TMA4125 Matematikk 4N

Numerics for Initial Value Problems

Ronny Bergmann

Department of Mathematical Sciences, NTNU.

March 30, 2023

Motivation

Similar to Integration: Only a very limited number of ODEs can be solved analytically.

Example. A simple pendulum

$$\begin{cases} \theta''(t) & = -\frac{g}{L}\sin(\theta(t)) \\ \theta(0) & = \theta_0 \\ \theta'(0) & = 0 \end{cases}$$

has no analytical solution!

Approximate for small θ: $\sin\theta \approx \theta$
$\Rightarrow$ We can solve the approximate ODE

$$\theta'' = -\frac{g}{L}\theta \quad \Leftrightarrow \quad \theta(t) = \theta_0 \cos\Big(\sqrt{\tfrac{g}{L}}t\Big), \quad \text{period } T = \frac{2\pi}{\sqrt{gL}} = 2\pi\sqrt{\frac{L}{g}}.$$

First Order ODEs

A scalar ODE of first order is an equation of the form

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0$$

where $y'(t) = \frac{\mathrm{d}y}{\mathrm{d}t}$ and $y(t_0) = y_0$ is required for uniqueness.

These ODEs are called initial value problems (IVP).
We are interested in the solution $y(t)$ for $t > t_0$
If f depends linearly on y it is called linear.

Examples.

- $y'(t) = 3y(t)$, $\quad f(t, y) = 3y$, linear
- $y'(t) = -2ty(t)$, $\quad f(t, y) = -2ty$, linear
- $y'(t) = t^3 - 2t^2 y(t)$, $\quad f(t, y) = t^3 - 2t^2 y$, linear
- $y'(t) = 1 - (y(t))^2$, $\quad f(t, y) = 1 - y^2$, nonlinear

NTNU

Systems of ODEs

A System of ODEs is given by

$$
\begin{aligned}
y_1' &= f_1(t, y_1, y_2, \ldots, y_m), \qquad & y_1(t_0) &= y_{1,0} \\
y_2' &= f_2(t, y_1, y_2, \ldots, y_m), \qquad & y_2(t_0) &= y_{2,0} \\
&\vdots \\
y_m' &= f_m(t, y_1, y_2, \ldots, y_m), \qquad & y_m(t_0) &= y_{m,0}
\end{aligned}
$$

or more compactly

$$
\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \qquad \mathbf{y}(t_0) = \mathbf{y}_0
$$

with

$$
\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} f_1(t, y_1, y_2, \ldots, y_m) \\ f_2(t, y_1, y_2, \ldots, y_m) \\ \vdots \\ f_m(t, y_1, y_2, \ldots, y_m) \end{pmatrix}, \quad \mathbf{y}_0 = \begin{pmatrix} y_{1,0} \\ y_{2,0} \\ \vdots \\ y_{m,0} \end{pmatrix}
$$

Example: Preditor-Prey or Lotka-Volterra-Model

We describe 2 species

- $y_1(t)$ is the population of some prey (maybe rabbits, or small fish)
- $y_2(t)$ is the population of some predator (maybe foxes or sharks)

Then we have a (simplified) model of their interaction as

$$y_1'(t) = \alpha y_1(t) - \beta y_1(t) y_2(t)$$
$$y_2'(t) = \delta y_1(t) y_2(t) - \gamma y_2(t)$$

where $\alpha, \beta, \delta, \gamma$ are parameters describing the interaction. Or in short

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \quad \text{with} \quad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} \alpha y_1 - \beta y_1 y_2 \\ \delta y_1 y_2 - \gamma y_2 \end{pmatrix}$$

Notes.

We need some initial populations $\mathbf{y}_0$ and some initial time t_0

But. The right hand side does not depend on t

Autonomous ODEs

A (system of) ODE(s) is called autonomous if the function f only depends on y and not (directy) on t

- $y'(t) = 3y(t)$, $\quad f(t,y) = 3y$, autonomous, linear
- $y'(t) = -2ty(t)$, $\quad f(t,y) = -2ty$, non-autonomous, linear
- $y'(t) = 1 - (y(t))^2$, $\quad f(t,y) = 1 - y^2$, autonomous, nonlinear

A trick. A system of ODEs, can be made autonomous introducing a $m+1$st variable

$$y'_{m+1} = 1, \qquad y_{m+1}(t_0) = t_0$$

and replacing all occurrences of t in $\mathbf{f}$ by y_{m+1}.

