensure that the estimated error is acceptably small.

On global error: We will keep the goal modest and not discuss strategies for bounding the **global error**, which is more difficult.

One might, for instance, want the global error to remain below a tolerance ϵ:

$$\max_{t\in[a,b]} |\widetilde{y}(t) - y(t)| < \epsilon.$$

Bounding the τ's individually does not lead to a natural bound on the global error, since there is propagation and so on to worry about as well. But local control is good enough in practice, so long as one is careful to be cautious (setting local tolerances to be smaller than you think is necessary).

7.1 Using two methods of different order

A good strategy is to used two methods of different order. Informally, we use the more accurate one as an estimate for the exact solution, to estimate the error for the less accurate method.

Assume that the value of the n-th step is exact (y_n), a step size h and we have two one-step methods:

i) Method A: Order p, producing $\widetilde{y}_{n+1}$ from y_n, truncation error $\tau_{n+1}(h)$ with step size h

ii) Method B: Order $p+1$, producing $\hat{y}_{n+1}$ from y_n

We seek a step size h_{new} such that

$$\tau_{n+1}(h_{\text{new}}) < \epsilon.$$

To derive the estimate, suppose that the value at time n is exact. We have that

$$\widetilde{y}_{n+1} = y_n + h\psi(t_n, y_n),$$

$$\hat{y}_{n+1} = y_n + h\hat{\psi}(t_n, y_n).$$

By the definition of the truncation error,

$$y(t_{n+1}) = \widetilde{y}_{n+1} + \tau_{n+1}, \tag{7.1}$$

$$y(t_{n+1}) = \hat{y}_{n+1} + O(h^{p+2}) \tag{7.2}$$

Now further approximate (7.1) by assuming that the error comes from a series with some leading order term:

$$\tau_{n+1}(h) = Ch^{p+1} + O(h^{p+2}).$$

Then

$$y(t_{n+1}) = \widetilde{y}_{n+1} + Ch^{p+1} + O(h^{p+2}).$$

Now subtract the more accurate method (7.2) from this to eliminate $y(t_{n+1})$, leaving

$$|\widetilde{y}_{n+1} - \hat{y}_{n+1}| = |C|h^{p+1} + O(h^{p+2}).$$

This gives an error estimate:

$$|\tau(h)| \approx |C|h^{p+1} \approx |\widetilde{y}_{n+1} - \hat{y}_{n+1}|. \tag{7.3}$$

At this point, we have enough to get the updated time step h_{new}. It must satisfy

$$|\tau(h_{\text{new}})| \approx |C|(h_{\text{new}})^{p+1} < \epsilon.$$

Taking the ratio of this and the estimate (7.3) (or solving for C) yields

$$(h_{\text{new}}/h)^{p+1} < \frac{\epsilon}{|\widetilde{y}_{n+1} - \hat{y}_{n+1}|}, \tag{7.4}$$

which is a formula for the next step h_{new} in terms of known quantities: The two approximations from the two methods, the last step h and the tolerance ϵ.

In practice, because this is just an estimate, one puts an extra coefficient in to be safe, typically something like

$$h_{\text{new}} = 0.8h \left(\frac{\epsilon}{|\widetilde{y}_{n+1} - \hat{y}_{n+1}|} \right)^{1/(p+1)}.$$

This formula decreases the step size if the error is too large (to stay accurate) and increases the step size if the error is too small (to stay efficient). Sometimes, other controls are added (like not decreasing or increasing h by too much per time step).

> **!** The estimate is based on some strong assumptions that may not be true. While it typically works, there is no rigorous guarantee that the error estimate is good. Moreover, as noted earlier, this strategy bounds the **local** error, which does not always translate into a reasonable **global** error bound.

7.2 Embedded RK methods

Each step with the error estimate appears to cost about twice as much one fixed step (since two methods are used). However, with a judicious choice, we can create some overlap in the computations, saving work.

One of the main approaches is to use an **embedded pair** of Runge-Kutta methods, which is a pair of RK formulas where most of the f_i's are the same for both.

