8 Single Step Methods

8.1 Initial value problems (IVP) for ODEs

Some grasp of the meaning and theory of ordinary differential equations (ODEs) is indispensable for understanding the construction and properties of numerical methods. Relevant information can be found in [52, Sect. 5.6, 5.7, 6.5].

Example 8.1.1 (Growth with limited resources). [1, Sect. 1.1]

$y : [0,T] \mapsto \mathbb{R}$: bacterial population density as a function of time

Model: autonomous logistic differential equations

$$\dot{y} = f(y) := (\alpha - \beta y)\, y \tag{8.1.1}$$

Notation (Newton): dot $\dot{} \;\hat{=}\;$ (total) derivative with respect to time t

- $y \hat{=}$ population density, $[y] = \frac{1}{\mathrm{m}^2}$
- growth rate $\alpha - \beta y$ with growth coefficients $\alpha, \beta > 0$, $[\alpha] = \frac{1}{\mathrm{s}}$, $[\beta] = \frac{\mathrm{m}^2}{\mathrm{s}}$: decreases due to more fierce competition as population density increases.

Note: we can only compute a solution of (8.1.3), when provided with an initial value $y(0)$.

The logisitc differential equation arises in autocatalytic reactions (as in haloform reaction, tin pest, binding of oxygen by hemoglobin or the spontaneous degradation of aspirin into salicylic acid and acetic acid, causing very old aspirin in sealed containers to smell mildly of vinegar):

$$A + B \longrightarrow 2B \quad \text{with rate } r = k c_A c_B \tag{8.1.2}$$

As $\dot{c}_A = -r$ and $\dot{c}_B = -r + 2r = r$ we have that $c_A + c_B = c_A(0) + c_B(0) = D$ is constant and we get two decoupled equations

$$\dot{c}_A = -k(D - c_A)c_A \tag{8.1.3}$$

$$\dot{c}_B = k(D - c_B)c_B \tag{8.1.4}$$

which are of the form of (8.1.3) with $\beta = k$, $\alpha = kD$ for the element B and $\beta = -k$, $\alpha = -kD$ for the element A.

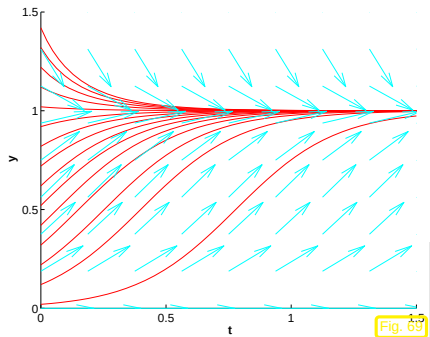


Fig. 69

Solution for different $y(0)$ ($\alpha, \beta = 5$)

By separation of variables

➥ solution of (8.1.3)

for $y(0) = y_0 > 0$

$$y(t) = \frac{\alpha y_0}{\beta y_0 + (\alpha - \beta y_0)\exp(-\alpha t)}\,, \tag{8.1.5}$$

for all $t \in \mathbb{R}$

$f'(y^*) = 0$ for $y^* \in \{0, \alpha/\beta\}$, which are the stationary points for the ODE (8.1.3). If $y(0) = y^*$ the solution will be constant in time.

◇

Example 8.1.2 (Predator-prey model). [1, Sect. 1.1] & [27, Sect. 1.1.1] initially proposed by Alfred J. Lotka in "The theory of autocatalytic chemical reactions" in 1910

Predators and prey coexist in an ecosystem. Without predators the population of prey would be governed by a simple exponential growth law. However, the growth rate of prey will decrease with increasing numbers of predators and, eventually, become negative. Similar considerations apply to the predator population and lead to an ODE model.

Model: autonomous Lotka-Volterra ODE:

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \quad \leftrightarrow \quad \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y} = \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{pmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{pmatrix}. \tag{8.1.6}$$

population sizes:

$u(t) \longrightarrow$ no. of prey at time t,
$v(t) \longrightarrow$ no. of predators at time t

vector field f for Lotka-Volterra ODE ▷

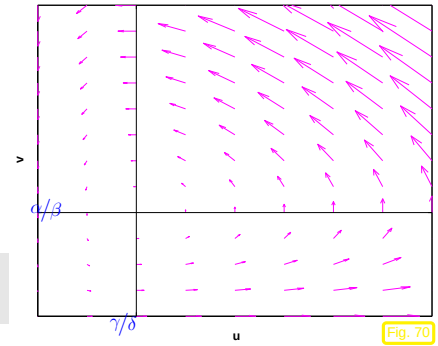
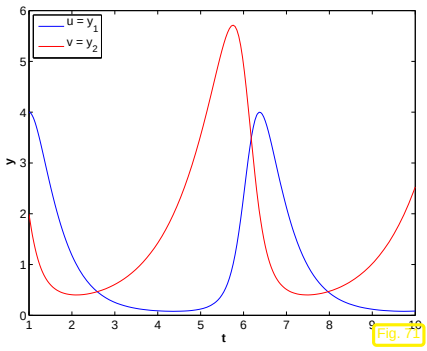


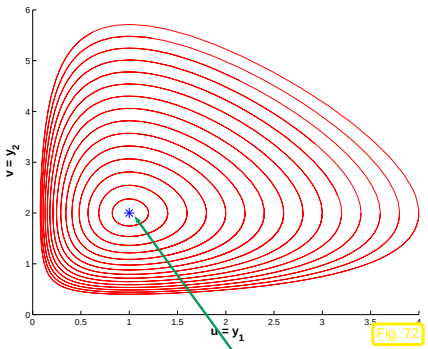
Fig. 70

Solution curves are trajectories of particles carried along by velocity field $\mathbf{f}$.

Parameter values for Fig. 70: $\alpha = 2, \beta = 1, \delta = 1, \gamma = 1$



Solution $\binom{u(t)}{v(t)}$ for $\mathbf{y}_0 := \binom{u(0)}{v(0)} = \binom{4}{2}$



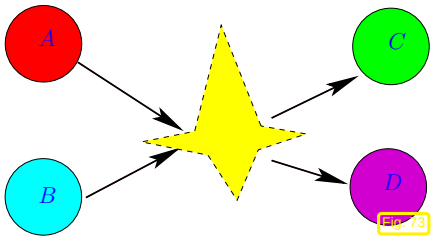
Solution curves for (8.1.6)

Parameter values for Figs. 72, 71: $\alpha = 1, \beta = 1, \delta = 1, \gamma = 2$

stationary point

◇

Example 8.1.3 (Bimolecular Reaction).



Reaction: $$\mathrm{A} + \mathrm{B} \underset{k_1}{\overset{k_2}{\rightleftharpoons}} \mathrm{C} + \mathrm{D}\,. \qquad (8.1.7)$$

with reaction constants k_1 ("forwards"), k_2 ("backwards"), $[k_1] = [k_2] = \frac{\mathrm{cm}^3}{\mathrm{mol\,s}}$.

Rule of thumb: Speed of a bimolecular reaction is proportional to the product of the concentrations of each component:

► for (8.1.7): $$\dot{c}_A = \dot{c}_B = -\dot{c}_C = -\dot{c}_D = -k_1 c_A c_B + k_2 c_C c_D\,. \qquad (8.1.8)$$

$c_A, c_B, c_C, c_D \hat{=}$ (time dependent) concentrations of components, $[c_X] = \frac{\mathrm{mol}}{\mathrm{cm}^3}$ ➥ $c_X(t) > 0; \forall t$

(8.1.8) = autonom ordinary differential equation (**??**) with

$$\mathbf{y}(t) = \begin{pmatrix} c_A(t) \\ c_B(t) \\ c_C(t) \\ c_D(t) \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = (-k_1 y_1 y_2 + k_2 y_3 y_4) \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}.$$

► Conservation of mass: $$\frac{d}{dt}(c_A(t) + c_B(t) + c_C(t) + c_D(t)) = 0$$

◇

Example 8.1.4 (Oregonator-Reaction).

Special case of a time-dependent oscillation Zhabotinski-Belousov-Reaction [21]:

$$\begin{aligned} \mathrm{BrO}_3^- + \mathrm{Br}^- &\mapsto \mathrm{HBrO}_2 \\ \mathrm{HBrO}_2 + \mathrm{Br}^- &\mapsto \mathrm{Org} \\ \mathrm{BrO}_3^- + \mathrm{HBrO}_2 &\mapsto 2\,\mathrm{HBrO}_2 + \mathrm{Ce(IV)} \\ 2\,\mathrm{HBrO}_2 &\mapsto \mathrm{Org} \\ \mathrm{Ce(IV)} &\mapsto \mathrm{Br}^- \end{aligned} \qquad (8.1.9)$$

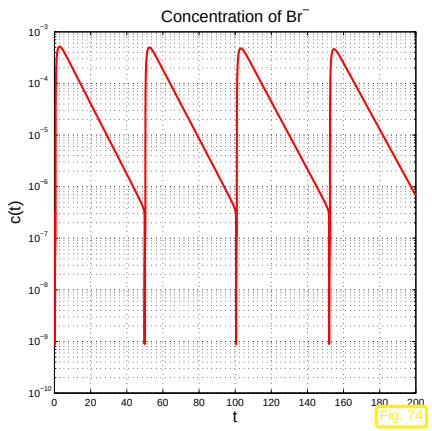
$$\begin{aligned} y_1 &:= c(\mathrm{BrO}_3^-): & \dot{y}_1 &= -k_1 y_1 y_2 - k_3 y_1 y_3\,, \\ y_2 &:= c(\mathrm{Br}^-): & \dot{y}_2 &= -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5\,, \\ y_3 &:= c(\mathrm{HBrO}_2): & \dot{y}_3 &= k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2\,, \\ y_4 &:= c(\mathrm{Org}): & \dot{y}_4 &= k_2 y_2 y_3 + k_4 y_3^2\,, \\ y_5 &:= c(\mathrm{Ce(IV)}): & \dot{y}_5 &= k_3 y_1 y_3 - k_5 y_5\,, \end{aligned} \qquad (8.1.10)$$

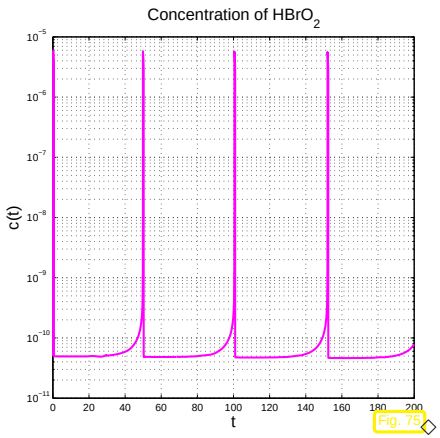
with (dimensionless) reaction constants:

$$k_1 = 1.34\,, \quad k_2 = 1.6 \cdot 10^9\,, \quad k_3 = 8.0 \cdot 10^3\,, \quad k_4 = 4.0 \cdot 10^7\,, \quad k_5 = 1.0\,.$$

► Periodical chemical reaction ➥ Video 1, Video 2

MATLAB-Simulation with initial values $y_1(0) = 0.06$, $y_2(0) = 0.33 \cdot 10^{-6}$, $y_3(0) = 0.501 \cdot 10^{-10}$, $y_4(0) = 0.03$, $y_5(0) = 0.24 \cdot 10^{-7}$:





◇

Abstract mathematical description:

Initial value problem (IVP) for first-order ordinary differential equation (ODE): (→ [52, Sect. 5.6])

$$\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \, . \tag{8.1.11}$$

- $\mathbf{f} : I \times D \mapsto \mathbb{R}^d \mathrel{\hat{=}}$ right hand side (r.h.s.) ($d \in \mathbb{N}$), given in procedural form

```
function v = f(t,y).
```

- $I \subset \mathbb{R} \mathrel{\hat{=}}$ (time)interval $\leftrightarrow$ "time variable" t
- $D \subset \mathbb{R}^d \mathrel{\hat{=}}$ state space/phase space $\leftrightarrow$ "state variable" $\mathbf{y}$ (*ger.:* Zustandsraum)
- $\Omega := I \times D \mathrel{\hat{=}}$ extended state space (of tupels $(t,\mathbf{y})$)
- $t_0 \mathrel{\hat{=}}$ initial time, $\mathbf{y}_0 \mathrel{\hat{=}}$ initial state ➢ initial conditions

Terminology: $\mathbf{f} = \mathbf{f}(\mathbf{y})$, r.h.s. does not depend on time ➔ $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ is autonomous ODE

For autonomous ODEs:
- $I = \mathbb{R}$ and r.h.s. $\mathbf{y} \mapsto \mathbf{f}(\mathbf{y})$ can be regarded as stationary vector field (velocity field)
- if $t \mapsto \mathbf{y}(t)$ is solution $\Rightarrow$ for any $\tau \in \mathbb{R}$ $t \mapsto \mathbf{y}(t+\tau)$ is solution, too.
- initial time irrelevant: canonical choice $t_0 = 0$

Note: autonomous ODEs naturally arise when modeling time-invariant systems/phenomena. All examples above led to autonomous ODEs.

Remark 8.1.5 (Conversion into autonomous ODE).

Idea: include time as an extra $d+1$-st component of an extended state vector.

This solution component has to grow linearly $\Leftrightarrow$ temporal derivative $= 1$

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ t \end{pmatrix} = \begin{pmatrix} \mathbf{z}' \\ z_{d+1} \end{pmatrix} : \quad \dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y}) \quad \leftrightarrow \quad \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) \, , \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{f}(z_{d+1},\mathbf{z}') \\ 1 \end{pmatrix} \, .$$

△

Remark 8.1.6 (From higher order ODEs to first order systems).

Ordinary differential equation of order $n \in \mathbb{N}$:

$$\boxed{\mathbf{y}^{(n)} = \mathbf{f}(t,\mathbf{y},\dot{\mathbf{y}},\dots,\mathbf{y}^{(n-1)})} \, . \tag{8.1.12}$$

✎ Notation: superscript $^{(n)} \mathrel{\hat{=}} n$-th temporal derivative t

➢ Conversion into 1st-order ODE (system of size nd)

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ \mathbf{y}^{(1)}(t) \\ \vdots \\ \mathbf{y}^{(n-1)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{pmatrix} \in \mathbb{R}^{dn} : \quad (8.1.12) \quad \leftrightarrow \quad \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) \, , \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{z}_2 \\ \mathbf{z}_3 \\ \vdots \\ \mathbf{z}_n \\ \mathbf{f}(t,\mathbf{z}_1,\dots,\mathbf{z}_n) \end{pmatrix} \, . \tag{8.1.13}$$

Note: n initial values $\mathbf{y}(t_0), \dot{\mathbf{y}}(t_0), \dots, \mathbf{y}^{(n-1)}(t_0)$ required!

△

Basic assumption: right hand side $\mathbf{f} : I \times D \mapsto \mathbb{R}^d$ locally Lipschitz continuous in $\mathbf{y}$

Definition 8.1.1 (Lipschitz continuous function). *(→ [52, Def. 4.1.4])*
$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ *is Lipschitz continuous (in the second argument), if*

$$\exists L > 0 : \quad \|\mathbf{f}(t,\mathbf{w}) - \mathbf{f}(t,\mathbf{z})\| \le L \, \|\mathbf{w} - \mathbf{z}\| \quad \forall (t,\mathbf{w}),(y,\mathbf{z}) \in \Omega \, .$$

Definition 8.1.2 (Local Lipschitz continuity). *(→ [52, Def. 4.1.5])*

✎ Notation: $D_{\mathbf{y}}\mathbf{f} \mathrel{\hat{=}}$ derivative of $\mathbf{f}$ w.r.t. state variable (= Jacobian $\in \mathbb{R}^{d,d}$!)

A simple criterion for local Lipschitz continuity:

Lemma 8.1.3 (Criterion for local Liptschitz continuity).
If $\mathbf{f}$ and $D_{\mathbf{y}}\mathbf{f}$ are continuous on the extended state space Ω, then $\mathbf{f}$ is locally Lipschitz continuous(→ Def. 8.1.2).