Higher Order ODEs

An initial value problem (IVP) for an ODE of order m is given by

$$
\begin{aligned}
u^{(m)} &= f(t, u, u', \ldots, u^{(m-1)}), \\
u(t_0) &= u_0, \\
u'(t_0) &= u'_0, \\
&\vdots \\
u^{(m-1)}(t_0) &= u_0^{(m-1)}
\end{aligned}
$$

where $u^{(1)} = u'$ and $u^{(m+1)} = \frac{\mathrm{d}u^{(m)}}{\mathrm{d}t}$ for $m > 0$ denotes the $(m+1)$st derivative.

Rewrite.
We can rewrite a higher order ODE into a system of first-order ODEs.

Higher order ODEs to System of ODEs

Introduce. New variables:
$y_1(t) = u(t), \quad y_2(t) = u'(t), \quad y_3(t) = u^{(2)}(t), \ldots, y_m(t) = u^{(m-1)}(t).$

We observe. Taking the derivative $y_i' = (u^{(i+1)})' = u^{(i+2)} = y_{i+1}$, $i = 1, \ldots, m-1.$
$\Rightarrow$ We obtain the following first order **System** of ODEs

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
&\vdots \\
y_{m-1}' &= y_m \\
y_m' &= f(t, y_1, y_2, y_3, \ldots, y_m))
\end{aligned}
$$

Example. Van der Pol's Equation

Van der Pol's equation is a second order differential equation given by

$$u'' = \mu(1-u^2)u' - u, \qquad u(0) = u_0, \quad u'(0) = u_0',$$

where $\mu > 0$.
Common choices. $u_0 = 2$, $u_0' = 0$.

System of First Order ODEs

$$\begin{aligned} y_1' &= y_2 & y_1(0) &= u_0 \\ y_2' &= \mu(1-y_1^2)y_2 - y_1 & y_2(0) &= u_0' \end{aligned}$$

Numerical Methods

Terms and Notation

Focus. Scalar ODEs, but can be directly applied to systems of ODEs, too.

Approach. We will take timesteps $t_0, t_1, t_2, \ldots$
introduce/compute the approximations $y_n \approx y(t_n)$.
$\Rightarrow$ for "errors" we consider $|y_n - y(t_n)|$!

Methods. We will only consider one-step methods.
Given

- an ODE (i.e. a right hand side f)
- initial values (t_0, y_0)
- a step size h

$\Rightarrow$ We compute a first step $y_1 \approx y(t_1)$ with $t_1 = t_0 + h$

$\Rightarrow$ based on (t_1, y_1): compute $y_2 \approx y(t_2)$ with $t_2 = t_1 + h$

- ... and so on until a final time t_{end} is reached.

One-Step Methods and Beyond

One-Step Methods are only “allowed” to use the information from the previous step,
i. e. the approximation y_{k+1} does not depend on $y_{k-1}, y_{k-2}, \ldots$

Main alternative. multi-step methods are allowed to take previous values into account as well.

Summary. We numerically compute an approximation to y at discrete time points $t_n, n = 0, 1, \ldots,$

Be careful! We often compare

- $y(t_n)$ the (analytical) solution at t_n
- y_n the nth step of the numerical methods (which only approximates $y(t_n)$)

Euler's method

A first algorithm to solve $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ is given as[1]

Euler's method

1. Given / Input: A function $\mathbf{f}(t, \mathbf{y})$ and initial value $(t_0, \mathbf{y}_0)$
2. Choose a step size h
3. For $n = 0, 1, 2 \ldots$
   - $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$
   - $t_{n+1} = t_n + h$

Let's look at two examples in Python.

[1] either derived via Taylor expansion or forward differences

Trapezoidal method

Idea. Let's integrate the ODE $\mathbf{y}'(t) = \mathbf{f}(t, y)$ from t_n to t_{n+1}.
And use the trapezoidal rule to approximate the integral

The update for the Trapezoidal rule reads

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}\big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})\big).$$

This is a so-called implicit method, since $\mathbf{y}_{n+1}$ appears on boths sides and is hence only implicitly given.
$\Rightarrow$ We would have to solve for $\mathbf{y}_{n+1}$ in every iteration.

NTNU

Heun's method

Remedy. Instead of solving the nonlinear equation for $\$\mathbf{y}_{n+1}$, first approximate/replace $\mathbf{y}_{n+1}$ on the right by applying a step from Euler's method. We obtain

Heun's method.

$$
\begin{aligned}
\mathbf{u}_{n+1} &= \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n) \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}\big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{u}_{n+1})\big).
\end{aligned}
$$

This is more commonly written (emphasizing reusing terms) as

$$
\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{y}_n), \\
\mathbf{k}_2 &= \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{k}_1), \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}\big(\mathbf{k}_1 + \mathbf{k}_2\big).
\end{aligned}
$$

Notation Interlude: Increment function Φ

We saw that the one-step methods can be written as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h_n \mathbf{\Phi}(t_n, \mathbf{y}_n, \mathbf{y}_{n+1}, h_n),$$

where for us $h_n = h$ does not change during the iterations (but it indeed could), and
The function $\mathbf{\Phi}$ is called increment function.