For example, suppose we wish to create a pair for estimating the error in Euler's method. We need a method of order 2 here. Recall that the **modified Euler method** is

$$\begin{aligned} f_1 &= f(t_n, y_n) \\ f_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{h}{2}f_1\right) \\ y_{n+1} &= y_n + hf_2 + O(h^3). \end{aligned}$$

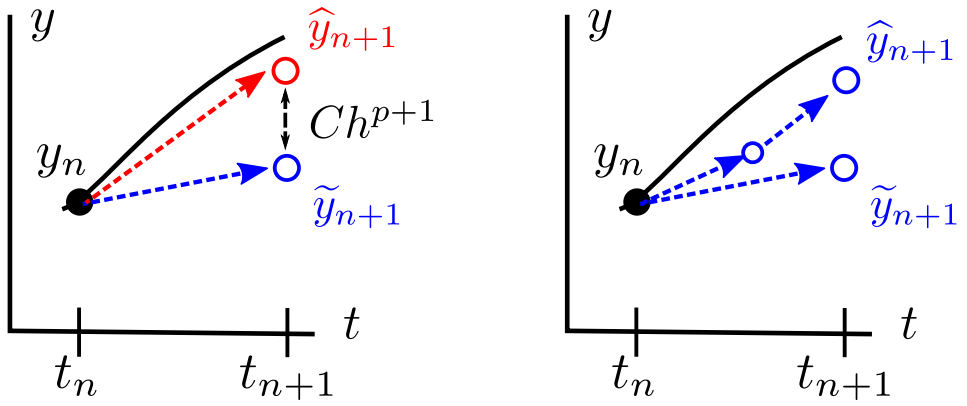


Figure 11: Left: error estimate using two methods of different order (red/blue). Right: sketch of step doubling using one method with one step of size h and two half-steps of size $h/2$.

The function evaluation needed for Euler's method is already there, so once modified Euler is applied, computing

$$\widetilde{y}_{n+1} = y_n + hf_1 + O(h^2)$$

requires essentially no extra work; the value of f_1 is used for both.

Following (7.4), the timestep would then be chosen to be something like

$$h_{\text{new}} \approx h(\epsilon/|\widetilde{y}_{n+1} - \hat{y}_{n+1}|)^{1/2}.$$

where $\widetilde{y}_{n+1}$ and $\hat{y}_{n+1}$ are the result of Euler (order 1) and modified Euler (order 2).

Embedded methods of higher order can be constructed by the right choice of coefficients. We saw that fourth-order RK methods have the best balance of accuracy and efficiency. There are a handful of good fourth/fifth order pairs.

One popular embedded pair is the **Runge-Kutta-Fehlberg** method, which uses a fourth-order and fifth-order formula that share most of the f_i's. A formula of this form with step size selected by (7.4) is the strategy employed, for instance, by MATLAB's ode45.[8]

7.3 Step doubling

Now suppose instead that we have only a single method that can take a step size h (for any h) and wish to choose a new step size h_{new} such that the local error is at most ϵ. By using (Richardson) extrapolation, an error estimate and a new step size can be obtained.

The idea is to use **step doubling**. Suppose, for the sake of example, that our method is a one step method of the form

$$y_{n+1} = y_n + h\psi(t_n, y_n) + O(h^{p+1}).$$

[8]Minor note for completeness: `ode45` uses a different set of coefficients than RKF, the 'Dormand-Prince' pair, which offers slightly better accuracy.

Assume y_n is exact; then the next step is

$$\widetilde{y}_{n+1} = y_n + h\psi(t_n, y_n).$$

Now take two steps of size $h/2$ (see Figure 11, right) to get a new approximation:

$$\hat{y}_{n+1/2} = y_n + \frac{h}{2}\psi(t_n, y_n),$$
$$\hat{y}_{n+1} = \hat{y}_{n+1/2} + \frac{h}{2}\psi(t_n, \hat{y}_{n+1/2}).$$

Assume that each application of a step creates a LTE

$$\tau \approx Ch^{p+1}$$

and that C is a single constant.[9] Then, if y_n is exact, we have

$$y_{n+1} \approx y(t_{n+1}) + Ch^{p+1}$$
$$\hat{y}_{n+1} \approx y(t_{n+1}) + 2C(h/2)^{p+1}$$

i.e. the 'doubled' method accumulates two truncation errors from a step size $h/2$. Subtracting the two approximations gives

$$|y_{n+1} - \hat{y}_{n+1}| \approx (1 - 2^{-p})|Ch^{p+1}|.$$

Thus the error estimate is

$$|Ch^{p+1}| \approx \frac{|y_{n+1} - \hat{y}_{n+1}|}{1 - 2^{-p}}$$

from which we can choose a new time step h_{new} such that $C(h_{\text{new}})^{p+1} < \epsilon$.