Theorem 8.1.4 (Theorem of Peano & Picard-Lindelöf). *[1, Satz II(7.6)], [52, Satz 6.5.1]*
If $\mathbf{f} : \hat{\Omega} \mapsto \mathbb{R}^d$ is locally Lipschitz continuous (→ Def. 8.1.2) then for all initial conditions $(t_0,\mathbf{y}_0) \in \hat{\Omega}$ the IVP (8.1.11) *has a solution $\mathbf{y} \in C^1(J(t_0,\mathbf{y}_0),\mathbb{R}^d)$ with maximal (temporal) domain of definition $J(t_0,\mathbf{y}_0) \subset \mathbb{R}$.*

Remark 8.1.7 (Domain of definition of solutions of IVPs).

Solutions of an IVP have an intrinsic maximal domain of definition

! domain of definition/domain of existence $J(t_0, \mathbf{y}_0)$ usually depends on $(t_0, \mathbf{y}_0)$!

Terminology: if $J(t_0, \mathbf{y}_0) = I$ ➥ solution $\mathbf{y} : I \mapsto \mathbb{R}^d$ is global. △

Notation: for autonomous ODE we always have $t_0 = 0$, therefore write $J(\mathbf{y}_0) := J(0, \mathbf{y}_0)$.

In light of Rem. 8.1.5 and Thm. 8.1.4: we consider only

$$\text{autonomous IVP:} \quad \boxed{\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0} \; , \tag{8.1.14}$$

with locally Lipschitz continuous (→ Def. 8.1.2) right hand side $\mathbf{f}$.

Assumption 8.1.5 (Global solutions).
All solutions of (8.1.14) *are global:* $J(\mathbf{y}_0) = \mathbb{R}$ *for all* $\mathbf{y}_0 \in D$.

Change of perspective: fix "time of interest" $t \in \mathbb{R} \setminus \{0\}$

➢ mapping $\mathbf{\Phi}^t : \begin{cases} D & \mapsto & D \\ \mathbf{y}_0 & \mapsto & \mathbf{y}(t) \end{cases}$, $t \mapsto \mathbf{y}(t)$ solution of IVP (8.1.14) ,

is well-defined mapping of the state space into itself, by Thm. 8.1.4 and Ass. 8.1.5

Now, we may also let t vary, which spawns a *family* of mappings $\{\mathbf{\Phi}^t\}$ of the state space into itself. However, it can also be viewed as a mapping with two arguments, a time t and an initial state value $\mathbf{y}_0$!

Definition 8.1.6 (Evolution operator).
Under Assumption 8.1.5 the mapping

$$\mathbf{\Phi} : \begin{cases} \mathbb{R} \times D & \mapsto & D \\ (t, \mathbf{y}_0) & \mapsto & \mathbf{\Phi}^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases} ,$$

where $t \mapsto \mathbf{y}(t) \in C^1(\mathbb{R}, \mathbb{R}^d)$ *is the unique (global) solution of the IVP* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$, *is the evolution operator for the ODE* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

Note: $t \mapsto \mathbf{\Phi}^t \mathbf{y}_0$ describes the solution of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ for $\mathbf{y}(0) = \mathbf{y}_0$ (a trajectory)

Remark 8.1.8 (Group property of autonomous evolutions).

Under Assumption 8.1.5 the evolution operator gives rise to a group of mappings $D \mapsto D$:

$$\mathbf{\Phi}^s \circ \mathbf{\Phi}^t = \mathbf{\Phi}^{s+t} \quad , \quad \mathbf{\Phi}^{-t} \circ \mathbf{\Phi}^t = Id \quad \forall t \in \mathbb{R} \; . \tag{8.1.15}$$

This is a consequence of the uniqueness theorem Thm. 8.1.4. It is also intuitive: following an evolution up to time t and then for some more time s leads us to the same final state as observing it for the whole time $s + t$. △

8.2 Euler methods

Targeted: initial value problem (8.1.11)

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \; . \tag{8.1.11}$$

Sought: *approximate* solution of (8.1.11) on $[t_0, T]$ up to final time $T \neq t_0$

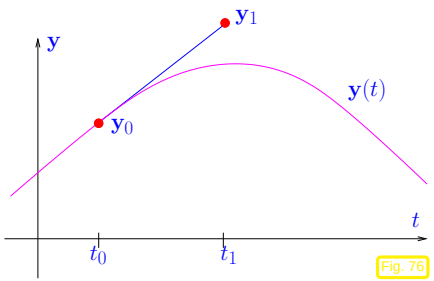
However, the solution of an initial value problem is a *function* $J(t_0, \mathbf{y}_0) \mapsto \mathbb{R}^d$ and requires a suitable approximate representation. We postpone this issue here and first study a geometric approach to numerical integration.

numerical integration = approximate solution of initial value problems for ODEs

(Please distinguish from "numerical quadrature", see Ch. 7.)

Idea: ❶ timestepping: successive approximation of evolution on *small* intervals $[t_{k-1}, t_k]$, $k = 1, \ldots, N$, $t_N := T$,

❷ approximation of solution on $[t_{k-1}, t_k]$ by tangent curve to current initial condition.



Example 8.2.1 (Visualization of explicit Euler method).

explicit Euler method (Euler 1768)

◁ First step of explicit Euler method ($d = 1$):

Slope of tangent $= f(t_0, \mathbf{y}_0)$

$\mathbf{y}_1$ serves as initial value for next step!

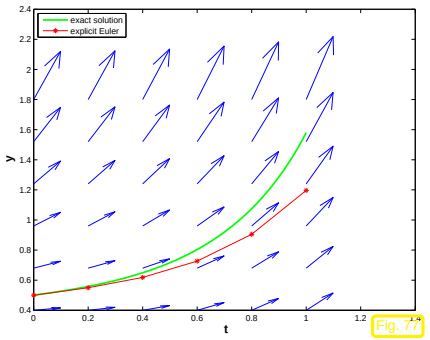
IVP for Riccati differential equation, see Ex. **??**

$$\dot{y} = y^2 + t^2 \,. \qquad \textbf{(??)}$$

Here: $y_0 = \frac{1}{2}$, $t_0 = 0$, $T = 1$, ▷

— $\hat{=}$ "Euler polygon" for uniform timestep $h = 0.2$

$\mapsto$ $\hat{=}$ tangent field of Riccati ODE



◇

Formula: explicit Euler method generates a *sequence* $(\mathbf{y}_k)_{k=0}^N$ by the recursion

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_k, \mathbf{y}_k) \,, \quad k = 0, \dots, N-1 \quad ,} \qquad (8.2.1)$$

with local (size of) timestep (stepsize) $h_k := t_{k+1} - t_k$.

Remark 8.2.2 (Explicit Euler method as difference scheme).

(8.2.1) by approximating derivative $\frac{d}{dt}$ by forward difference quotient on a (temporal) mesh $\mathcal{M} := \{t_0, t_1, \dots, t_N\}$:

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_k, \mathbf{y}_h(t_k)) \,, \quad k = 0, \dots, N-1 \,. \qquad (8.2.2)$$

Difference schemes follow a simple policy for the *discretization* of differential equations: replace all derivatives by difference quotients connecting solution values on a set of discrete points (the mesh).

△

Why forward difference quotient and not backward difference quotient? Let's try!

On (temporal) mesh $\mathcal{M} := \{t_0, t_1, \dots, t_N\}$ we obtain

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_{k+1}, \mathbf{y}_h(t_{k+1})) \,, \quad k = 0, \dots, N-1 \,. \qquad (8.2.3)$$

Backward difference quotient

This leads to another simple timestepping scheme analoguous to (8.2.1):

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \,, \quad k = 0, \dots, N-1} \,, \qquad (8.2.4)$$

with local timestep (stepsize) $h_k := t_{k+1} - t_k$.

(8.2.4) = implicit Euler method

Note: (8.2.4) requires solving of a (possibly non-linear) system of equations to obtain $\mathbf{y}_{k+1}$!
(➤ Terminology "implicit")

Remark 8.2.3 (Feasibility of implicit Euler timestepping).

Consider autonomous ODE and assume continuously differentiable right hand side: $\mathbf{f} \in C^1(D, \mathbb{R}^d)$.

(8.2.4) $\leftrightarrow$ h-dependent non-linear system of equations:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \quad \Leftrightarrow \quad G(h, \mathbf{y}_{k+1}) = 0 \quad \text{with} \quad G(h, \mathbf{z}) := \mathbf{z} - h\mathbf{f}(\mathbf{z}) - \mathbf{y}_k \,.$$

Partial derivative:

$$\frac{dG}{d\mathbf{z}}(0, \mathbf{z}) = \mathbf{I}$$

Implicit function theorem: for *sufficiently small* $|h|$ the equation $G(h, \mathbf{z}) = 0$ defines a continuous function $\mathbf{z} = \mathbf{z}(h)$.

△

How to interpret the sequence $(\mathbf{y}_k)_{k=0}^N$ from (8.2.1)?

By "geometric insight" we expect: $\boxed{\mathbf{y}_k \approx \mathbf{y}(t_k)}$

(Throughout, we use the notation $\mathbf{y}(t)$ for the exact solution of an IVP.)

If we are merely interested in the final state $\mathbf{y}(T)$, then the explicit Euler method will give us the answer $\mathbf{y}_N$.

If we are interested in an approximate solution $\mathbf{y}_h(t) \approx \mathbf{y}(t)$ as a function $[t_0, T] \mapsto \mathbb{R}^d$, we have to do

post-processing = reconstruction of a function from $\mathbf{y}_k$, $k = 0, \dots, N$

Technique: *interpolation*, see Ch. 5

Simplest option: piecewise linear interpolation (→ Sect. **??**) ➔ Euler polygon, see Fig. 77.

Abstract single step methods

Recall Euler methods for autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$:

explicit Euler: $\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k)$,
implicit Euler: $\mathbf{y}_{k+1}$: $\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_{k+1})$.

Both formulas provide a mapping

$$(\mathbf{y}_k, h_k) \mapsto \mathbf{\Psi}(h, \mathbf{y}_k) := \mathbf{y}_{k+1} \ . \tag{8.2.5}$$

Recall the interpretation of the $\mathbf{y}_k$ as approximations of $\mathbf{y}(t_k)$:

$$\mathbf{\Psi}(h, \mathbf{y}) \approx \mathbf{\Phi}^h \mathbf{y} \ , \tag{8.2.6}$$

where $\mathbf{\Phi}$ is the evolution operator (→ Def. 8.1.6) for $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

The Euler methods provide approximations for evolution operator for ODEs

This is what every single step method does: it tries to approximate the evolution operator $\mathbf{\Phi}$ for an ODE by a mapping of the type (8.2.5).

➔ mapping $\mathbf{\Psi}$ from (8.2.5) is called discrete evolution.

Vice versa: a mapping $\mathbf{\Psi}$ as in (8.2.5) defines a single step method.

☞ In a sense, a single step method defined through its associated discrete evolution does not approximate and initial value problem, but tries to approximate an ODE.

Definition 8.2.1 (Single step method (for autonomous ODE)).
Given a discrete evolution $\mathbf{\Psi} : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^d$, *an initial state* $\mathbf{y}_0$, *and a temporal mesh* $\mathcal{M} := \{t_0 < t_1 < \cdots < t_N = T\}$ *the recursion*

$$\mathbf{y}_{k+1} := \mathbf{\Psi}(t_{k+1} - t_k, \mathbf{y}_k) \ , \quad k = 0, \dots, N-1 \ , \tag{8.2.7}$$

defines a single step method (SSM, ger.: *Einschrittverfahren) for the autonomous IVP* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$.

Procedural view of discrete evolutions:

$\mathbf{\Psi}^h \mathbf{y}$ ⟷ `function y1 = esvstep(h,y0)` .
(`function y1 = esvstep( rhs,h,y0)`)

✎ Notation: $\mathbf{\Psi}^h \mathbf{y} := \mathbf{\Psi}(h, \mathbf{y})$

Concept of single step method according to Def. 8.2.1 can be generalized to non-autonomous ODEs, which leads to recursions of the form:

$$\mathbf{y}_{k+1} := \mathbf{\Psi}(t_k, t_{k+1}, \mathbf{y}_k) \ , \quad k = 0, \dots, N-1 \ ,$$

for discrete evolution defined on $I \times I \times D$.

Remark 8.2.4 (Notation for single step methods).

Many authors specify a single step method by writing down the first step:

$$\mathbf{y}_1 = \text{expression in } \mathbf{y}_0 \text{ and } \mathbf{f} \ .$$

Also this course will sometimes adopt this practice. △

8.3 Convergence of single step methods

Important issue: accuracy of approximation $\mathbf{y}_k \approx \mathbf{y}(t_k)$?

As in the case of composite numerical quadrature, see Sect. 7.2: in general impossible to predict error $\|\mathbf{y}_N - \mathbf{y}(T)\|$ for particular choice of timesteps.

Tractable: asymptotic behavior of error for timestep $h := \max_k h_k \to 0$

Will tell us asymptotic gain in accuracy for extra computational effort.
(computational effort = no. of $\mathbf{f}$-evaluations)

Example 8.3.1 (Speed of convergence of Euler methods).

- IVP for Riccati ODE (**??**) on $[0,1]$
- explicit Euler method (8.2.1) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- Error $\mathrm{err}_h := |y(1) - y_N|$

Observation:

algebraic convergence $\mathrm{err}_h = O(h)$

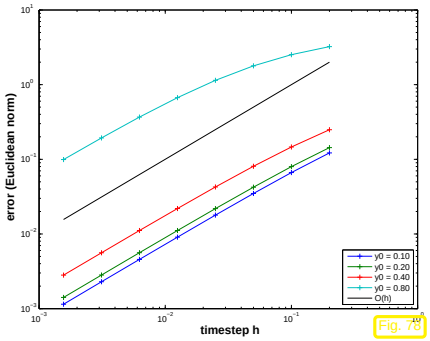


Fig. 75

- IVP for logistic ODE, see Ex. 8.1.1

$$\dot{y} = \lambda y(1-y) \quad , \quad y(0) = 0.01 \; .$$

- Explicit and implicit Euler methods (8.2.1)/(8.2.4) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- Monitored: Error at final time $E(h) := |y(1) - y_N|$

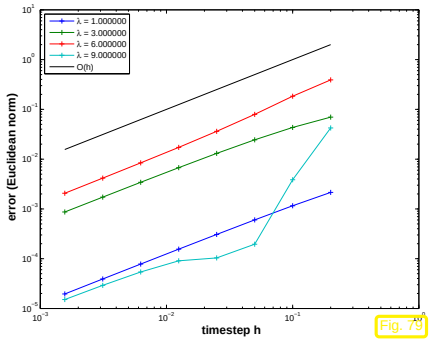


Fig. 76

explicit Euler method

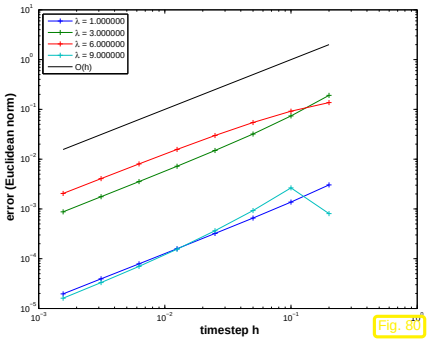


Fig. 77

implicit Euler method

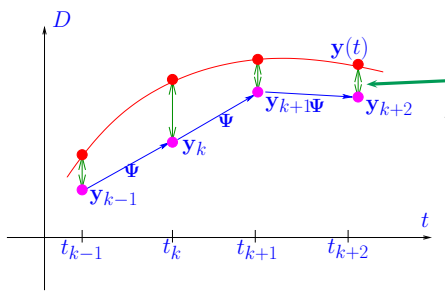
$O(h)$ algebraic convergence in both cases ◇

Convergence analysis for explicit Euler method (8.2.1) for autonomous IVP (8.1.11) with sufficiently smooth and (*globally*) Lipschitz continuous $\mathbf{f}$, that is,

$$\exists L > 0: \quad \|\mathbf{f}(t,\mathbf{y}) - \mathbf{f}(t,\mathbf{z})\| \leq L\,\|\mathbf{y} - \mathbf{z}\| \quad \forall \mathbf{y}, \mathbf{z} \in D \; . \tag{8.3.1}$$

Recall: recursion for explicit Euler method

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) \; , \quad k = 1, \ldots, N-1 \; . \tag{8.2.1}$$



Error sequence: $\mathbf{e}_k := \mathbf{y}_k - \mathbf{y}(t_k)$.