A method is called

- explicit if $\mathbf{\Phi}$ does not depend on $\mathbf{y}_{n+1}$
- implicit if $\mathbf{\Phi}$ does depend on $\mathbf{y}_{n+1}$

Examples. Are the following implicit or explicit?

- Euler: $\mathbf{\Phi}(t_n, \mathbf{y}_n, \mathbf{y}_{n+1}, h) = \mathbf{f}(t_n, \mathbf{y}_n)$
- Trapezoidal: $\mathbf{\Phi}(t_n, \mathbf{y}_n, \mathbf{y}_{n+1}, h) = \frac{1}{2}\big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_n + h, \mathbf{y}_{n+1})\big)$
- Heun: $\mathbf{\Phi}(t_n, \mathbf{y}_n, \mathbf{y}_{n+1}, h) = \frac{1}{2}\big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n))\big)$

Runge-Kutta

Runge-Kutta-Method

Definition. An s-stage Runge-Kutta method is given by

$$\mathbf{k}_i = \mathbf{f}(t_n + c_i h,\ \mathbf{y}_n + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j), \qquad i = 1, 2, \ldots, s$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\sum_{i=1}^{s} b_i \mathbf{k}_i$$

Defined by its coefficients, which are given in a Butcher tableau

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

with $c_i = \sum_{j=1}^{s} a_{ij}, \qquad i = 1, \ldots, s.$

$\Rightarrow$ THe method is explicit if $a_{ij} = 0$ for $j \geq i$ (diagonal and above).

Examples of Runge-Kutta

Euler.

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

Heun.

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

Trapezoidal.

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

RK4.

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

Theoretical Results

Motivation.

We want to be able to tell

- When does a solution exist?
- When is a solution unique?
- How large is the error one step introduces?
- How large can the global (overall) error get?
- How does the error behave depending on the step size h?

In other words: While we do not have the exact solution $y(t)$,
we still want to be able to say how close we are to that,
so what the error is in our computations,
and how much is needed to improve/reduce it.

Lipschitz condition

Definition. A function $\mathbf{f}\colon \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ satisfies a Lipschitz condition with respect to $\mathbf{y}$ on a domain $(a,b) \times D$ where $D \subset \mathbb{R}^m$ if there exists a constant L such that

$$\|\mathbf{f}(t,\mathbf{y}) - \mathbf{f}(t,\mathbf{z})\| \leq L\|\mathbf{y} - \mathbf{z}\|, \qquad \text{for all} \quad t \in (a,b) \text{ and } \mathbf{y}, \mathbf{z} \in D$$

holds.
The constant L is called the Lipschitz constant.

Remark. $\mathbf{f}$ is Lipschitz if

- if $\dfrac{\partial f_i}{\partial y_j}, i,j = 1,\ldots,m$ are continuous and bounded on $(a,b) \times D$
- D is open and convex

Existence of solutions

Theorem (Existence and uniqueness of a solution)

Consider the initial value problem

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y}_0$$

with given initial values $t_0 \in (a, b)$ *and* $\mathbf{y}_0 \in D$*.*

If

- $\mathbf{f}(t, \mathbf{y})$ *is continuous in* $(a, b) \times D$
- $\mathbf{f}(t, \mathbf{y})$ *satisfies the Lipschitz condition with respect to* $\mathbf{y}$ *in* $(a, b) \times D$*,*

then the ODE has one and only one solution in $(a, b) \times D$*.*

In the following we assume that our ODE under consideration fulfils this.

Error Analysis

When we aolve an ODE e. g. with Euler's method on $[t_0, t_{\text{end}}]$ we can ask ourselves

- How will the error at the end t_{end} (or any point in between) depend on the number of steps?
- phrased differently: Choose N and set $h = \frac{t_{\text{end}} - t_0}{N}$.
  Then we get $t_N = t_{\text{end}}$.
  What can we say about the error $e_N = y(t_{\text{end}} - y_N)$?

In the following we will just consider the scalar equation $y' = f(t, y)$.
The results also hold for systems of equations.

Local and Global Error

We will consider two types of errors:

- Local Truncation Error (LTE) d_{n+1}
  denotes the error made in one single step starting from the exact/true point $(t_n, y(t_n))$
- Global error e_n
  denotes the difference between the exact ($y(t_n)$) and numerical (y_n) solution after n steps:
  $e_n = y(t_n) - y_n$

Goals.