8 Multistep methods

A **linear multistep method** has the form

$$\sum_{j=0}^{m} a_j y_{n-j} = h \sum_{j=0}^{m} b_j f_{n-j} \tag{8.1}$$

where $f_{n-j} = f(t_{n-j}, y_{n-j})$. The methods are so named because they involve values from the m previous steps, and are linear in f (unlike Runge-Kutta methods). For example, Euler's method

$$y_n = y_{n-1} + hf_{n-1}$$

and the Backward Euler method

$$y_n = y_{n-1} + hf_n$$

are both (trivially) linear multistep methods that only involve the previous step; we call these **one step methods**. Note that the method (8.1) is implicit if $b_0 \neq 0$ and explicit otherwise. We are interested here in methods that use more than just the previous step.

For simplicity, we will assume throughout that the discrete times are $t_0, t_1, \ldots$ with fixed timestep h.

[9] This can be made more precise by assuming that the LTE for a step of size h starting at t is $\tau(h; t) \approx C(t)h^{p+1}$ where $C(t)$ is a smooth function of t.

8.1 Adams methods

The starting point is the integrated form of the ODE:

$$y(t_n) = y(t_{n-1}) + \int_{t_{n-1}}^{t_n} y'(t)\,dt.$$

Note that $y'(t) = f(t, y(t))$, but it is convenient to leave it as y'.

Change variables to $t = t_n + sh$. Then

$$y(t_n) = y(t_{n-1}) + h\int_{-1}^{0} y'(t_n + sh)\,ds. \tag{8.2}$$

To get an explicit method using the previous m values, we estimate the integral using the previous times $t_{n-1}, t_{n-2}, \ldots, t_{n-m}$, or equivalently $s = -1, -2, \ldots, -m$:

$$\int_{-1}^{0} g(s)\,ds = \sum_{j=1}^{m} b_j g(-j) + Cg^{(m)}(\xi).$$

We already know how to derive such formulas. For example, for $m = 2$,

$$\int_{-1}^{0} g(s)\,ds = \frac{3}{2}g(-1) - \frac{1}{2}g(-2) + \frac{5}{12}g^{(2)}(\xi).$$

Now plugging in $y'(t_n + sh)$ into the formula, we get

$$y(t_n) = y(t_{n-1}) + \frac{3h}{2}y'(t_{n-1}) - \frac{h}{2}y'(t_{n-2}) + \frac{5}{12}y^{(3)}(t_n + h\xi)h^3$$

for some $\xi \in (-m, 0)$. The numerical method is then

$$y_n = y_{n-1} + \frac{3h}{2}f_{n-1} - \frac{h}{2}f_{n-2}.$$

In the notation of (8.1), we have $m = 2$ and $a_0 = 1, b_1 = 3/2, b_2 = -1/2$. The other a-values are zero: $a_1 = a_2 = 0$.

A method that uses the m previous steps to estimate the integral is called an **Adams-Bashforth** method. For a general m, it takes the form

$$y_n = y_{n-1} + h\left(\sum_{j=1}^{m} b_j f_{n-j}\right).$$

The local truncation error is $O(h^{m+1})$, so the method has order m. There is a trick for deriving the coefficients; see example below.

We can also derive a method from (8.2) by including t_n (or $s = 0$). With $m = 1$ this would be the trapezoidal rule:

$$\int_{-1}^{0} g(s)\,ds = \frac{1}{2}g(-1) + \frac{1}{2}g(0) + Cg^{(2)}(\xi).$$

Note that this method uses $m+1$ points (the m previous points plus t_n). The result is then the **trapezoidal method**

$$y_n = y_{n-1} + \frac{h}{2}(f_n + f_{n-1})$$

which has order 2. In general, the method will have the form

$$y_n = y_{n-1} + h\left(\sum_{j=0}^{m} b_j f_{n-j}\right).$$

and is called an **Adams-Moulton** method. The local truncation error is $O(h^{m+2})$, so the method is order $m+1$. Note that the method is implicit, since f_n appears on the RHS.