◁ — $\hat{=}$ $t \mapsto \mathbf{y}(t)$
— $\hat{=}$ Euler polygon,
• $\hat{=}$ $\mathbf{y}(t_k)$,
• $\hat{=}$ $\mathbf{y}_k$,
⟶ $\hat{=}$ discrete evolution $\Psi^{t_{k+1}-t_k}$

① Abstract splitting of error:

Here and in what follows we rely on the abstract concepts of the evolution operator $\boldsymbol{\Phi}$ associated with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ (→ Def. 8.1.6) and discrete evolution operator $\boldsymbol{\Psi}$ defining the explicit Euler single step method, see Def. 8.2.1:

$$(8.2.1) \quad \Rightarrow \quad \boldsymbol{\Psi}^h \mathbf{y} = \mathbf{y} + h\mathbf{f}(\mathbf{y}) \; . \tag{8.3.2}$$

We argue that in this context the abstraction pays off, because it helps elucidate a general technique for the convergence analysis of single step methods.

Fundamental error splitting

$$\begin{aligned} \mathbf{e}_{k+1} &= \boldsymbol{\Psi}^{h_k}\mathbf{y}_k - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) \\ &= \underbrace{\boldsymbol{\Psi}^{h_k}\mathbf{y}_k - \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k)}_{\text{propagated error}} \\ &\quad + \underbrace{\boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k)}_{\text{one-step error}} \; . \end{aligned} \tag{8.3.3}$$

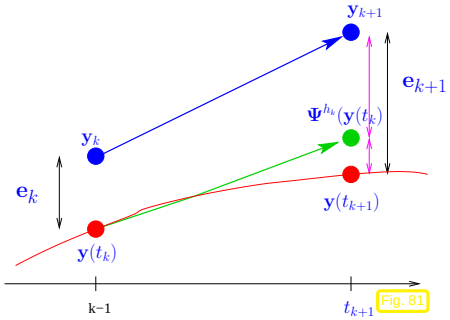


Fig. 81

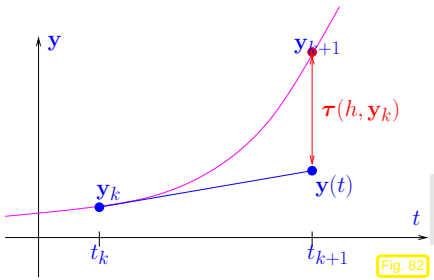


Fig. 82

one-step error:

$$\boldsymbol{\tau}(h,\mathbf{y}) := \boldsymbol{\Psi}^h\mathbf{y} - \boldsymbol{\Phi}^h\mathbf{y} \; . \tag{8.3.4}$$

◁ geometric visualisation of one-step error for explicit Euler method (8.2.1), *cf.* Fig. 76.

✎ notation: $t \mapsto \mathbf{y}(t) \hat{=}$ (unique) solution of IVP, *cf.* Thm. 8.1.4.

② **Estimate for one-step error**:

Geomtric considerations: distance of a smooth curve and its tangent shrinks as the square of the distance to the intersection point (curve locally looks like a parabola in the $\xi - \eta$ coordinate system, see Fig. 84).

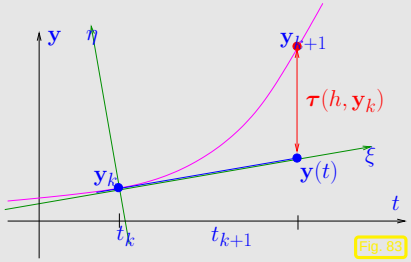


Fig. 83

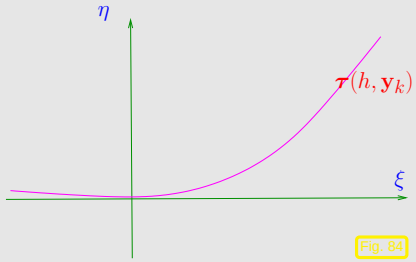


Fig. 84

Analytic considerations: recall Taylor's formula for function $\mathbf{y} \in C^{K+1}$

$$\mathbf{y}(t+h) - \mathbf{y}(t) = \sum_{j=0}^{K} \mathbf{y}^{(j)}(t)\frac{h^j}{j!} + \underbrace{\int_t^{t+h} f^{(K+1)}(\tau)\frac{(t+h-\tau)^K}{K!}\,\mathrm{d}\tau}_{= \frac{f^{(K+1)}(\xi)}{K!}h^{K+1}} \; , \tag{8.3.5}$$

for some $\xi \in [t, t+h]$

$\Rightarrow$ if $\mathbf{y} \in C^2([0,T])$, then

$$\mathbf{y}(t_{k+1}) - \mathbf{y}(t_k) = \dot{\mathbf{y}}(t_k)h_k + \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \quad \text{for some } t_k \le \xi_k \le t_{k+1}$$

$$= \mathbf{f}(\mathbf{y}(t_k))h_k + \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \; ,$$

since $t \mapsto \mathbf{y}(t)$ solves the ODE, which implies $\dot{\mathbf{y}}(t_k) = \mathbf{f}(\mathbf{y}(t_k))$. This leads to an expression for the one-step error from (8.3.4)

$$\begin{aligned} \boldsymbol{\tau}(h_k,\mathbf{y}(t_k)) &= \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \mathbf{y}(t_k + h_k) \\ &\overset{(8.3.2)}{=} \mathbf{y}(t_k) + h_k\mathbf{f}(\mathbf{y}(t_k)) - \mathbf{y}(t_k) - \mathbf{f}(\mathbf{y}(t_k))h_k + \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \\ &= \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \; . \end{aligned} \tag{8.3.6}$$

Sloppily speaking, we observe $\boxed{\boldsymbol{\tau}(h_k,\mathbf{y}(t_k)) = O(h_k^2)}$ uniformly for $h_k \to 0$.

③ **Estimate for the propageted error** from (8.3.3)

$$\begin{aligned} \left\| \boldsymbol{\Psi}^{h_k}\mathbf{y}_k - \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) \right\| &= \|\mathbf{y}_k + h_k\mathbf{f}(\mathbf{y}_k) - \mathbf{y}(t_k) - h_k\mathbf{f}(\mathbf{y}(t_k))\| \\ &\overset{(8.3.1)}{\le} (1 + Lh_k)\,\|\mathbf{y}_k - \mathbf{y}(t_k)\| \; . \end{aligned} \tag{8.3.7}$$

③ *Recursion* for error norms $\epsilon_k := \|\mathbf{e}_k\|$ by △-inequality:

$$\epsilon_{k+1} \le (1 + h_k L)\epsilon_k + \rho_k \; , \quad \rho_k := \tfrac{1}{2}h_k^2 \max_{t_k \le \tau \le t_{k+1}} \|\ddot{\mathbf{y}}(\tau)\| \; . \tag{8.3.8}$$

Taking into account $\epsilon_0 = 0$ this leads to

$$\epsilon_k \le \sum_{l=1}^{k} \prod_{j=1}^{l-1} (1 + Lh_j)\, \rho_l \;, \quad k = 1, \dots, N \;. \tag{8.3.9}$$

Use the elementary estimate $(1 + Lh_j) \le \exp(Lh_j)$ (by convexity of exponential function):

$$(8.3.9) \quad \Rightarrow \quad \epsilon_k \le \sum_{l=1}^{k} \prod_{j=1}^{l-1} \exp(Lh_j) \cdot \rho_l = \sum_{l=1}^{k} \exp(L \sum_{j=1}^{l-1} h_j) \rho_l \;.$$

Note: $\sum_{j=1}^{l-1} h_j \le T$ for final time T

$$\epsilon_k \le \exp(LT) \sum_{l=1}^{k} \rho_l \le \exp(LT) \max_k \frac{\rho_k}{h_k} \sum_{l=1}^{k} h_l$$
$$\le T \exp(LT) \max_{l=1,\dots,k} h_l \cdot \max_{t_0 \le \tau \le t_k} \|\ddot{\mathbf{y}}(\tau)\| \;.$$

$$\|\mathbf{y}_k - \mathbf{y}(t_k)\| \le T \exp(LT) \max_{l=1,\dots,k} h_l \cdot \max_{t_0 \le \tau \le t_k} \|\ddot{\mathbf{y}}(\tau)\| \;.$$

Total error arises from accumulation of one-step errors!

- error bound $= O(h)$, $h := \max_l h_l$ (➤ 1st-order algebraic convergence)
- Error bound grows *exponentially* with the length T of the integration interval.

Most commonly used single step methods display algebraic convergence of integer order with respect to the meshwidth $h := \max_k h_k$. This offers a criterion for gauging their quality.

The sequence $(\mathbf{y}_k)_k$ generated by a

single step method (→ Def. 8.2.1) of order (of consistency) $p \in \mathbb{N}$

for $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ on a mesh $\mathcal{M} := \{t_0 < t_1 < \dots < t_N = T\}$ satisfies

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \le Ch^p \quad \text{for} \quad h := \max_{k=1,\dots,N} |t_k - t_{k-1}| \to 0 \;,$$

with $C > 0$ independent of $\mathcal{M}$, provided that $\mathbf{f}$ is *sufficiently smooth*.

8.4 Structure Preservation

Example 8.4.1 (Euler method for pendulum equation).

Hamiltonian form of equations of motion for pendulum

$$\text{angular velocity} \quad p := \dot{\alpha} \quad \Rightarrow \quad \frac{d}{dt}\begin{pmatrix} \alpha \\ p \end{pmatrix} = \begin{pmatrix} p \\ -\frac{g}{l}\sin\alpha \end{pmatrix} \;, \quad g = 9.8, l = 1 \;. \tag{8.4.1}$$

- numerical solution with explicit/implicit Euler method (8.2.1)/(8.2.4),
- constant time-step $h = T/N$, end time $T = 5$ fixed, $N \in \{50, 100, 200\}$,
- initial value: $\alpha(0) = \pi/4$, $p(0) = 0$.

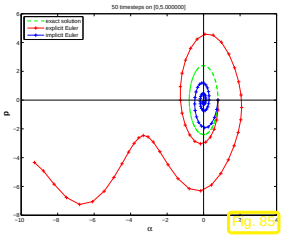


Fig. 85

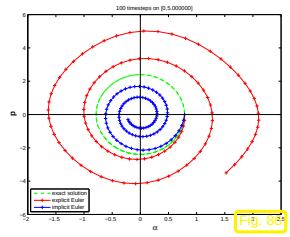


Fig. 86

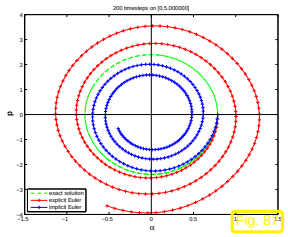


Fig. 87

Behavior of the computed energy: kinetic energy : $E_{\text{kin}}(t) = \frac{1}{2}p(t)^2$
potential energy : $E_{\text{pot}}(t) = -\frac{g}{l}\cos\alpha(t)$

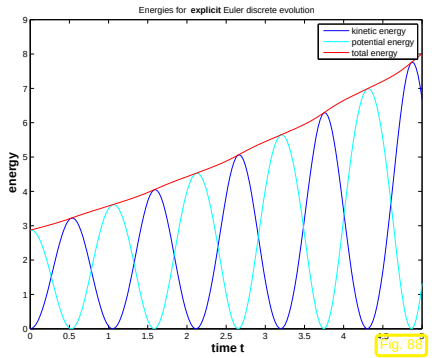


Fig. 88

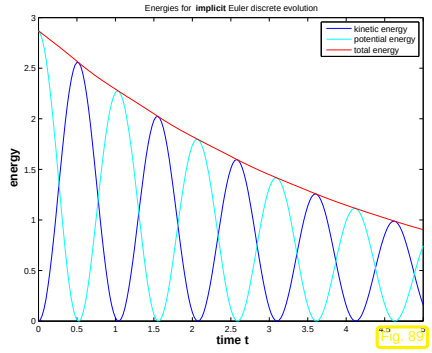


Fig. 89

☞ explicit Euler: increase of total energy

☞ implicit Euler: decrease of total energy ("numerical friction")

◇

Example 8.4.2 (Euler method for long-time evolution).

Initial value problem for , $D = \mathbb{R}^2$:

$$\dot{\mathbf{y}} = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix} \quad , \quad \mathbf{y}(0) = \mathbf{y}_0 \quad \blacktriangleright \quad \mathbf{y}(t) = \begin{pmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{pmatrix} \mathbf{y}_0 \, .$$

▶ Note that $I(\mathbf{y}) = \|\mathbf{y}\|$ is constant.
(movement with constant velocity on the circle)

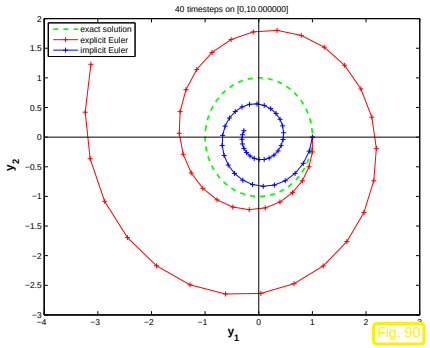


Fig. 90

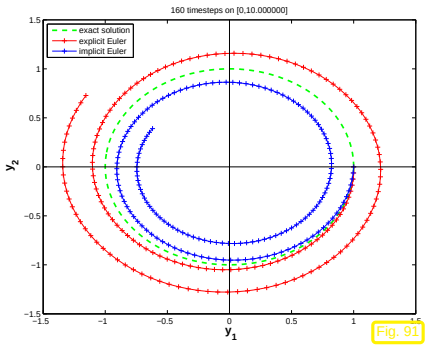


Fig. 91

☞ explicit Euler: numerical solution flyes away

☞ implicit Euler: numerical solution falls off into the center

◇

8.4.1 Implicit Midpoint Rule

Can we avoid the energy drift ?

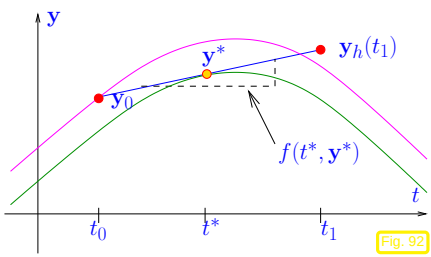


Fig. 92

Idea: Approximate solution through $(t_0, \mathbf{y}_0)$ on $[t_0, t_1]$ via

- linear polynomial through $(t_0, \mathbf{y}_0)$
- with slope $f(t^*, \mathbf{y}^*)$,
  $t^* := \frac{1}{2}(t_0 + t_1)$, $\mathbf{y}^* = \frac{1}{2}(\mathbf{y}_0 + \mathbf{y}_1)$

◁ — $\hat{=}$ solution through (t_0, y_0),
— $\hat{=}$ solution through (t^*, y^*),
— $\hat{=}$ tangent at — in (t^*, y^*).

Apply on small time intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{N-1}, t_N]$ ➤ implicit midpoint rule

▶ via implicit midpoint rule generated approximation $\mathbf{y}_{k+1}$ for $\mathbf{y}(t_k)$ fulfils

$$\boxed{\mathbf{y}_{k+1} := \mathbf{y}_h(t_{k+1}) = \mathbf{y}_k + h_k \mathbf{f}(\tfrac{1}{2}(t_k + t_{k+1}), \tfrac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})) \, , \quad k = 0, \ldots, N-1} \quad , \qquad (8.4.2)$$

with local (Time)step $h_k := t_{k+1} - t_k$.

Note: (8.4.2) requires soltion of a (evtl. non-linear) equation for $\mathbf{y}_{k+1}$!
(➤ "'implicit")

Remark 8.4.3 (Implicit midpoint rule as difference method).

(8.4.2) from the approximation of time-derivative $\frac{d}{dt}$ via centred difference quotients on time-grid $\mathcal{G} := \{t_0, t_1, \ldots, t_N\}$:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_h(t_{k+1}) - \mathbf{y}_h(t_k)}{h_k} = \mathbf{f}(\tfrac{1}{2}(t_k + t_{k+1}), \tfrac{1}{2}(\mathbf{y}_h(t_k) + \mathbf{y}(t_{k+1})), \; k = 0, \ldots, N-1 \, .$$

△

Example 8.4.4 (Implicit midpoint rule for logistic equation).