- find an expression for d_n
- look at the relation between local errors and the global error
- find an upper bound for the global error

Local Truncation Error for Euler's method

Investigating the error in one step for an ODE $y' = f(t,y)$.
From the Taylor expansion we get

$$y(t_n + h) = y(t_n) + hy'(t_n) + \frac{1}{2}y''(\xi), \qquad t_n < \xi < t_n + h$$

Eulers method starting from $(t_n, y(t_n T))$ yields

$$y(t_n + h) \approx y(t_n) + hf(t_n, y(t_n)) = y(t_n) + hy'(t_n)$$

$\Rightarrow$ The local truncation error for Euler's method: their difference

$$d_{n+1} = y(t_n + h) - (y(t_n) + hy'(t_n)) = \frac{1}{2}h^2y''(\xi), \qquad \xi \in (t_m, t_n + h)$$

We can observe two things

- with a bound C for $y''(\xi)$ yields that $d_{n+1} = \mathcal{O}(h^2)$
- see the error of Euler's method in:
  $y(t_n + h) = y(t_n) + h(f(t_n, y_n)) + d_{n+1}$

Global Error for Euler's Scheme

Summary.

We have the exact step (inculding d_{n+1}) and the numerical scheme

$$y(t_n + h) = y(t_n) + hf(t_n, y(t_n)) + d_{n+1}$$
$$y_{n+1} = y_n + hf(t_n, y_n)$$

Goal. Using

- $e_n = y(t_n) - y_n$
- an upper bound for $f_y = \frac{\partial f}{\partial y}$ as $|f_y(t, y)| \leq L$ (a Lipschitz constant)
- an upper bound for $|y''(t)| \leq 2D$

find

1. an upper bound for $|e_{n+1}|$
2. by iteration an upper bound for $|e_N|$

$\Rightarrow$ We want to say something about decreasing h or increasing N and how e_N behaves then.

Global Error for Euler's Scheme – Result

We obtain

$$|y(t_{\text{end}}) - y_N| = |e_N| \le Dh \le Ch,$$

where $C = \dfrac{\mathrm{e}^{L(t_{\text{end}}-t_0)} - 1}{L} D$ depends on

- the length of our interval $t_{\text{end}} - t_0$
- certain properties of y (the D) and f (the L)
- **!** but not on N or h!

We especially get

$$\lim_{N\to\infty} |e_N| = 0$$

Remark.

- We got locally $d_{n+1} = \mathcal{O}(h^2)$
- We got globally $e_N = \mathcal{O}(h)$

Roughly speaking because it takes N (in order of $\frac{1}{N}$) steps (with local errors) to "reach" t_{end}.

Local Truncation Error & Consistency

For a Numerical method to solve the ODE $y' = f(t, y)$ we consider the Increment function Φ (again), that is, the function that describes our numerical method as

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

then the local truncation error reads as

$$d_{n+1} = y(t_n + h) - \big(y(t_n) + h\Phi(t_n, y(t_n), h)\big).$$

Definition (Consistency)

A numerical method is called consistent if

$$\lim_{h\to 0} \frac{d_{n+1}}{h} = 0, \qquad \text{for all } n = 0, 1, \ldots, N, N = \left\lceil \frac{b-a}{h} \right\rceil$$

It is called consistent of order p if $d_{n+1} = \mathcal{O}(h^{p+1})$.

Remark on Consistency

For systems of equations

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\boldsymbol{\Phi}(t_n, \mathbf{y}_n, h)$$

by considering the absolute norm of the LTE, i. e. if

$$\|\mathbf{y}(t+h) - \big(\mathbf{y}(t) + h\boldsymbol{\Phi}(t, \mathbf{y}(t), h)\big)\| \leq Dh^{p+1}$$

then numerical method for the system of ODEs is consistent of order p.

Convergence

Theorem (Convergence of one-step methods)

Assume that there exist positive constants M and D such that the increment function $\mathbf{\Phi}$ satisfies

$$\|\mathbf{\Phi}(t, \mathbf{y}, h) - \mathbf{\Phi}(t, \mathbf{z}, h)\| \leq M\|\mathbf{y} - \mathbf{z}\|$$

and the method is consistent of oder p, that is the local truncation error satisfies

$$\|\mathbf{y}(t+h) - \big(\mathbf{y}(t) + h\mathbf{\Phi}(t, \mathbf{y}(t), h)\big)\| \leq Dh^{p+1}$$

for all t, $\mathbf{y}$ and $\mathbf{z}$ in a neighbourhood of the solution.
In that case, the global error satisfies

$$\|\mathbf{e}_N\| = \|\mathbf{y}(t_{\text{end}}) - \mathbf{y}_N\| \leq Ch^p, \qquad \text{with } C = \frac{\mathrm{e}^{M(t_{\text{end}} - t_0)} - 1}{M} D.$$

The method is then called (convergent) of order p.