Examples: The Adams-Bashforth method for $m = 3$ requires a formula

$$\int_{-1}^{0} g(s)\,ds \approx b_1 g(-1) + b_2 g(-2) + b_3 g(-3).$$

We will omit the details of the error term here. The easiest way to derive the coefficients is to use undetermined coefficients on the Newton basis:

$$1, \quad s+1, \quad (s+1)(s+2), \ldots$$

The reason is that the resulting linear system will be triangular and easy to solve (and can in fact there is a nice general formula). The method should have degree of accuracy 2, so we require that it be exact for $1, s+1$ and $(s+1)(s+2)$. Plugging these in, we get

$$\begin{aligned} 1 &= b_1 + b_2 + b_3 \\ \frac{1}{2} &= -2b_2 - b_3 \\ \frac{5}{6} &= 2b_3 \end{aligned}$$

so $b_1 = 23/12, b_2 = -16/12$ and $b_3 = 5/12$. This gives the method

$$y_{n+1} = y_n + \frac{h}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3})$$

which has order of accuracy 3. The Adams-Moulton method with the same order of accuracy has $m = 2$ and requires the formula

$$\int_{-1}^{0} g(s)\,ds \approx b_0 g(0) + b_1 g(-1) + b_2 g(-2).$$

Requiring that the formula is exact for $1, s, s(s+1)$ (same trick as before) yields

$$\begin{aligned} 1 &= b_0 + b_1 + b_2 \\ -\frac{1}{2} &= -b_1 - 2b_2 \\ -\frac{1}{6} &= 2b_2. \end{aligned}$$

After solving, the result is the method

$$y_n = y_{n-1} + \frac{h}{12}\left(8f_n + 5f_{n-1} - f_{n-2}\right)$$

with order of accuracy 3.

8.2 Properties of the Adams methods

The main benefit of these methods is efficiency: one function evaluation is required per step.

It is also true that

- The order p Adams-Moulton method has a (much) smaller error than the order p Adams-Bashforth method (the constant for the LTE is much smaller).
- When $m \geq 2$, the Adams-Moulton methods have a bounded stability region of a moderate size. (When $m = 0$ or $m = 1$ the method is A-stable.)
- Adams-Bashforth methods have a bounded stability region that is very small.

For this reason, neither type of method (even the implicit one!) is useful for stiff problems. The Adams-Moulton method is superior in terms of accuracy and (absolute) stability, but it is implicit, so it requires much more work per step.

In practice, the two are combined to form an explicit method that has some of the stability of the implicit method, but is easy to compute. The implicit term $f_n = f(t_n, y_n)$ is estimated using the result $\tilde{y}_n$ from an explicit method. This strategy is called a **predictor-corrector method**. For example, we can combine the two-step explicit formula with the one-step implicit formula:

$$\tilde{y}_n = y_{n-1} + \frac{h}{2}(3f_{n-1} - f_{n-2}) \tag{8.3}$$

$$\tilde{f}_n = f(t_n, \tilde{y}_n) \tag{8.4}$$

$$y_n = y_{n-1} + \frac{h}{2}(\tilde{f}_n + f_{n-1}). \tag{8.5}$$

Now the formula is not implicit (and as a bonus, the error can be estimated using y_n and $\tilde{y}_n$ by standard techniques). It turns out that this trick gives a method that has a reasonably good stability region (not as good as the implicit one!) and is essentially strictly better than the pure explicit method (8.3) alone.

Practical note (starting values): To start, we need m previous values, which means $y_1, \ldots, y_{m-2}$ must be computed by some other method (not a multistep formula). A high-order RK method is the simplest choice, because it only requires one previous point.