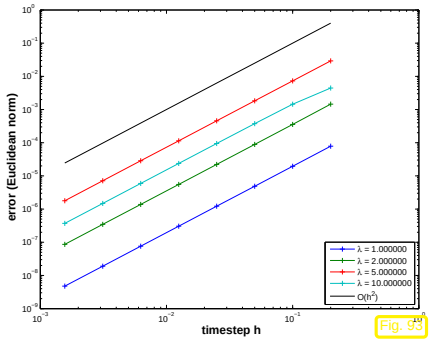


Fig. 93

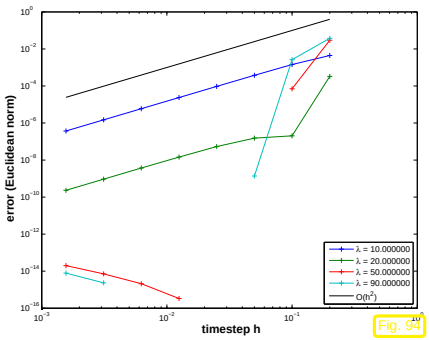


Fig. 94

λ small: $O(h^2)$-convergence (asymptotical)

λ large: stable for all time steps h !

◇

Example 8.4.5 (Implicit midpoint rule for circular motion).

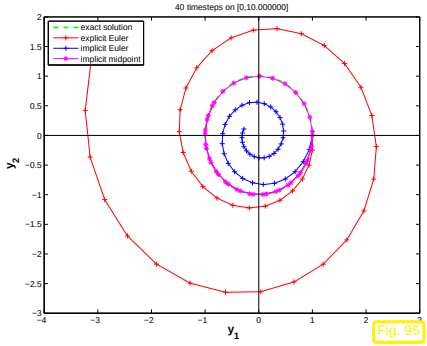


Fig. 95

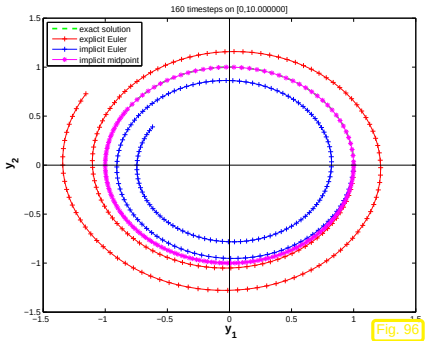


Fig. 96

☞ Implicit midpoint rule: perfect conservation of length !

◇

Example 8.4.6 (Implicit midpoint rule for pendulum).

Initial values and problem as in Bsp. 8.4.1

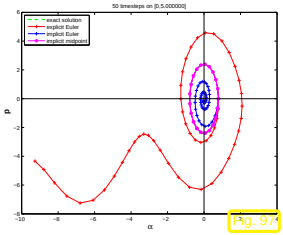


Fig. 97

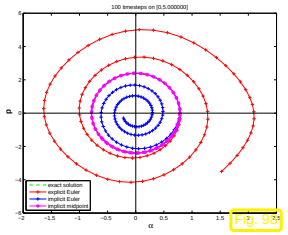


Fig. 98

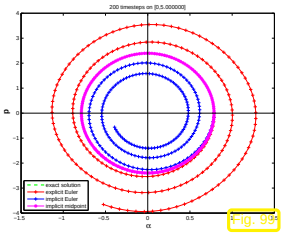


Fig. 99

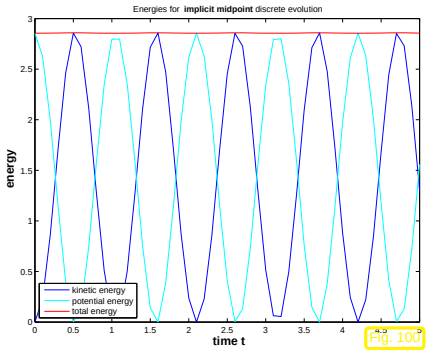


Fig. 100

◁ Behavior of the energy of the numerical solution computed with the midpoint rule (8.4.2), $N = 50$.

No energy drift although large time step)

◇

8.4.2 Störmer-Verlet Method [?]

use the idea of Euler method for an equation of 2nd order

? $$\ddot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \; . \tag{8.4.3}$$

Given $\mathbf{y}_{k-1} \approx \mathbf{y}(t_{k-1})$, $\mathbf{y}_k \approx \mathbf{y}(t_k)$ approximate $\mathbf{y}(t)$ on $[t_{k-1}, t_{k+1}]$ via

- parabola $\mathbf{p}(t)$ through $(t_{k-1}, \mathbf{y}_{k-1})$, $(t_k, \mathbf{y}_k)$ $(*)$,
- with $\ddot{\mathbf{p}}(t_k) = \mathbf{f}(\mathbf{y}_k)$ $(*)$.

$(*)$ ➜ Parabola is uniquely determined.

▶ $$\mathbf{y}_{k+1} := \mathbf{p}(t_{k+1}) \approx \mathbf{y}(t_{k+1})$$

▶ Störmer-Verlet method for (8.4.3) (time-grid $\mathcal{G} := \{t_0, t_1, \dots, t_N\}$):

$$\mathbf{y}_{k+1} = -\frac{h_k}{h_{k-1}}\mathbf{y}_{k-1} + \left(1 + \frac{h_k}{h_{k-1}}\right)\mathbf{y}_k + \tfrac{1}{2}(h_k^2 + h_k h_{k-1})\mathbf{f}(t_k, \mathbf{y}_k) \; , \quad k = 1, \dots, N-1 \; . \tag{8.4.4}$$

In case of a fixed time step h:

$$\mathbf{y}_{k+1} = -\mathbf{y}_{k-1} + 2\mathbf{y}_k + h^2\mathbf{f}(t_k, \mathbf{y}_k) \; , \quad k = 1, \dots, N-1 \; . \tag{8.4.5}$$

Note: (8.4.4) does not require the solution of an equation (➜ explicit method)

We say: $\mathbf{y}_{k+1} = \mathbf{y}_{k+1}(\mathbf{y}_k, \mathbf{y}_{k-1})$ ➜ (8.4.4) is a two-step method
(explicit/implicit Euler method, midpoint rule = one-step method)

Remark 8.4.7 (Störmer-Verlet method as difference method).

(8.4.5) from the approximation of the second time-derivative by a second centered difference quotients on time-grid $\mathcal{G} := \{t_0, t_1, \dots, t_N\}$: for cosntant time-step $h > 0$

$$\ddot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \longleftrightarrow \quad \frac{\frac{\mathbf{y}_h(t_{k+1}) - \mathbf{y}_h(t_k)}{h} - \frac{\mathbf{y}_h(t_k) - \mathbf{y}_h(t_{k-1})}{h}}{h} = \frac{\mathbf{y}_h(t_{k+1}) - 2\mathbf{y}_h(t_k) + \mathbf{y}_h(t_{k-1})}{h^2} = \mathbf{f}(\mathbf{y}_h(t_k)) \; .$$

△

Remark 8.4.8 (Initialisation of the Störmer-Verlet method).

Initial values for (8.4.3)$\mathbf{y}(0) = \mathbf{y}_0$, $\dot{\mathbf{y}}(0) = \mathbf{v}_0$

- use virtual moment $t_{-1} := t_0 - h_0$
- apply (8.4.5) to $[t_{-1}, t_1]$:

$$\mathbf{y}_1 = -\mathbf{y}_{-1} + 2\mathbf{y}_0 + h_0^2 \mathbf{f}(t_0, \mathbf{y}_0) \ . \tag{8.4.6}$$

- centered difference quotient on $[t_{-1}, t_1]$:

$$\frac{\mathbf{y}_1 - \mathbf{y}_{-1}}{2h_0} = \mathbf{v}_0 \ . \tag{8.4.7}$$

compute $\mathbf{y}_1$ from (8.4.6) **&** (8.4.7) △

Example 8.4.9 (Störmer-Verlet method for pendulum).

- (8.4.5)
- initialisation via. 8.4.8
- constant time step $h := T/N$, $N \in \mathbb{N}$ time steps
- reference solution via very precise integration ode45()
- $\alpha_0 = \pi/2$, $p_0 = 0$, $T = 5$, vgl. Bsp. 8.4.1
- Number of time steps: $N = 40$

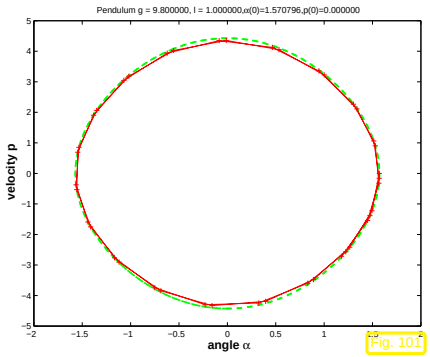


Fig. 101

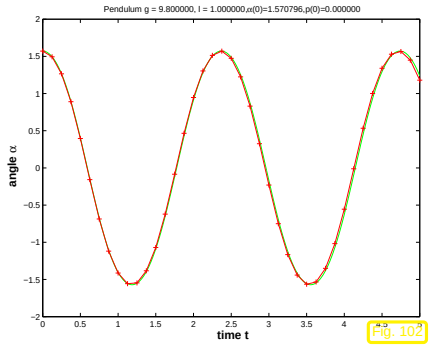


Fig. 102

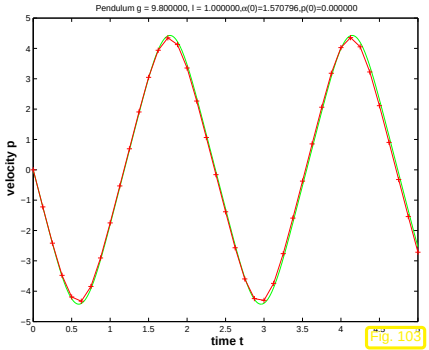


Fig. 103

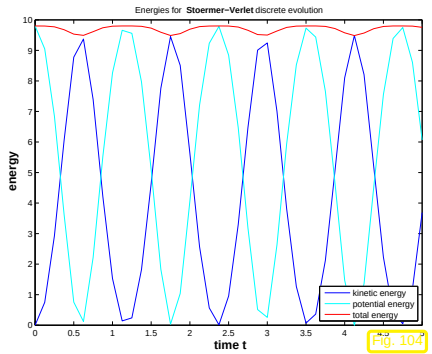


Fig. 104

☞ No energy drift thought large time steps

perfect periodical orbits **!**

On the contrary: Ex. 8.4.1

◇

Remark 8.4.10 (One-step formulation of the Störmer-Verlet method).

In case of constant time step, see (8.4.5), analogously to the refromulation of a 2nd order equation to

a fisrt order equation, see (8.1.13): with $\mathbf{v}_{k+\frac{1}{2}} := \frac{\mathbf{y}_{k+1}-\mathbf{y}_k}{h}$

$$\ddot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \longleftrightarrow \quad \begin{aligned} \dot{\mathbf{y}} &= \mathbf{v} \,, \\ \dot{\mathbf{v}} &= \mathbf{f}(\mathbf{y}) \,. \end{aligned}$$

$$\updownarrow \qquad\qquad\qquad \updownarrow$$

$$\mathbf{y}_{k+1} - 2\mathbf{y}_k + \mathbf{y}_{k-1} = h_k^2 \mathbf{f}(\mathbf{y}_k) \quad \longleftrightarrow \quad \boxed{\begin{aligned} \mathbf{v}_{k+\frac{1}{2}} &= \mathbf{v}_k + \tfrac{h}{2}\mathbf{f}(\mathbf{y}_k) \,, \\ \mathbf{y}_{k+1} &= \mathbf{y}_k + h\mathbf{v}_{k+\frac{1}{2}} \,, \\ \mathbf{v}_{k+1} &= \mathbf{v}_{k+\frac{1}{2}} + \tfrac{h}{2}\mathbf{f}(\mathbf{y}_{k+1}) \,. \end{aligned}}$$

two-steps method — one-step method

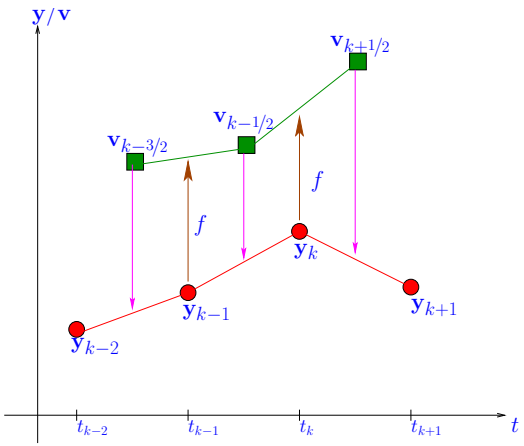
Initialisation (→ Rem. 8.4.8) is built in already in the formulation.

△

Remark 8.4.11 (Störmer-Verlet method as polygonal line method).

Perspective: Störmer-Verlet method as polygonal line method (see Ren. 8.4.10)

$$\begin{aligned} \mathbf{v}_{k+\frac{1}{2}} &= \mathbf{v}_{k-\frac{1}{2}} + h\mathbf{f}(\mathbf{y}_k) \,, \\ \mathbf{y}_{k+1} &= \mathbf{y}_k + h\mathbf{v}_{k+\frac{1}{2}} \,. \end{aligned}$$



△

"Why so many different numerical methods for ODEs?"

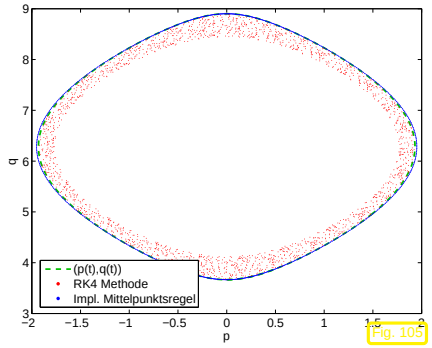
Answer: each numerical integrator has its specific properties
➥ best (not) suited for certain classes of initial value problems

8.4.3 Examples

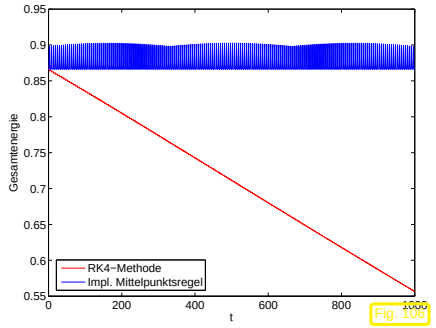
Example 8.4.12 (Energy conservation). ↔ Bsp. 8.4.6

Pendulum IVP (8.4.1) on $[0, 1000]$, $p(0) = 0$, $q(0) = {}^{7}\pi/6$.

Compare classical Runge-Kutta method (8.6.6) (order 4) with implicit midpoint rule 8.4.2), constant time-step $h = \frac{1}{2}$:



Trajectories of "'exact" /discrete evolutions



Energy conservation of discrete evolutions

➢ No drift in enrgy in case of midpoint rule

◇

Fascinating observation

?

Some $(*)$ numerical time-integrators:
approximative long-time energy conservation (no energy drift)

$(*)$ Implicit midpoint rule (8.4.2) → Ex. 8.4.12, 8.4.6,
Störmer-Verlet mehtod (**??**) → Ex. 8.4.9

Example 8.4.13 (Spring-pendulum).

Friction-free spring-pendulum: Hamilton function (energy) $H(\mathbf{p}, \mathbf{q}) = \frac{1}{2}\|\mathbf{p}\|^2 + \frac{1}{2}(\|\mathbf{q}\| - 1)^2 + q_2$
($\mathbf{q} \hat{=}$ position, $\mathbf{p} \hat{=}$ momentum)

$$\dot{\mathbf{p}} = -(\|\mathbf{q}\| - 1)\frac{\mathbf{q}}{\|\mathbf{q}\|} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \,, \quad \dot{\mathbf{q}} = \mathbf{p} \,. \tag{8.4.8}$$

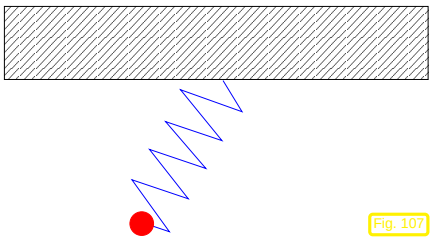


Fig. 107

Trajectories for long-time evolution ▷
(chaotical mechanical system)

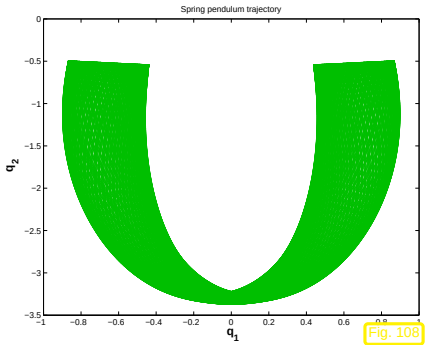


Fig. 108

ESV:
- Störmer-Verlet method (8.4.5) (order 2),
- explicit trapezoidal rule (8.6.3) (order 2).