Remarks on Convergence

We saw. Consistency (of order p) and Lipschitz condition
$\Rightarrow$ Convergence (of oder p).

Example. For Euler's method: $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$
$\Rightarrow \mathbf{\Phi}(t, \mathbf{y}, h) = \mathbf{f}(t, \mathbf{y})$.
$\Rightarrow L = M$ is the same constant

For Runge-Kutta Methods. The corresponding increment function $\mathbf{\Phi}$ is maybe complicated a constant M as required in the Theorem always exists.

Convergence Order for Runge-Kutta

For Runge-Kutta methods, one can prove:

The method is of order p with $p \leq 4$ if all of the conditions in the table on the left up to and including p are fulfilled.
(all sums run from 1 to s.)

Remark.
Similar conditions for higher order exist, but get a bit more complicated.

Example. Let's check Heun's method.

p	conditions
1	$\sum_i b_i = 1$
2	$\sum_i b_i c_i = 1/2$
3	$\sum_i b_i c_i^2 = 1/3$
	$\sum_{i,j} b_i a_{ij} c_j = 1/6$
4	$\sum_i b_i c_i^3 = 1/4$
	$\sum_{i,j} b_i c_i a_{ij} c_j = 1/8$
	$\sum_{i,j} b_i a_{ij} c_j^2 = 1/12$
	$\sum_{i,j,k} b_i a_{ij} a_{jk} c_k = 1/24$

Summary (including systems of Equations)

Notation / Definitions.

- the ODE: $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$
- the exact solution through $(t^*, \mathbf{y}^*)$: $\mathbf{y}(t, t^*, \mathbf{y}^*)$
- the exact solution of the ODE with initial values $t_0, \mathbf{y}_0$:
$$\mathbf{y}(t) = \mathbf{y}(t\,;\, t_0, \mathbf{y}_0)$$
- one step of the (explicit) method (increment function $\mathbf{\Phi}$):
$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{\Phi}(t_n, \mathbf{y_n}, h)$$

Error Concepts.

Let $\mathbf{\Phi}$ represent a method of order p and $\hat{\mathbf{\Phi}}$ a method of order $p+1$.

- local truncation error (LTE)
(comparing exact solution to one step “from there”):
$$\mathbf{d}_{n+1} = \mathbf{y}(t_n + h; t_n, \mathbf{y}(t_n)) - \big(\mathbf{y}(t_n) + h\mathbf{\Phi}(t_n, \mathbf{y}(t_n), h)\big)$$
- local error: $\mathbf{l}_{n+1} = \mathbf{y}(t_n + h; t_n, \mathbf{y}_n) - \big(\mathbf{y}_n + h\mathbf{\Phi}(t_n, \mathbf{y}_n, h)\big)$
- the global error: $\mathbf{e}_n = \mathbf{y}(t_n) - \mathbf{y}_n$

Error Estimates & Adaptivity

Motivation

In our numerical tests we saw that the step size h has to be chosen

- if it is too large $\Rightarrow$ too inexact results in $\mathbf{y}_n$
- if it is too small $\Rightarrow$ too much computational time or memory usage

Approaches.

1. control the global error $\mathbf{e}_n = \mathbf{y}(t_n) - \mathbf{y}_n$
   this is quite difficult and beyond the scope of this lecture
2. control the local error $\mathbf{l}_{n+1} \coloneqq \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \mathbf{y}_{n+1}$
   where
   - we "start from" $(t_n, \mathbf{y}_n)$: We consider the exact solution $\mathbf{y}(t; t_n, \mathbf{y}_n)$ but running through $(t_n, \mathbf{y}_n)$.
   - compared to $\mathbf{d}_{n+1}$ we have hence a "different starting point" $\mathbf{y_n}$ and not $\mathbf{y}(t_n)$!

Approximating the local error

Recap. We saw that for the convergence order we needed power serieses in h.

Now since we do not have the actual solution $\mathbf{y}(t)$, let's take two methods, where one is more exact. For both we start from our previous point $(t_n, \mathbf{y}_n)$:

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\mathbf{\Phi}(t_n, \mathbf{y}_n, h) \\ \hat{\mathbf{y}}_{n+1} &= \mathbf{y}_n + h\hat{\mathbf{\Phi}}(t_n, \mathbf{y}_n, h) \end{aligned}$$