8.3 Other multistep methods

The Adams methods only use previous f_n's and not y_n's. There are other classes of multistep methods of the form (8.1) that are derived in different ways. One such class are the **Backward differentiation formulas** (BDFs). These are derived by approximating y' using a backward difference:

$$hy'(t_n) \approx c_0 y(t_n) + c_1 y(t_{n-1}) + \cdots + c_m y(t_{n-m}).$$

For example, the first order BDF is Backward Euler; the second order BDF is

$$\frac{3y_n - 4y_{n-1} + y_{n-2}}{2h} = f(t_n, y_n)$$

which rearranges to

$$y_n = \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2} + \frac{2h}{3}f_n.$$

This method is sometimes called **Gears' method**. BDFs are valuable because for orders up to 6, they do well on stiff problems (see next section) compared to Adams-Moulton methods, and moreover have the stiff decay property.

8.4 Analysis of multistep methods

8.4.1 Consistency

The general multi-step method (8.1) can be shown to be consistent by Taylor expanding $y(t_{n-1}), \ldots$ and $y'(t_{n-1})$ (in place of f) around t_n, then canceling out terms to get the local truncation error. We have

$$y(t_{n-j}) = y(t_n) - jhy'(t_n) + \frac{1}{2}(jh)^2 y''(t_n) + \cdots$$

and

$$y'(t_{n-j}) = y'(t_n) - jhy''(t_n) + \frac{1}{2}(jh)^2 y'''(t_n) + \cdots .$$

The truncation error is

$$\begin{aligned}
\tau_n &= \sum_{j=0}^{m} a_j y(t_{n-j}) - h\sum_{j=0}^{m} b_j y'(t_{n-j}) \\
&= \sum_{j=0}^{m} a_j y(t_n) - h\sum_{j=0}^{m} j a_j y'(t_n) - h\sum_{j=0}^{m} b_j y'(t_n) + O(h^2) \\
&= y(t_n)\sum_{j=0}^{m} a_j + hy'(t_n)\left[\sum_{j=0}^{m}(b_j - ja_j)\right] + O(h^2).
\end{aligned}$$

Thus the method (8.1) is consistent if and only if

$$\sum_{j=0}^{m} a_j = 0, \qquad \sum_{j=0}^{m} b_j = \sum_{j=0}^{m} j a_j.$$

Further conditions can be derived by taking more terms. Of course, for the Adams formulas and BDFs, we derived the method and proved consistency by other (more elegant) means, but with the above we can construct more general methods.

8.4.2 Zero stability

One necessary condition for convergence is that when applied to the trivial ODE $y' = 0$, solutions remain bounded. If $f = 0$ then the method reduces to

$$\sum_{j=0}^{m} a_j y_{n-j} = 0 \tag{8.6}$$

which is a linear recurrence for y_n. This turns out to be sufficient! The following theorem is true.

Theorem 8.1 (Dahlquist equivalence theorem)**.** A linear multistep method of the form (8.1) is convergent if and only if it is consistent and all solutions to (8.6) remain bounded as $n \to \infty$.

The equation (8.6) can be solved directly (see Section C for a review), leading to the following condition.

Theorem 8.2. All solutions of the linear recurrence (8.6) are bounded if and only if all the roots of the characteristic polynomial

$$p(r) = \sum_{j=0}^{m} a_j r^{m-j} \tag{8.7}$$

have magnitude ≤ 1.

The Dahlquist Theorem lets us verify a multistep method is convergent simply by proving consistency and finding the roots of the characteristic polynomial (8.7).

8.4.3 Strongly stable methods

From the consistency result, we have that

$$\text{if the method is consistent, } r = 1 \text{ is a root of } p(r).$$

For zero stability, this is the "good" root, since it means that if $y' = 0$ then $y_n = \text{const.}$ is a solution. The other roots are "bad", in that they are just spurious terms that appear due to errors. If one of these roots is also 1, the bad term will stay bounded but not decay, which is not ideal.

A method is called **strongly stable** if there is one root $r = 1$ and all other roots are strictly less than 1 in magnitude. The method is **weakly stable** if there are multiple roots with magnitude 1.

Weakly stable methods tend not to damp out errors as much, which makes them undesirable unless they are needed for some other compelling reason.