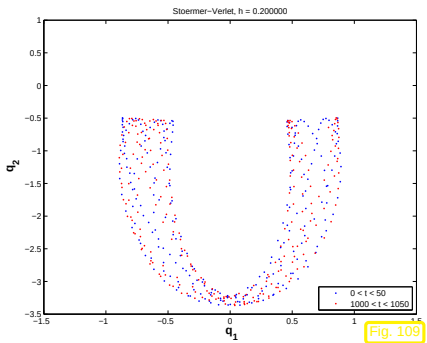


Fig. 109

Störmer-Verlet

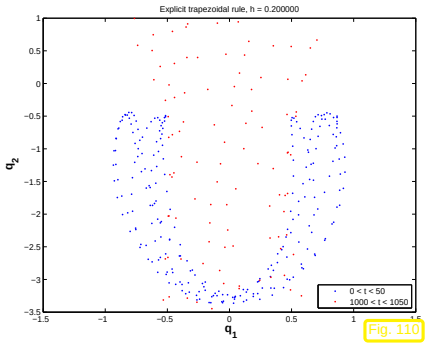


Fig. 110

Explicit trapezoidal rule

Störmer-Verlet: positions in “allowed domain” even for long times

Explicit trapezoidal rule: trajectories leave the “allowed domain” at long times (energy drift !)

◇

Example 8.4.14 (Molecular dynamics). → [13, Sect. 1.2]

- Space of states for $n \in \mathbb{N}$ atoms in $d \in \mathbb{N}$ dimensions: $D = \mathbb{R}^{2dn}$
(positions $\mathbf{q} = [\mathbf{q}^1; \dots; \mathbf{q}^n]^T \in \mathbb{R}^{dn}$, Momenta $\mathbf{p} = [\mathbf{p}^1, \dots, \mathbf{p}^n]^T \in \mathbb{R}^{dn}$)

Energy (Hamilton-function):

$$H(\mathbf{p}, \mathbf{q}) = \tfrac{1}{2} \|\mathbf{p}\|_2^2 + V(\mathbf{q}) \ .$$

Lenard-Jones-potential: ▷

$$V(\mathbf{q}) = \sum_{j=1}^{n} \sum_{i \neq j} \mathcal{V}(\left\|\mathbf{q}^i - \mathbf{q}^j\right\|_2) \ ,$$
$$\mathcal{V}(\xi) = \xi^{-12} - \xi^{-6} \ . \qquad (8.4.9)$$

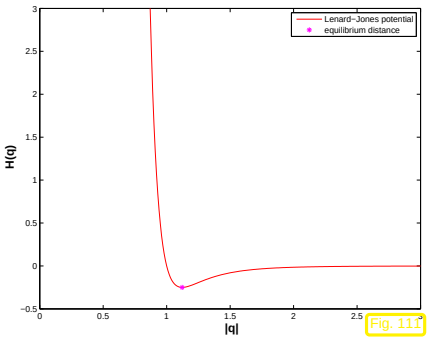


Fig. 111

➥ Hamiltonian ODE

$$\dot{\mathbf{p}}^j = -\sum_{i \neq j} \mathcal{V}'(\left\|\mathbf{q}^j - \mathbf{q}^i\right\|_2) \frac{\mathbf{q}^j - \mathbf{q}^i}{\left\|\mathbf{q}^j - \mathbf{q}^i\right\|_2} \quad , \quad \dot{\mathbf{q}}^j = \mathbf{p}^j \ , \ j = 1, \dots, n \ .$$

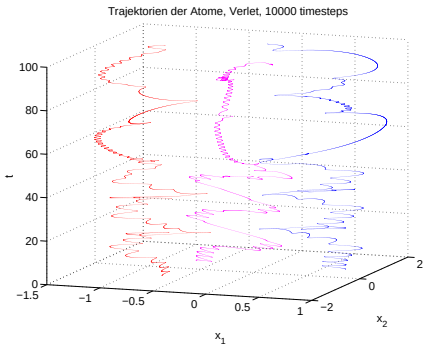
Störmer-Verlet method (8.4.5):

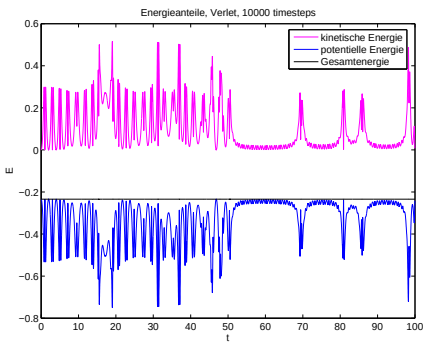
$$\mathbf{q}_h(t + \tfrac{1}{2}h) = \mathbf{q}_h(t) + \tfrac{h}{2}\mathbf{p}_h(t) \ ,$$

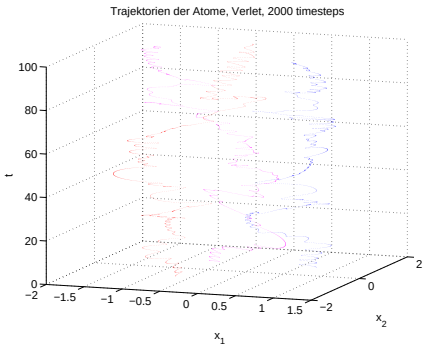
$$\mathbf{p}_h^j(t+h) = \mathbf{p}_h^j(t) - h \sum_{i \neq j} \mathcal{V}'(\left\|\mathbf{q}_h^j(t+\tfrac{1}{2}h) - \mathbf{q}_h^i(t+\tfrac{1}{2}h)\right\|_2) \frac{\mathbf{q}_h^j(t+\frac{1}{2}h) - \mathbf{q}_h^i(t+\frac{1}{2}h)}{\left\|\mathbf{q}_h^j(t+\frac{1}{2}h) - \mathbf{q}_h^i(t+\frac{1}{2}h)\right\|_2} \ ,$$

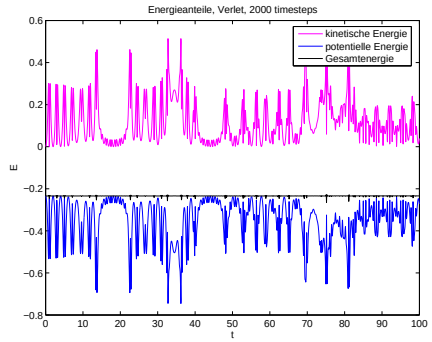
$$\mathbf{q}_h(t+h) = \mathbf{q}_h(t+\tfrac{1}{2}h) + \tfrac{h}{2}\mathbf{p}_h(t+h) \ .$$

Simulation with $d = 2$, $n = 3$, $\mathbf{q}^1(0) = \frac{1}{2}\sqrt{2}\binom{-1}{-1}$, $\mathbf{q}^2(0) = \frac{1}{2}\sqrt{2}\binom{1}{1}$, $\mathbf{q}^3(0) = \frac{1}{2}\sqrt{2}\binom{-1}{1}$, $\mathbf{p}(0) = 0$, end-time $T = 100$









Observations:

- very different trajektories for long-times with different tim=steps h.
- Qualitatively correct trajektories in any case.

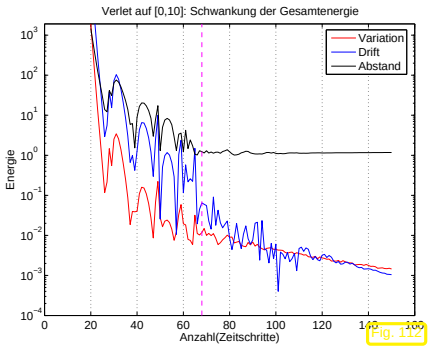


Fig. 112

$T = 10$, $d = 2$, $n = 3$, $\mathbf{q}^1(0) = \frac{1}{2}\sqrt{2}\binom{-1}{-1}$, $\mathbf{q}^2(0) = \frac{1}{2}\sqrt{2}\binom{1}{1}$, $\mathbf{q}^3(0) = \frac{1}{2}\sqrt{2}\binom{-1}{1}$, $\mathbf{p}(0) = 0$.

Variation $= \sum_{i=1}^{N-1} |E_{\text{tot}}((i+1)h) - E_{\text{tot}}(ih)|$,

Drift $= |E_{\text{tot}}(T) - E_{\text{tot}}(0)|$,

Abstand $= \max\{\left\|\mathbf{q}_h^j(T)\right\|_2, j = 1,2,3\}$.

◇

Example 8.4.15 (Many particles molecular dynamics). → [**?**, Sect. 4.5.1]

2D conservative system of many particles with Lennard-Jones-potential → Bsp. 8.4.14

initial positions ▷

(initial momenta $= 0 \leftrightarrow$ 0K)

Obseravtions for explicite trapezoiudal rule (8.6.3), Störmer-Verlet (8.4.5)

- Approximation of energy $H(\mathbf{p}, \mathbf{q})$
- middle kinetic energy ("temperature")

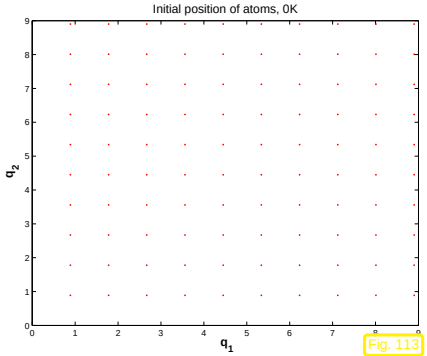


Fig. 113

Animation ▷

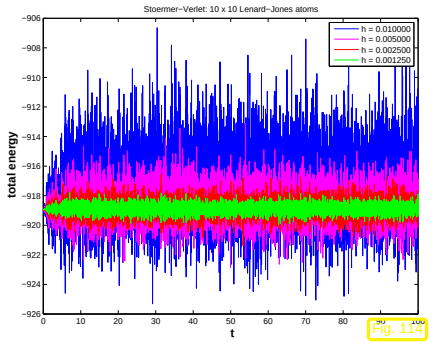


Fig. 114

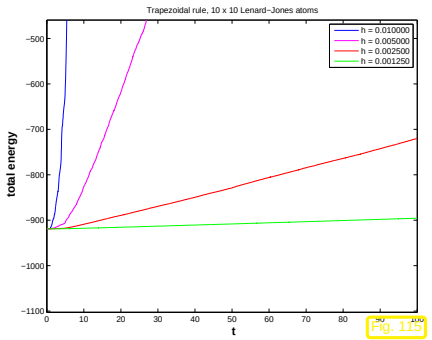


Fig. 115

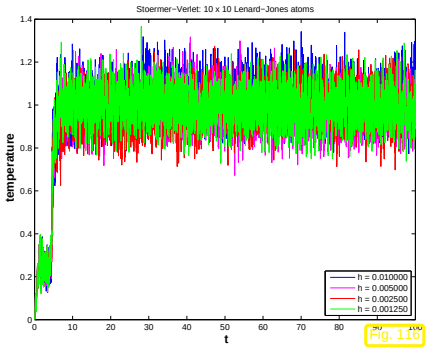


Fig. 116

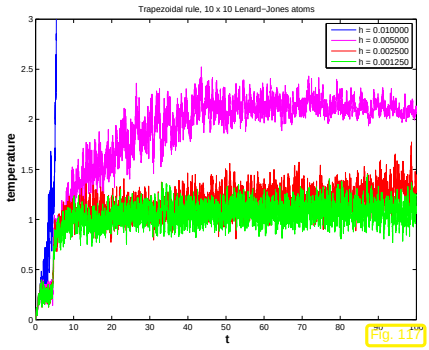


Fig. 117

Symplectic time-integrator (Verlet): qualitatively correct behaviour of the temperature

◇

8.5 Splitting methods [27, Sect. 2.5]

Autonomous IVP with right hand side of the form:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) + \mathbf{g}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0 \; , \tag{8.5.1}$$

with $\mathbf{f} : D \subset \mathbb{R}^d \mapsto \mathbb{R}^d$, $\mathbf{g} : D \subset \mathbb{R}^d \mapsto \mathbb{R}^d$ "suff. smooth", locally Lipschitz continuous (→ Def. 8.1.2)

(Continuous) Evolution:
$$\begin{array}{lcll} \mathbf{\Phi}_f^t & \leftrightarrow & \text{eq.} & \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \; , \\ \mathbf{\Phi}_g^t & \leftrightarrow & \text{eq.} & \dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}) \; . \end{array}$$

Assume: $\mathbf{\Phi}_f^t$, $\mathbf{\Phi}_g^t$ (analytically) known

Idea: Construct Single-step method with discrete evolution

Lie-Trotter-Splitting: $\mathbf{\Psi}^h = \mathbf{\Phi}_g^h \circ \mathbf{\Phi}_f^h \; ,$ (8.5.2)

Strang-Splitting: $\mathbf{\Psi}^h = \mathbf{\Phi}_f^{h/2} \circ \mathbf{\Phi}_g^h \circ \mathbf{\Phi}_f^{h/2} \; .$ (8.5.3)

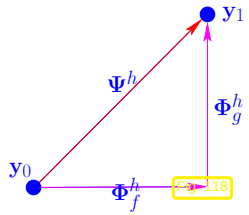
(8.5.2) ↔ 

Fig. 118

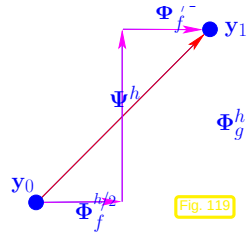
(8.5.3) ↔ 

Fig. 119

Example 8.5.1 (Convergence of simple splitting methods).

$$\dot{y} = \underbrace{\lambda y(1-y)}_{=:f(y)} + \underbrace{\sqrt{1-y^2}}_{=:g(y)} \quad , \quad y(0) = 0 \; .$$

- $\mathbf{\Phi}_f^t y = \dfrac{1}{1 + (y^{-1} - 1)e^{-\lambda t}} \; , \; t > 0, y \in]0,1]$ (logistic differential equation (8.1.3))
- $\mathbf{\Phi}_g^t y = \begin{cases} \sin(t + \arcsin(y)) & \text{, if } t + \arcsin(y) < \frac{\pi}{2} \; , \\ 1 & \text{, else,} \end{cases} \quad t > 0, y \in [0,1] \; .$

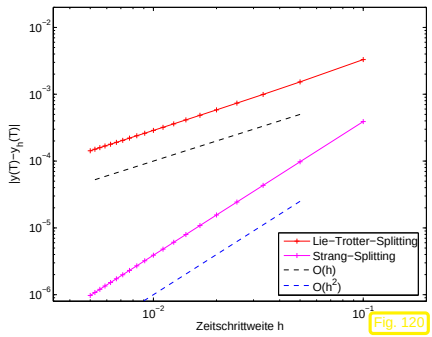


Fig. 120

Numerical experiment:

$T = 1$, $\lambda = 1$, compare splitting method (constant time-step) with very precise numerical solution

◁ Error at end-time $T = 1$

◇

Example 8.5.2 (Splitting methods for mechanical systems).