8.5 Absolute stability

The definition of absolute stability is the same. When more than one previous y-value is involved, the recurrence has more than two terms, so the analysis is more involved. When the general method (8.1) is applied to $y' = \lambda y$ we get

$$\sum_{j=0}^{m} a_j y_{n-j} = h \sum_{j=0}^{m} \lambda b_j y_{n-j}.$$

Let $z = h\lambda$ as before. The above is a linear recurrence with characteristic polynomial

$$p(r; z) = \sum_{j=0}^{m} (a_j - z b_j) r^{m-j}.$$

To have solutions to the recurrence go to zero as $n \to \infty$, we need all the roots of $p(r, z)$ to have magnitude less than one. Thus, with R the region of absolute stability,

$$z \in R \text{ if and only if the roots of } p(r; z) = 0 \text{ all have } |r| < 1.$$

This is now a condition that can, with some effort, be computed and we can plot the stability region.

Example: Stability region for Adams methods The two-step Adams-Bashforth method is

$$y_n = y_{n-1} + h\left(\frac{3}{2} f_{n-1} - \frac{1}{2} f_{n-2}\right).$$

Applied to $y' = \lambda y$, this becomes

$$y_n = y_{n-1} + z\left(\frac{3}{2} y_{n-1} - \frac{1}{2} y_{n-2}\right).$$

with $z = h\lambda$. The characteristic polynomial for this recurrence is

$$p(r; z) = r^2 - r - z\left(\frac{3}{2} r + \frac{1}{2}\right).$$

Rearranging, we get (equivalently) that the roots satisfy

$$2r^2 - (2 + 3z)r + z = 0.$$

Solving, we get

$$r = \frac{2 + 3z \pm \sqrt{(2 + 3z)^2 - 8z}}{4}.$$

Thus z is in the region of absolute stability if and only if the two roots above are less than 1 in magnitude. This is not a nice looking condition. However, we can determine the largest b such that $(-b, 0)$ is in the interval of absolute stability.

Observe that on the boundary of R, one of the roots must have $|r| = 1$. If $z < 0$ is real then both roots are real (by the calculations above). Thus we need only find the values of z for which $r = \pm 1$ is a root, which occurs when

$$z = 0 \text{ or } z = -1.$$

Thus the interval of absolute stability is $(-1, 0)$, which is half the size of Euler's method!

Some facts absolute stability regions for multistep methods:

- A linear multistep method with order ≥ 2 cannot be A-stable. (This is the **second Dahlquist barrier**.)
- The interval of absolute stability for BDFs up to order 6 contains $(-\infty, 0)$.
- The stability region for Adams-Moulton methods is bounded for $m > 2$.
- The stability region for Adams-Bashforth methods shrinks as the order increases.

It follows that higher order Adams-Moulton methods are not to be used on stiff problems even though they are implicit, and that Adams-Bashforth methods should not be used on anything remotely stiff.[10]

A Nonlinear systems

Solving implicit systems of ODEs requires solving non-linear systems, which is a problem in itself. Here we briefly consider numerical solution of the non-linear system $\mathbf{F}(\mathbf{x}) = 0$ where

$$\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n, \qquad \mathbf{F} = (F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_n(\mathbf{x})).$$

This describes a general system of n equations for n variables. Let DF be the Jacobian matrix with entries

$$(D\mathbf{F})_{ij} = \frac{\partial F_i}{\partial x_j}$$

We willl consider finding a zero $\mathbf{x}^*$ with the property that

$$\mathbf{F}(\mathbf{x}^*) = 0 \text{ and } D\mathbf{F}(\mathbf{x}^*) \text{ is invertible,}$$

analogous to the $f'(x^*) \neq 0$ assumption in 1D. This guarantees that Newton's method (see below) works and that the zero is isolated.[11]

[10] More to the point, predictor-corrector methods using Adams-Bashforth and Adams-Moluton formulas work much better, so there is not much reason to use the Adams-Bashforth formula alone.

[11] The result is a consequence of the inverse function theorem, which ensures that $\mathbf{F}$ is invertible in a neighborhood of $\mathbf{x}^*$.