Newton eq. of motion $\ddot{\mathbf{r}} = a(\mathbf{r}) \overset{(8.1.13)}{\Longleftrightarrow} \dot{\mathbf{y}} := \begin{pmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ a(\mathbf{r}) \end{pmatrix} =: \mathbf{F}(\mathbf{y}) \; .$

Splitting: $F(\mathbf{y}) = \underbrace{\begin{pmatrix} 0 \\ a(\mathbf{r}) \end{pmatrix}}_{=:\mathbf{f}(\mathbf{y})} + \underbrace{\begin{pmatrix} \mathbf{v} \\ 0 \end{pmatrix}}_{=:\mathbf{g}(\mathbf{y})} \; .$

- $\mathbf{\Phi}_f^t \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 + ta(\mathbf{r}_0) \end{pmatrix} \quad , \quad \mathbf{\Phi}_g^t \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{r}_0 + t\mathbf{v}_0 \\ \mathbf{v}_0 \end{pmatrix} \; .$

Lie-Trotter-Splitting (8.5.2): ➢ Symplectic Euler method

$$\Psi^h\begin{pmatrix}\mathbf{r}\\ \mathbf{v}\end{pmatrix} = \left(\Phi_g^h \circ \Phi_f^h\right)\begin{pmatrix}\mathbf{r}\\ \mathbf{v}\end{pmatrix} = \begin{pmatrix}\mathbf{r}+h(\mathbf{v}+ha(\mathbf{r}))\\ \mathbf{v}+ha(\mathbf{r})\end{pmatrix} . \tag{8.5.4}$$

Strang-Splitting (8.5.3):

$$\Psi^h\begin{pmatrix}\mathbf{r}\\ \mathbf{v}\end{pmatrix} = \left(\Phi_g^{h/2} \circ \Phi_f^h \circ \Phi_g^{h/2}\right)\begin{pmatrix}\mathbf{r}\\ \mathbf{v}\end{pmatrix} = \begin{pmatrix}\mathbf{r}+h\mathbf{v}+\frac{1}{2}h^2a(\mathbf{r}+\frac{1}{2}h\mathbf{v})\\ \mathbf{v}+ha(\mathbf{r}+\frac{1}{2}h\mathbf{v})\end{pmatrix} . \tag{8.5.5}$$

= single-step formulation of Störmer-Verlet methods (8.4.5), see Rem. 8.4.10 !

$$(8.5.5) \quad \longleftrightarrow \quad \begin{aligned} \mathbf{r}_{k+\frac{1}{2}} &= \mathbf{r}_k + \tfrac{1}{2}h\mathbf{v}_k , \\ \mathbf{v}_{k+1} &= \mathbf{v}_k + ha(\mathbf{r}_{k+\frac{1}{2}}) , \\ \mathbf{r}_{k+1} &= \mathbf{r}_{k+\frac{1}{2}} + \tfrac{1}{2}h\mathbf{v}_{k+1} . \end{aligned} \tag{8.5.6}$$

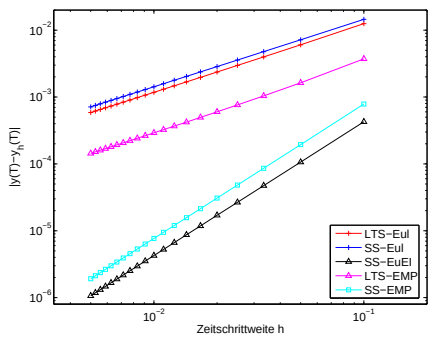
◇

Idea: Replace

exact evolution	$\longrightarrow$	discrete evolution
Φ_g^h, Φ_f^h	$\longrightarrow$	Ψ_g^h, Ψ_f^h

Example 8.5.3 (Inexact splitting method). Cont. Ex. 8.5.1

IVP of Eq. 8.5.1, inexact splitting method when we rely on (several) inexact underlying methods:



LTS-Eul	explicit Euler method (**??**) $\longrightarrow$ $\Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Lie-Trotter-Splitting (8.5.2)
SS-Eul	explicit Euler method (**??**) $\longrightarrow$ $\Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Strang-Splitting (8.5.3)
SS-EuEl	Strang-Splitting (8.5.3): explicit Euler method (**??**) $\circ$ exact evolution Φ_g^h $\circ$ implicite Euler method (**??**)
LTS-EMP	explicit midpoint rule (8.6.4) $\longrightarrow$ $\Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Lie-Trotter-Splitting (8.5.2)
SS-EMP	explicit midpoint rule (8.6.4) $\longrightarrow$ $\Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Strang-Splitting (8.5.3)

☞ Order of splitting methods is determined by the quality of Φ_f^h, Φ_g^h.

◇

Remark 8.5.4. Note that the converegence order of a reversible method (i.e., if we exchange $h \leftrightarrow -h$ and $y_k \leftrightarrow y_{k+1}$ we get the same method) is always even.

8.6 Runge-Kutta methods

So far we only know first order methods, the explicit and implicit Euler method (8.2.1) and (8.2.4), respectively.

Now we will buld a class of methods that achieve orders > 1. The starting point is a simple *integral equation* satisfied by solutions of initial value problems:

IVP: $\begin{aligned}\dot{\mathbf{y}}(t) &= \mathbf{f}(t,\mathbf{y}(t)) , \\ \mathbf{y}(t_0) &= \mathbf{y}_0\end{aligned} \quad \Rightarrow \quad \mathbf{y}(t_1) = \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}(\tau,\mathbf{y}(t_0+\tau))\,\mathrm{d}\tau$

Idea: approximate integral by means of s-point quadrature formula ($\longrightarrow$ Sect. 7.1, defined on reference interval $[0,1]$) with nodes $c_1,\dots,c_s$, weights $b_1,\dots,b_s$.

$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{f}(t_0+c_ih, \boxed{\mathbf{y}(t_0+c_ih)}) , \quad h := t_1 - t_0 . \tag{8.6.1}$$

Obtain these values by bootstrapping

bootstrapping = use the same idea in a simpler version to get $\mathbf{y}(t_0+c_ih)$, noting that these values can be replaced by other approximations obtained by methods already constructed (This approach will be elucidated in the next example).

What error can we afford in the approximation of $\mathbf{y}(t_0+c_ih)$ (under the assumption that $\mathbf{f}$ is Lipschitz continuous)?

Goal: one-step error $\mathbf{y}(t_1) - \mathbf{y}_1 = O(h^{p+1})$

This goal can already be achieved, if only

$\mathbf{y}(t_0+c_ih)$ is approximated up to an error $O(h^p)$,

because in (8.6.1) a factor of size h multiplies $\mathbf{f}(t_0+c_i,\mathbf{y}(t_0+c_ih))$.

This is accmomplished by a less accurate discrete evolution than the one we are bidding for. Thus, we can construct discrete evolutions of higher and higher order, successively.

Example 8.6.1 (Construction of simple Runge-Kutta methods).

Quadrature formula = trapezoidal rule (8.6.2):

$$Q(f) = \tfrac{1}{2}(f(0)+f(1)) \quad \leftrightarrow \quad s=2: \quad c_1 = 0, c_2 = 1\,, \quad b_1 = b_2 = \frac{1}{2}\,, \tag{8.6.2}$$

and $\mathbf{y}(T)$ approximated by explicit Euler step (8.2.1)

$$\mathbf{k}_1 = \mathbf{f}(t_0,\mathbf{y}_0)\,, \quad \mathbf{k}_2 = \mathbf{f}(t_0+h, \mathbf{y}_0 + h\mathbf{k}_1)\,, \quad \mathbf{y}_1 = \mathbf{y}_0 + \tfrac{h}{2}(\mathbf{k}_1+\mathbf{k}_2)\,. \tag{8.6.3}$$

(8.6.3) = explicit trapezoidal rule (for numerical integration of ODEs)

Quadrature formula $\longrightarrow$ simplest Gauss quadrature formula = midpoint rule & $\mathbf{y}(\frac{1}{2}(t_1+t_0))$ approximated by explicit Euler step (8.2.1)

$$\mathbf{k}_1 = \mathbf{f}(t_0,\mathbf{y}_0)\,, \quad \mathbf{k}_2 = \mathbf{f}(t_0+\tfrac{h}{2}, \mathbf{y}_0 + \tfrac{h}{2}\mathbf{k}_1)\,, \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}_2\,. \tag{8.6.4}$$

(8.6.4) = explicit midpoint rule (for numerical integration of ODEs)

◇

Example 8.6.2 (Convergence of simple Runge-Kutta methods).

- IVP: $\dot{y} = 10y(1-y)$ (logistic ODE (8.1.3)), $y(0) = 0.01$, $T = 1$,
- Explicit single step methods, uniform timestep h.

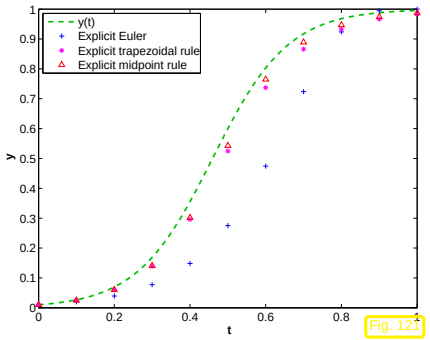


Fig. 121

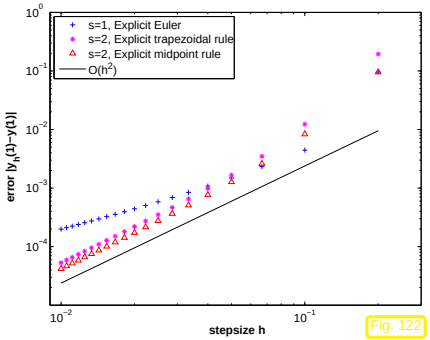


Fig. 122

$y_h(j/10)$, $j = 1,\dots,10$ for explicit RK-methods

Errors at final time $y_h(1) - y(1)$

Observation: obvious algebraic convergence with integer rates/orders

explicit trapezoidal rule (8.6.3) order 2
explicit midpoint rule (8.6.4) order 2

◇

The formulas that we have obtained follow a general pattern:

Definition 8.6.1 (Explicit Runge-Kutta method)**.**
For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i,j = 1,\dots,s$, $s \in \mathbb{N}$, an s-stage explicit Runge-Kutta single step method (RK-SSM) for the IVP (8.1.11) *is defined by*

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j)\,, \quad i = 1,\dots,s \quad , \quad \mathbf{y}_1 := \mathbf{y}_0 + h\sum_{i=1}^{s} b_i\mathbf{k}_i\,.$$

The $\mathbf{k}_i \in \mathbb{R}^d$ are called increments.

The implementation of an s-stage explicit Runge-Kutta single step method according to Def. 8.6.1 is straightforward: The increments $\mathbf{k}_i \in \mathbb{R}^d$ are computed successively, starting from $\mathbf{k}_1 = \mathbf{f}(t_0 + c_1 h, \mathbf{y}_0)$.

► Only s $\mathbf{f}$-evaluations and AXPY operations are required.

Shorthand notation for (explicit) Runge-Kutta methods

Butcher scheme ▷

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|ccc} c_1 & 0 & \cdots & 0 \\ c_2 & a_{21} & \ddots & \vdots \\ \vdots & \vdots & & \ddots \;\; \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} \;\; 0 \\ \hline & b_1 & \cdots & b_s \end{array}\,. \tag{8.6.5}$$

(Note: $\mathfrak{A}$ is strictly lower triangular $s \times s$-matrix)

Note that in Def. 8.6.1 the coefficients b_i can be regarded as weights of a quadrature formula on $[0,1]$: apply explicit Runge-Kutta single step method to "ODE" $\dot{y} = f(t)$.

► Necessarily $\displaystyle\sum_{i=1}^{s} b_i = 1$

Example 8.6.3 (Butcher scheme for some explicit RK-SSM).

- Explicit Euler method (8.2.1): $\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$ ➢ order = 1

- explicit trapezoidal rule (8.6.3): $\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$ ➢ order = 2

- explicit midpoint rule (8.6.4): $\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$ ➢ order = 2

- Classical 4th-order RK-SSM: $\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$ ➢ order = 4

- Kutta's 3/8-rule: $\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array}$ ➢ order = 4

◇

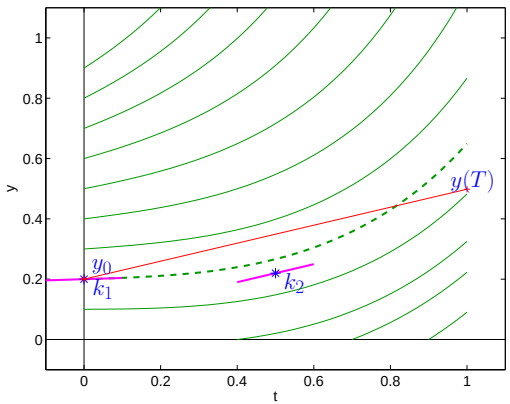
Example 8.6.4 (explicit Runge-Kutta method for Ricatti equation).
Initial value problem: $\dot{y} = t^2 + y^2$, $y(0) = 0.2$.

Geometrical interpretation of explicit RK-methods as polygonal lines

explicit midpoint rule:

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

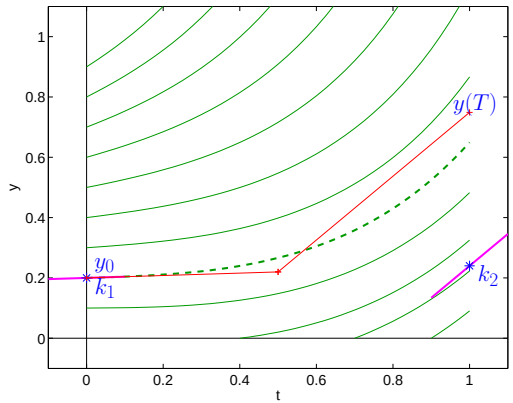
green solution
magenta local slops $\mathbf{k}_i$
$*$ points f-evaluation
red: polygonal line



explicit trapezoidal rule

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

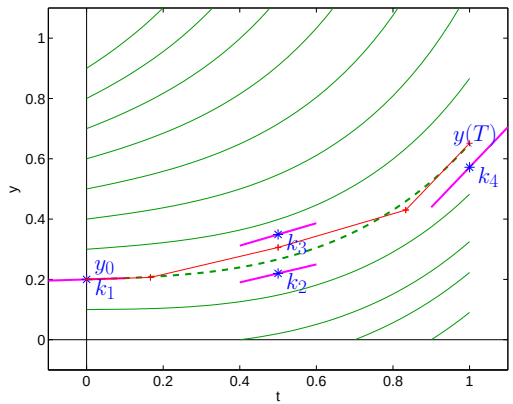
green: solution
magenta: local slops $\mathbf{k}_i$
$*$: points f-evaluation
red: polygonal line



classical Runge-Kutta method (RK4)

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} \qquad (8.6.6)$$

green solution
magenta local slopes $\mathbf{k}_i$
$*$ point f-evaluation
red: polygonal line



Kuttas 3/8-rule

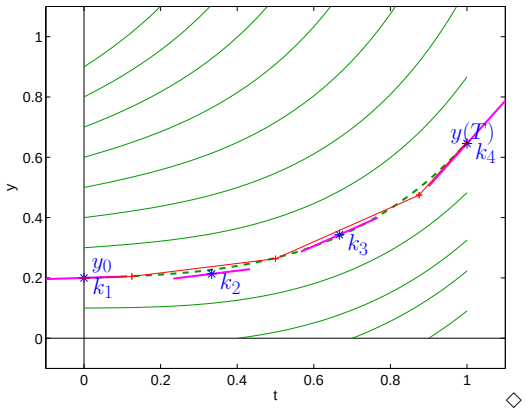
$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array} \qquad (8.6.7)$$

green solution

magenta local slopes $\mathbf{k}_i$

$*$ point f-evaluation

red: polygonal line



◇

Remark 8.6.5 ("Butcher barriers" for explicit RK-SSM).

order p	1	2	3	4	5	6	7	8	≥ 9
minimal no. s of stages	1	2	3	4	6	7	9	11	$\geq p+3$

No general formula available so far

Known: order p $<$ number s of stages of RK-SSM

△

Remark 8.6.6 (Explicit ODE integrator a la MATLAB).

Syntax:

```
t,y = ode45(odefun,tspan,y0);
```

`odefun` : Handle to a function of type `(t,y)` $\leftrightarrow$ r.h.s. $\mathbf{f}(t,\mathbf{y})$

`tspan` : vector $(t_0,T)^T$, initial and final time for numerical integration

`y0` : (vector) passing initial state $\mathbf{y}_0 \in \mathbb{R}^d$

Return values:

`t` : temporal mesh $\{t_0 < t_1 < t_2 < \cdots < t_{N-1} = t_N = T\}$

`y` : sequence $(\mathbf{y}_k)_{k=0}^N$ (column vectors)

△

Example 8.6.7 (Numerical integration of logistic ODE).

usage of ode45

```
from ode45 import ode45
fn = lambda t, y:  5.*y*(1.-y)
t,y = ode45(fn,(0, 1.5),[1.5])
```

integrator: `ode45()`:

- Handle passing r.h.s.
- initial and final time
- initial state $\mathbf{y}_0$

◇

8.7 Stepsize control

Example 8.7.1 (Oregonator reaction).

Special case of oscillating Zhabotinski-Belousov reaction [21]:

$$\begin{array}{rcl} \mathrm{BrO}_3^- + \mathrm{Br}^- & \mapsto & \mathrm{HBrO}_2 \\ \mathrm{HBrO}_2 + \mathrm{Br}^- & \mapsto & \mathrm{Org} \\ \mathrm{BrO}_3^- + \mathrm{HBrO}_2 & \mapsto & 2\,\mathrm{HBrO}_2 + \mathrm{Ce(IV)} \\ 2\,\mathrm{HBrO}_2 & \mapsto & \mathrm{Org} \\ \mathrm{Ce(IV)} & \mapsto & \mathrm{Br}^- \end{array} \qquad (8.7.1)$$

$$\begin{array}{lrcl} y_1 := c(\mathrm{BrO}_3^-): & \dot{y}_1 & = & -k_1y_1y_2 - k_3y_1y_3\,, \\ y_2 := c(\mathrm{Br}^-): & \dot{y}_2 & = & -k_1y_1y_2 - k_2y_2y_3 + k_5y_5\,, \\ y_3 := c(\mathrm{HBrO}_2): & \dot{y}_3 & = & k_1y_1y_2 - k_2y_2y_3 + k_3y_1y_3 - 2k_4y_3^2\,, \\ y_4 := c(\mathrm{Org}): & \dot{y}_4 & = & k_2y_2y_3 + k_4y_3^2\,, \\ y_5 := c(\mathrm{Ce(IV)}): & \dot{y}_5 & = & k_3y_1y_3 - k_5y_5\,, \end{array} \qquad (8.7.2)$$

with (non-dimensionalized) reaction constants:

$$k_1 = 1.34\,,\quad k_2 = 1.6\cdot 10^9\,,\quad k_3 = 8.0\cdot 10^3\,,\quad k_4 = 4.0\cdot 10^7\,,\quad k_5 = 1.0\,.$$

periodic chemical reaction ➡ Video 1, Video 2

simulation with inital state $y_1(0) = 0.06$, $y_2(0) = 0.33\cdot 10^{-6}$, $y_3(0) = 0.501\cdot 10^{-10}$, $y_4(0) = 0.03$, $y_5(0) = 0.24\cdot 10^{-7}$:

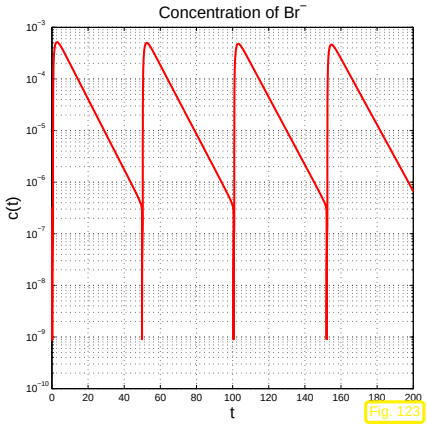


Fig. 123

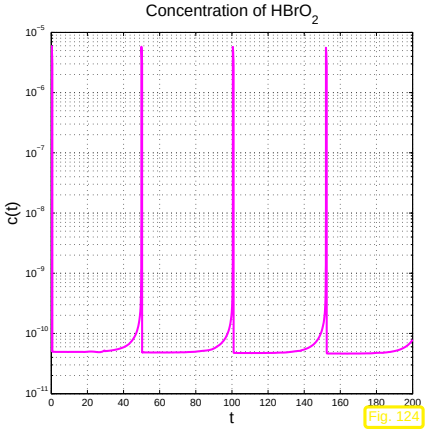


Fig. 124

We observe a strongly non-uniform behavior of the solution in time.

This is very common with evolutions arising from practical models (circuit models, chemical reaction models, mechanical systems)

◇

Example 8.7.2 (Blow-up).

Scalar autonomous IVP:

$$\dot{y} = y^2 \,, \quad y(0) = y_0 > 0 \,.$$

$$\blacktriangleright \quad y(t) = \frac{y_0}{1 - y_0 t} \,, \quad t < 1/y_0 \,.$$

Solution exists only for finite time and then suffers a Blow-up, that is, $\lim_{t \to 1/y_0} y(t) = \infty$: $J(y_0) =]-\infty, 1/y_0]$!

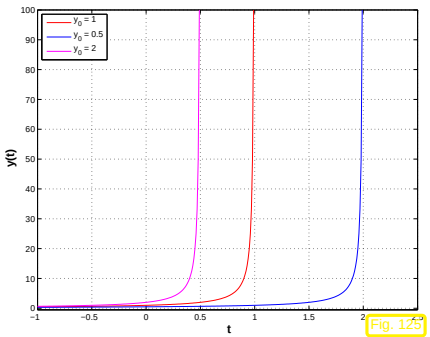


Fig. 125

How to choose temporal mesh $\{t_0 < t_1 < \cdots < t_{N-1} < t_N\}$ for single step method in case $J(y_0)$ is not known, even worse, if it is not clear a priori that a blow up will happen?

Just imagine: what will result from equidistant explicit Euler integration (8.2.1) applied to the above IVP?

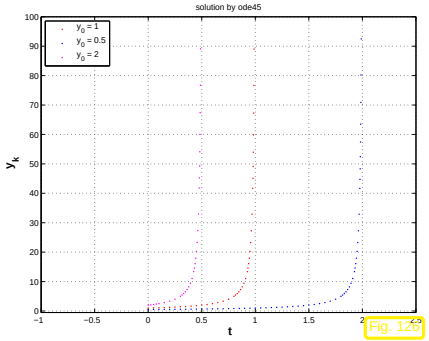


Fig. 126

```
odefun = lambda t,y: y**2.0
print 'y0 = ',0.5
t2, y2 = ode45(odefun, (0,2), 0.5,
  stats=True)
print '\n','y0 = ',1
t1, y1 = ode45(odefun, (0,2), 1,
  stats=True)
print '\n','y0 = ',2
t3, y3 = ode45(odefun, (0,2), 2,
  stats=True)
```

warning messages:

```
y0 = 0.5
error: OdePkg:InvalidArgument
Solving has not been successful. The iterative integration loop exited at time t =

Number of successful steps: 151
Number of failed attempts:  74
Number of function calls:   1344

y0 = 1
error: OdePkg:InvalidArgument
Solving has not been successful. The iterative integration loop exited at time t =

Number of successful steps: 146
Number of failed attempts:  74
Number of function calls:   1314

y0 = 2
error: OdePkg:InvalidArgument
Solving has not been successful. The iterative integration loop exited at time t =

Number of successful steps: 144
Number of failed attempts:  72
Number of function calls:   1290
```

We observe: `ode45` manages to reduce stepsize more and more as it approaches the singularity of the solution!

◇

Key issue (discussed for autonomous ODEs below):

Choice of *good temporal mesh* $\{0 = t_0 < t_1 < \cdots < t_{N-1} < t_N\}$
for a given single step method applied to an IVP

What does “good” mean ?

be efficient — be accurate

Objective: N as small as possible & $\max_{k=1,\dots,N} \|\mathbf{y}(t_k) - \mathbf{y}_k\| < \mathtt{TOL}$ or $\|\mathbf{y}(T) - \mathbf{y}_N\| < \mathtt{TOL}$, $\mathtt{TOL}$ = tolerance

Policy: Try to curb/balance one-step error by

- adjusting *current* stepsize h_k,
- predicting suitable *next* timestep h_{k+1}

} local-in-time stepsize control

Tool: Local-in-time one-step error estimator (*a posteriori*, based on $\mathbf{y}_k, h_{k-1}$)

Why local-in-time timestep control (based on estimating the one-step error)**?**

Consideration: If a small time-local error in a single timestep leads to large error $\|\mathbf{y}_k - \mathbf{y}(t_k)\|$ at later times, then local-in-time timestep control is powerless about it and will not even notice!!

Nevertheless, local-in-time timestep control is used almost exclusively,

☞ because we do not want to discard past timesteps, which could amount to tremendous waste of computational resources,

☞ because it is inexpensive and it works for many practical problems,

☞ because there is not reliable method that can deliver guaranteed accuracy for general IVP.

"Recycle" heuristics already employed for adaptive quadrature, see Sect. 7.3:

Idea: *Estimation of one-step error*, *cf.* Sect. 7.3

Compare two discrete evolutions $\mathbf{\Psi}^h$, $\widetilde{\mathbf{\Psi}}^h$ of different order for *current timestep* h:

If Order$(\widetilde{\mathbf{\Psi}}) >$ Order$(\mathbf{\Psi})$

$$\Rightarrow \quad \underbrace{\mathbf{\Phi}^h\mathbf{y}(t_k) - \mathbf{\Psi}^h\mathbf{y}(t_k)}_{\text{one-step error}} \approx \mathtt{EST}_k := \widetilde{\mathbf{\Psi}}^h\mathbf{y}(t_k) - \mathbf{\Psi}^h\mathbf{y}(t_k) \,. \tag{8.7.3}$$

Heuristics for concrete h

absolute tolerance

Compare $\mathtt{EST}_k \leftrightarrow \mathtt{ATOL}$, $\mathtt{EST}_k \leftrightarrow \mathtt{RTOL}\,\|\mathbf{y}_k\|$ ➢ Reject/accept current step (8.7.4)

relative tolerance

Simple algorithm:

$\mathtt{EST}_k < \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\}$: Carry out next timestep (stepsize h)
Use larger stepsize (e.g., αh with some $\alpha > 1$) for following step $(*)$

$\mathtt{EST}_k > \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\}$: *Repeat* current step with smaller stepsize $< h$, e.g., $\frac{1}{2}h$

Rationale for $(*)$: if the current stepsize guarantees sufficiently small one-step error, then it might be possible to obtain a still acceptable one-step error with a larger timestep, which would enhance efficiency (fewer timesteps for total numerical integration). This should be tried, since timestep control will usually provide a safeguard against undue loss of accuracy.

Code 8.7.3: simple local stepsize control for single step methods

```
def odeintadapt(Psilow, Psihigh, T, y0, h0, reltol, abstol, hmin):
        t=[0]; y=[y0]; h=h0 #
        while t[-1] < T and h > hmin:
                yh = Psihigh(h,y0)
                yH = Psilow(h,y0)
                est = norm(yH-yh)
                if est < max(reltol*norm(y0), abstol):
                        y0 = yh; y.append(y0);
                          t.append(t[-1]+min(T-t[-1],h))
                        h = 1.1*h
                else:
                        h = h/2.0
        return (array(t),array(y))

def odeintadapt_ext(Psilow, Psihigh, T, y0, h0, reltol, abstol, hmin):
        """extended version of odeintadapt. also returns vector of
          rejected points and vector of estimated errors"""
        t=[0]; y=[y0]; h=h0; rej = []; ee = [0]
        while t[-1] < T and h > hmin:
                yh = Psihigh(h,y0)
                yH = Psilow(h,y0)
                est = norm(yH-yh)
                if est < max(reltol*norm(y0), abstol):
# print 'accept step, h = y0 = yh; y.append(y0); t.append(t[-1]+min(T-t[-1],h))
                        h = 1.1*h; ee.append(est)
                else:
# print 'reject step, h = rej = hstack([rej,t]); h = h/2.0
        return (array(t),array(y),rej,ee)
```

Comments on Code 8.7.2:

- Input arguments:

- `Psilow`, `Psihigh`: function handles to discrete evolution operators for autonomous ODE of different order, type `@(y,h)`, expecting a state (column) vector as first argument, and a stepsize as second,
- `T`: final time $T > 0$,
- `y0`: initial state $\mathbf{y}_0$,
- `h0`: stepsize h_0 for the first timestep
- `reltol`, `abstol`: relative and absolute tolerances, see (8.7.4),
- `hmin`: minimal stepsize, timestepping terminates when stepsize control $h_k < h_{\min}$, which is relevant for detecting blow-ups or collapse of the solution.

- line **??**: check whether final time is reached or timestepping has ground to a halt ($h_k < h_{\min}$).
- line **??**, **??**: advance state by low and high order integrator.
- line **??**: compute norm of estimated error, see (**??**).
- line **??**: make comparison (8.7.4) to decide whether to accept or reject local step.
- line **??**, **??**: step accepted, update state and current time and suggest 1.1 times the current stepsize for next step.
- line **??** step rejected, try again with half the stepsize.
- Return values:


```
        """Simple adaptive timestepping strategy of
          Code~\ref{mc:odeintadapt}
        based on explicit Euler \eqref{eq:eeul} and explicit
  trapezoidal
        rule \eqref{eq:exTrap}
        """

        # autonomous ODE ẏ = cos(ay) and its general solution
        f = lambda y: (cos(a*y)**2)
        sol = lambda t: arctan(a*(t-1))/a
        # Initial state y0
        y0 = sol(0)

        # Discrete evolution operators, see Def. 8.2.1
        # Explicit Euler (8.2.1)
        Psilow = lambda h,y: y + h*f(y)
        # Explicit trapzoidal rule (8.6.3)
        Psihigh = lambda h,y: y + 0.5*h*( f(y) + f(y+h*f(y)) )

        # Heuristic choice of initial timestep and h_min
        h0 = T/(100.0*(norm(f(y0))+0.1)); hmin = h0/10000.0;
        # Main adaptive timestepping loop, see Code 8.7.2
        t,y,rej,ee =
```


- `t`: temporal mesh $t_0 < t_1 < t_2 < \ldots < t_N < T$, where $t_N < T$ indicated premature termination (collapse, blow-up),
- `y`: sequence $(\mathbf{y}_k)_{k=0}^N$.

! By the heuristic considerations, see (8.7.3) it seems that EST_k measures the one-step error for the low-order method $\boldsymbol{\Psi}$ and that we should use $\mathbf{y}_{k+1} = \boldsymbol{\Psi}^{h_k}\boldsymbol{y}_k$, if the timestep is accepted.

However, it would be foolish not to use the better value $\mathbf{y}_{k+1} = \widetilde{\boldsymbol{\Psi}}^{h_k}\boldsymbol{y}_k$, since it is available for free. This is what is done in every implementation of adaptive methods, also in Code 8.7.2, and this choice can be justified by control theoretic arguments [13, Sect. 5.2].

Example 8.7.4 (Simple adaptive stepsize control).

- IVP for ODE $\dot{y} = \cos(\alpha y)^2$, $\alpha > 0$, solution $y(t) = \arctan(\alpha(t-c))/\alpha$ for $y(0) \in]-\pi/2, \pi/2[$
- Simple adaptive timestepping based on explicit Euler (8.2.1) and explicit trapezoidal rule (8.6.3)

Code 8.7.5: function for Ex. 8.7.4

```
def odeintadaptdriver(T,a,reltol=1e-2,abstol=1e-4):
```



```
        odeintadapt_ext(Psilow,Psihigh,T,y0,h0,reltol,abstol,hmin)

        # Plotting the exact the approximate solutions and rejected timesteps
        fig = plt.figure()
        tp = r_[0:T:T/1000.0]
        plt.plot(tp,sol(tp),'g-',linewidth=2, label=r'y(t)')
        plt.plot(t,y,'r.',label=r'y_k')
        plt.plot(rej,zeros(len(rej)),'m+',label='rejection')
        plt.title('Adaptive timestepping, rtol = %1.2f, atol = %1.4f, a
          = %d' % (reltol,abstol,a))
        plt.xlabel(r't',fontsize=14)
        plt.ylabel(r'y',fontsize=14)
        plt.legend(loc='upper left')
        plt.show()
# plt.savefig('../PICTURES/odeintadaptsol.eps')

        print '%d timesteps, %d rejected timesteps' %
          (size(t)-1,len(rej))

        # Plotting estimated and true errors
        fig = plt.figure()
        plt.plot(t,abs(sol(t) - y),'r+',label=r'true error |y(t_k) - y_k|')
        plt.plot(t,ee,'m*',label='estimated error EST_k')
```

```
        plt.xlabel(r't',fontsize=14)
        plt.ylabel(r'error',fontsize=14)
        plt.legend(loc='upper left')
        plt.title('Adaptive timestepping, rtol = %1.2f, atol = %1.4f, a = %d' % (reltol,abstol,a))
        plt.show()
# plt.savefig('../PICTURES/odeintadapterr.eps')

if __name__ == '__main__':
        odeintadaptdriver(T=2,a=20,reltol=1e-2,abstol=1e-4)
```

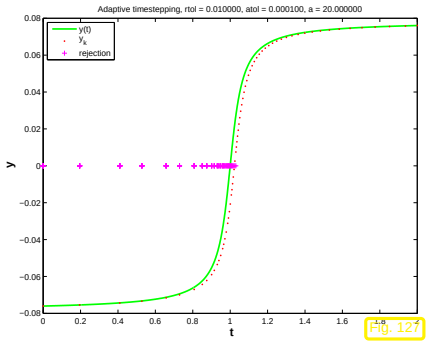



Fig. 127

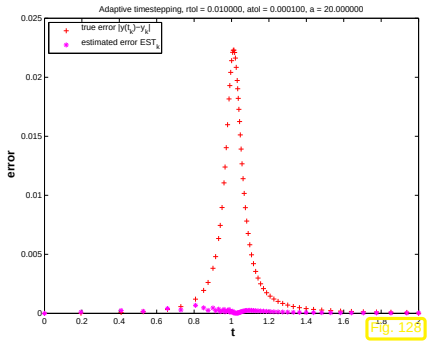


Fig. 128

Statistics: `66 timesteps, 131 rejected timesteps`

Observations:

- ☞ Adaptive timestepping well resolves local features of solution $y(t)$ at $t = 1$
- ☞ Estimated error (an estimate for the one-step error) and true error are **not** related!