Simple example: The system of two equations

$$0 = e^{x_1}(x_2 - 1)$$
$$0 = x_1^2 - x_2^2$$

can be written in the form $\mathbf{F}(\mathbf{x}) = 0$ where

$$\mathbf{x} = (x_1, x_2), \quad \mathbf{F}(\mathbf{x}) = (e^{x_1}(x_2 - 1), x_1^2 - x_2^2).$$

It has a zero at $\mathbf{x}^* = (1, 1)$. The Jacobian is

$$DF = \begin{bmatrix} e^{x_1}(x_2 - 1) & e^{x_1} \\ 2x_1 & -2x_2 \end{bmatrix}.$$

Finally, we check that

$$DF(\mathbf{x}^*) = \begin{bmatrix} 0 & 1 \\ 2 & -2 \end{bmatrix}$$

is invertible.

A.1 Taylor's theorem

For the informal treatment here, we need only a simplified version of Taylor's theorem.

Theorem A.1 (Taylor's theorem, simple variant)**.** Suppose $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m$ is C^2 in a neighborhood of $\mathbf{x}_0$ and that $DF(x_0)$ is invertible. Then

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}_0) + D\mathbf{F}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + R(\mathbf{x})$$

where the remainder satisfies

$$|R(\mathbf{x})| \leq \frac{M}{2}\|\mathbf{x} - \mathbf{x}_0\|^2$$

for a constant M depending on the size of the partial derivatives of $\mathbf{F}$.

Much more precise statements can be made about the error, but this version is enough for our purposes.

A.2 Newton's method

Newton's method extends to $\mathbb{R}^n$ directly. Let $x^{(k)}$ (to be defined) be the sequence of iterates. As before, we use Taylor's theorem to get a linear approximation of F near $x^{(k)}$:

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}^{(k)}) + D\mathbf{F}(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) + O(\|\mathbf{x} - \mathbf{x}_k\|^2).$$

Using this 'locally linear' approximation, we pick $\mathbf{x}_{k+1}$ so that

$$0 \approx \mathbf{F}(\mathbf{x}_{k+1}) \approx \mathbf{F}(\mathbf{x}^{(k)}) + D\mathbf{F}(\mathbf{x}^{(k)})(\mathbf{x}^{k+1} - \mathbf{x}^{(k)})$$

leading to

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}_k^{-1}\mathbf{F}(x_k), \qquad \mathbf{J}_k := D\mathbf{F}(\mathbf{x}_k).$$

In practice, we solve the linear system

$$\mathbf{J}_k\mathbf{v} = -\mathbf{F}(\mathbf{x}_k)$$

for the correction $\mathbf{v}$ and then update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}$.

The convergence results transfer to the n-dimensional case. If $D\mathbf{F}(\mathbf{x}^*)$ is invertible then Newton's method converges quadratically when $\|\mathbf{x}_0 - \mathbf{x}^*\|$ is small enough (see the Newton-Kantorovich theorem).

A good guess is even more valuable in $\mathbb{R}^n$ since there are more dimensions to work with - which means more possibilities for failure. Without a good guess, Newton's method will likely not work. Moreover, searching for a root is much more difficult (no bisection).

A.3 Application to ODEs

Nonlinear systems arise in solving systems of ODEs implicitly. As an example, consider the pendulum equation from earlier,

$$\theta'' + \sin\theta = f(t)$$

which was written as the system

$$\mathbf{x}' = \mathbf{F}(t, \mathbf{x}), \qquad \mathbf{F}(t, \mathbf{x}) = \begin{bmatrix} x_2 \\ -\sin x_1 + f(t) \end{bmatrix}$$

and $\mathbf{x} = (\theta, \theta')$. Suppose we apply Backwards Euler to this system:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + h\mathbf{F}(t_{j+1}, \mathbf{x}_{j+1}).$$

To obtain $\mathbf{x}_{j+1}$, define the function

$$\mathbf{G}(\mathbf{z}) = \mathbf{z} - \mathbf{x}_j - h\mathbf{F}(t_{j+1}, \mathbf{z}).$$

The Jacobian of $\mathbf{G}$ is

$$D\mathbf{G} = I - hD_{\mathbf{x}}\mathbf{F}(t_{j+1}, \mathbf{z}) = \begin{bmatrix} 1 & h \\ -h\cos z_1 & 1 \end{bmatrix}$$

where $D_{\mathbf{x}}\mathbf{F}$ denotes the Jacobian of $\mathbf{F}$ as a function of $\mathbf{x}$ (not t):

$$D_{\mathbf{x}}\mathbf{F}(t, \mathbf{x}) = \begin{bmatrix} 0 & 1 \\ -\cos x_1 & 0 \end{bmatrix}.$$