◇

Example 8.7.6 (Gain through adaptivity). → Ex. 8.7.4

Simple adaptive timestepping from previous experiment Ex. 8.7.4.

New: initial state $y(0) = 0$!

Now we study the dependence of the maximal point error on the computational effort, which is proportional to the number of timesteps.

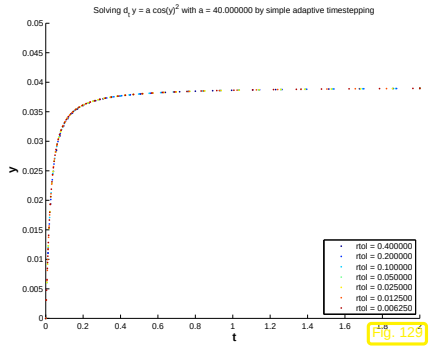


Fig. 129

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

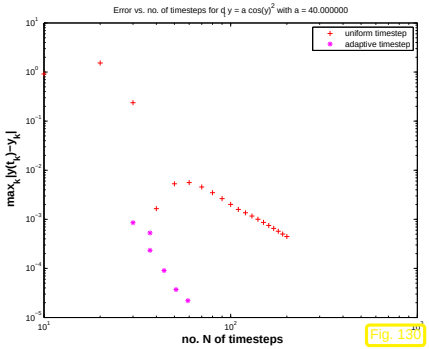


Fig. 130

Error vs. computational effort

Observations:

- ☞ Adaptive timestepping achieves much better accuracy for a fixed computational effort.

◇

Example 8.7.7 ("Failure" of adaptive timestepping). → Ex. 8.7.6

Same ODE and simple adaptive timestepping as in previous experiment Ex. 8.7.6. Same evaluations.

Now: initial state $y(0) = -0.0386$ as in Ex. 8.7.4

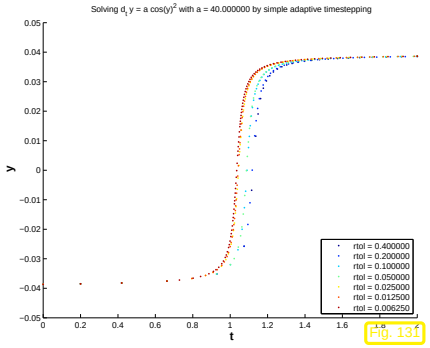


Fig. 131

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

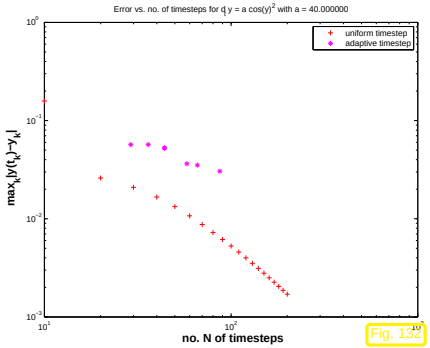


Fig. 132

Error vs. computational effort

Observations:

☞ Adaptive timestepping leads to larger errors at the same computational cost as uniform timestepping.

Explanation: the position of the steep step of the solution has a sensitive dependence on an initial value $y(0) \approx -\pi/2$. Hence, small local errors in the initial timesteps will lead to large errors at around time $t \approx 1$. The stepsize control is mistaken in condoning these small one-step errors in the first few steps and, therefore, incurs huge errors later.

◇

Remark 8.7.8 (Refined local stepsize control).

The above algorithm (Code 8.7.2) is simple, but the rule for increasing/shrinking of timestep arbitrary and "wastes" informatiom contained in EST_k : $\mathtt{TOL}$:

More ambitious goal **!** When $\mathrm{EST}_k > \mathtt{TOL}$: stepsize adjustment better $h_k = ?$
When $\mathrm{EST}_k < \mathtt{TOL}$: stepsize prediction good $h_{k+1} = ?$

Assumption: At our disposal are two discrete evolutions:

- $\boldsymbol{\Psi}$ with order$(\boldsymbol{\Psi}) = p$ (➜ "low order" single step method)
- $\widetilde{\boldsymbol{\Psi}}$ with order$(\widetilde{\boldsymbol{\Psi}}) > p$ (➜ "higher order" single step method)

These are the same building blocks as for the simple adaptive strategy employed in Code 8.7.2 (, passed as arguments `Psilow`, `Psihigh` there).

Asymptotic expressions for one-step error for $h \to 0$:

$$\begin{aligned} \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &= ch^{p+1} + O(h_k^{p+2}) \,, \\ \widetilde{\boldsymbol{\Psi}}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &= O(h^{p+2}) \,, \end{aligned} \tag{8.7.5}$$

with some $c > 0$.

Why h^{p+1}? Remember estimate (**??**) from the error analysis if the explicit Euler method: we also found $O(h_k^2)$ there for the one-step error of a single step method of order 1.

Heuristics: the timestep h is small ➥ "higher order terms" $O(h^{p+2})$ can be ignored.

$$\begin{aligned} \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &\doteq ch_k^{p+1} + O(h_k^{p+2}) \,, \\ \widetilde{\boldsymbol{\Psi}}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &\doteq O(h_k^{p+2}) \,. \end{aligned} \quad \Rightarrow \quad \boxed{\mathrm{EST}_k \doteq ch_k^{p+1}} \,. \tag{8.7.6}$$

✎ notation: $\doteq$ equality up to higher order terms in h_k

$$\mathrm{EST}_k \doteq ch_k^{p+1} \quad \Rightarrow \quad c \doteq \frac{\mathrm{EST}_k}{h_k^{p+1}} \,. \tag{8.7.7}$$

Available in algorithm, see (8.7.3)

For the sake of *accuracy* (stipulates "$\mathrm{EST}_k < \mathtt{TOL}$") & *efficiency* (favors "$>$") we aim for

$$\mathrm{EST}_k \overset{!}{=} \mathtt{TOL} := \max\{\mathtt{ATOL}, \|\mathbf{y}_k\| \, \mathtt{RTOL}\} \,. \tag{8.7.8}$$

What timestep h_* can actually achieve (8.7.8), if we "believe" in (8.7.6) (and, therefore, in (8.7.7))?

$$\text{(8.7.7) \& (8.7.8)} \quad \Rightarrow \quad \mathtt{TOL} = \frac{\mathrm{EST}_k}{h_k^{p+1}} h_*^{p+1} \,.$$

➤ "'Optimal timestep": (stepsize prediction) $\boxed{h_* = h \sqrt[p+1]{\frac{\mathtt{TOL}}{\mathrm{EST}_k}}}$. (8.7.9)

adjusted stepsize (**A**)

suggested stepsize (**B**)

(**A**): In case $\mathrm{EST}_k > \mathtt{TOL}$ ➢ *repeat step* with stepsize h_*.

(**B**): If $\mathrm{EST}_k \leq \mathtt{TOL}$ ➢ use h_* as stepsize for *next step*.

Code 8.7.9: refined local stepsize control for single step methods

```
from numpy import *
from numpy.linalg import norm
import matplotlib.pyplot as plt

def odeintssctrl(Psilow,p,Psihigh,T,y0,h0,reltol,abstol,hmin):
        t = [0]; y = [y0]; h = h0          #
        while t[-1] < T and h > hmin:      #
```

```
8                 yh = Psihigh(h,y0)        #
9                 yH = Psilow(h,y0)         #
10                est = norm(yH-yh)         #
11                tol = max(reltol*norm(y[-1]),abstol)     #
12                h = h*max(0.5, min(2,(tol/est)**(1.0/(p+1))))    #
13                if est < tol:    #
14                        y0 = yh; y.append(y0);
                            t.append(t[-1]+min(T-t[-1],h))    #
15
16        return (array(t),array(y)) # convert to numpy array
17
18 if __name__ == '__main__':
19        """this␣example␣demonstrates␣that␣a␣slight␣reduction␣in␣the
20 ␣␣␣␣␣␣␣␣tolerance␣can␣have␣a␣large␣effect␣on␣the␣quality␣of␣the␣
   solution
21 ␣␣␣␣␣␣␣␣"""
22        # use two different tolerances
23        reltol1=1e-2; abstol1=1e-4
24        reltol2=1e-3; abstol2=1e-5
25        T=2; a=20
26
27        # autonomous ODE ẏ = cos(ay) and its general solution
28        f = lambda y: (cos(a*y)**2)
```



```
29        sol = lambda t: arctan(a*(t-1))/a
30        # Initial state y_0
31        y0 = sol(0)
32
33        # Discrete evolution operators, see Def. 8.2.1
34        Psilow = lambda h,y: y + h*f(y) # Explicit Euler (8.2.1)
35        o = 1 # global order of lower-order evolution
36        Psihigh = lambda h,y: y + 0.5*h*( f(y) + f(y+h*f(y)) ) # Explicit
            trapzoidal rule (8.6.3)
37
38        # Heuristic choice of initial timestep and h_min
39        h0 = T/(100.0*(norm(f(y0))+0.1)); hmin = h0/10000.0;
40        # Main adaptive timestepping loop, see Code 8.7.2
41        t1,y1 =
            odeintssctrl(Psilow,o,Psihigh,T,y0,h0,reltol1,abstol1,hmin)
42        t2,y2 =
            odeintssctrl(Psilow,o,Psihigh,T,y0,h0,reltol2,abstol2,hmin)
43
44        # Plotting the exact the approximate solutions and rejected timesteps
45        fig = plt.figure()
46        tp = r_[0:T:T/1000.0]
47        plt.plot(tp,sol(tp),'g-',linewidth=2, label=r'y(t)')
48        plt.plot(t1,y1,'r.',label='reltol:␣%1.3f,␣abstol:␣
            %1.5f'%(reltol1, abstol1))
```



```
49        plt.plot(t2,y2,'b.',label='reltol:␣%1.3f,␣abstol:␣
            %1.5f'%(reltol2, abstol2))
50        plt.title('Adaptive␣timestepping,␣a␣=␣%d' % a)
51        plt.xlabel(r't',fontsize=14)
52        plt.ylabel(r'y',fontsize=14)
53        plt.legend(loc='upper␣left')
54        plt.show()
55 # plt.savefig('../PICTURES/odeintssctrl.eps')
```

Comments on Code 8.7.8 (see comments on Code 8.7.2 for more explanations):

- Input arguments as for Code 8.7.2, except for p $\hat{=}$ order of lower order discrete evolution.
- line 12: compute presumably better local stepsize according to (8.7.9),
- line 13: decide whether to repeat the step or advance,
- line 14: extend output arrays if current step has not been rejected.

Example 8.7.10 (Adaptive timestepping for mechanical problem).

Movement of a point mass in a conservative force field: $t \mapsto \mathbf{y}(t) \in \mathbb{R}^2 \hat{=}$ trajectory

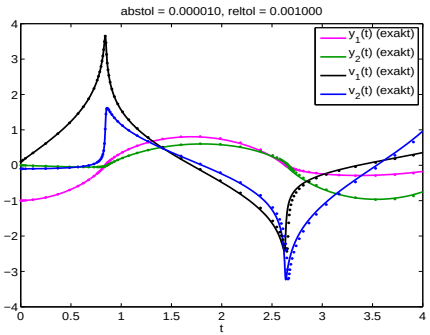
Newton's law:
$$\ddot{\mathbf{y}} = F(\mathbf{y}) := -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \,. \tag{8.7.10}$$
acceleration force

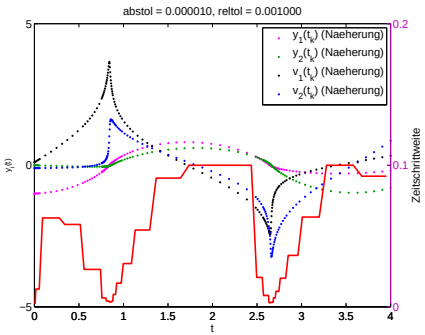
Equivalent 1st-order ODE, see Rem. 8.1.6: with velocity $\mathbf{v} := \dot{\mathbf{y}}$

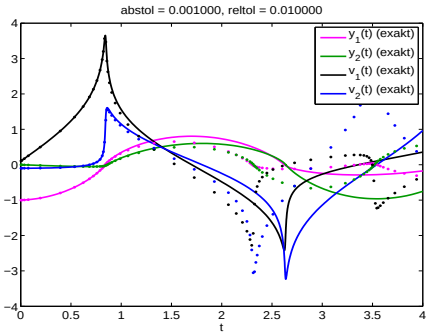
$$\begin{pmatrix} \dot{\mathbf{y}} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \end{pmatrix} \,. \tag{8.7.11}$$

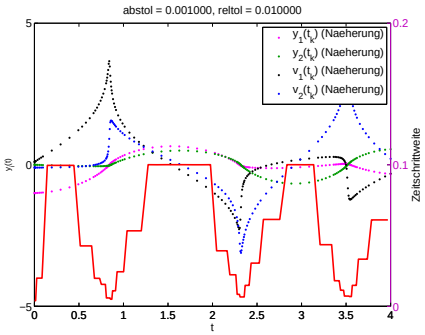
Initial values used in the experiment:

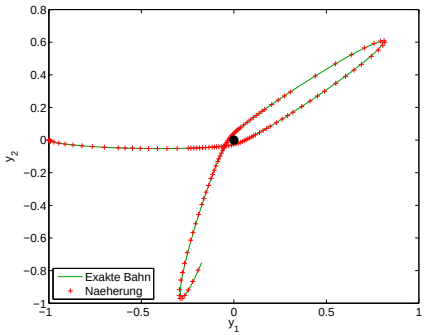
$$\mathbf{y}(0) := \begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad , \quad \mathbf{v}(0) := \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix}$$



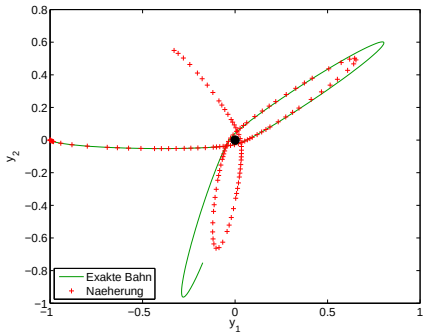








reltol=0.001, abstol=1e-5



reltol=0.01, abstol=1e-3

Observations:

☞ Fast changes in solution components captured by adaptive approach through very small timesteps.

☞ Completely wrong solution, if tolerance reduced slightly.

◇

An inevitable consequence of time-local error estimation:

Absolute/relative tolerances do *not* allow to predict accuracy of solution!

8.8 Essential Skills Learned in Chapter 8

You should know:

- what is an autonomous ODE
- how to reduce an ODE to an autonomous first order system of ODEs
- the meaning of evolution operator
- the Euler methods with pros and contras
- the derivation and advantages of the implicit midpoint rule