At step j, we find a root of $\mathbf{G}(\mathbf{v})$ using Newton's method:

$$(D\mathbf{G}(\mathbf{z}_k))\,(\mathbf{z}_{k+1} - \mathbf{z}_k) = -\mathbf{G}(\mathbf{z}_k) \tag{A.1}$$

and then use this as $\mathbf{x}_{j+1}$. Newton's method is particularly nice here because we have access to a good initial guess ($\mathbf{z}_0 = \mathbf{x}_j$).

Practical note: A general routine would require the user to input the ODE function $\mathbf{F}$ and its Jacobian $D_{\mathbf{x}}\mathbf{F}$, both as functions of $(t, \mathbf{x})$. If the dimension is large, it may be much more efficient to write a solver specific to that problem that knows how to solve the Newton's method system (A.1) efficiently (e.g. Appendix B).

B Boundary value problems

We give an example of using Newton's method to solve boundary value problems.

The system of N equations

$$\begin{aligned} 2y_1 - y_2 &= \cos(y_1) \\ -y_{i-1} + 2y_i - y_{i+1} &= \sin(y_i), \qquad i = 2, \cdots N-1 \\ -y_{N-1} + 2y_N &= \cos(y_N) \end{aligned}$$

are a discretization of the **boundary value problem**

$$-y'' = \sin(y), \quad y(0) = y(1) = 0$$

for equally spaced data $\{y_j\}$ at points $x_0, \cdots, x_N$ in $[0, 1]$. To write as a system, set

$$\mathbf{y} = (y_1, \cdots, y_n),$$

and define the function $\mathbf{F}$ with i-th component

$$\mathbf{F}_i(\mathbf{y}) = -y_{i-1} + 2y_i - y_{i+1} - \cos(y_i), \qquad i = 2, \cdots N-1.$$

and the exceptional cases

$$\begin{aligned} \mathbf{F}_1(\mathbf{y}) &= 2y_1 - y_2 - \sin(y_1), \\ \mathbf{F}_N(\mathbf{y}) &= -y_{N-1} + 2y_N - \sin(y_N). \end{aligned}$$

Note that the Jacobian of $\mathbf{F}$ is a tridiagonal matrix.

To apply Newton's method, we first compute the Jacobian $\mathbf{J}$. Let $r_i = \sin(y_i)$. Then

$$\mathbf{J}(\mathbf{y}) = \begin{bmatrix} 2+\sin(y_1) & -1 & 0 & \cdots & 0 \\ -1 & 2+\sin(y_2) & -1 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & -1 & 2++\sin(y_{N-1}) & -1 \\ 0 & \cdots & 0 & -1 & 2++\sin(y_N) \end{bmatrix}.$$

To apply Newton's method, pick some starting vector $\mathbf{y}_0$. Then, at step k, solve

$$(\mathbf{J}(\mathbf{y}_k))\mathbf{v}_k = -\mathbf{F}(\mathbf{y}_k)$$

which is a tri-diagonal system (so it can be solved efficiently) and then update

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \mathbf{v}_k.$$

C Difference equations: review

Consider the linear difference equation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots c_m a_{n-m}.$$

We solve by looking for solutions of the form

$$a_n = r^n.$$

This is a solution if r is a root of the characteristic polynomial

$$p(r) := r^n - c_1 r^{n-1} - c_2 r^{n-2} - \cdots - c_{m-1} r - c_m.$$

There are m (complex) roots $r_1, \cdots r_m$. Since the equation is linear, any linear combination is a solution. The general solution is therefore

$$a_n = \sum_{j=1}^{m} b_j r_j^n$$

for constants $b_1, \cdots b_m$. Note that the solution is bounded for any initial conditions if and only if

$$|r_j| \leq 1 \text{ for all } j$$

and that $a_n \to 0$ for any initial conditions if and only if

$$|r_j| < 1 \text{ for all } j.$$