ETH Lecture 401-0663-00L Numerical Methods for CSE

Numerical Methods for Computational Science and Engineering

Prof. R. Hiptmair, SAM, ETH Zurich

(with contributions from Prof. P. Arbenz and Dr. V. Gradinaru)

Autumn Term 2016

URL: http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE16.pdf

XI. Numerical Integration - Single-Step Methods -

→ Numerical solution of ODEs

11.1. Initial Value Problems for ODEs

"time variable"

ODE: $[\frac{d}{dt}y =] \dot{y} = f(t, y(t))$

right-hand-side function $f: D \subset \mathbb{R}^d \to \mathbb{R}^d$ (continuous)

solution: C^1-function $t \to y(t) \in \mathbb{R}^d$ that satisfies the ODE pointwise $\forall t \in I \subset \mathbb{R}$

$D \subset \mathbb{R}^d \triangleq$ state space

For $d > 1$ $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ can be viewed as a system of ordinary differential equations:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \iff \begin{bmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_d \end{bmatrix} = \begin{bmatrix} f_1(t, y_1, \ldots, y_d) \\ \vdots \\ f_d(t, y_1, \ldots, y_d) \end{bmatrix}.$$

$f \in C^m \Rightarrow$ solution $y \in C^{m+1}$

11.1.1. Model with ODEs

Discrete mechanics: $\ddot{y} = f(y)$

transient circuit models

reaction kinetics

.....

Example (population dynamics): Predator-prey model

$d = 2$: $u = u(t)$: prey, $v = v(t)$: predator

ODE-based model: autonomous Lotka-Volterra ODE: $(\alpha, \beta, \gamma, \delta > 0)$

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \quad \leftrightarrow \quad \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{bmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{bmatrix}, \qquad (11.1.10)$$

$v = 0$: $\dot{u} = \alpha u \Rightarrow u(t) = u(0)\, e^{\alpha t}$

$u = 0$: $\dot{v} = -\gamma v \Rightarrow$ exponential decay

Definition 11.1.8. Autonomous ODE

An ODE of the from $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, that is, with a right hand side function that does not depend on time, but only on state, is called autonomous.

$\Rightarrow$ model time-invariant models

(periodic solution)

Solution:

Initial values

Needed to fix a unique solution of an ODE

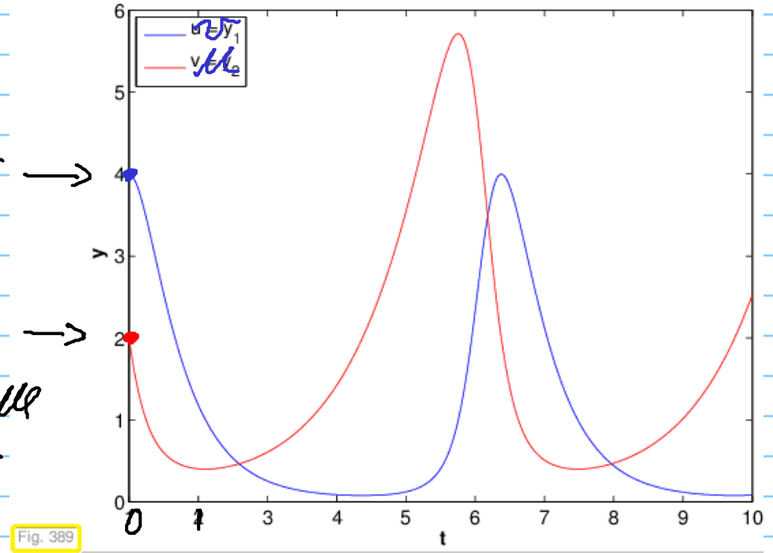


Fig. 389

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

$\Rightarrow$ In numerical integration we solve initial value problem (IVP) = ODE + initial values

$$\dot{y} = f(t, y), \quad y(t_0) = y_0$$

(t_0: initial time)

$\Rightarrow$ For autonomous ODEs:

If $t \mapsto y(t)$ solution $\Rightarrow$ $t \mapsto y(t - \tau)$ is a solution

$\Rightarrow$ Free choice of t_0: $t_0 := 0$

Solution of IVP → function of time
→ curve in state space: $t \to D$

predator

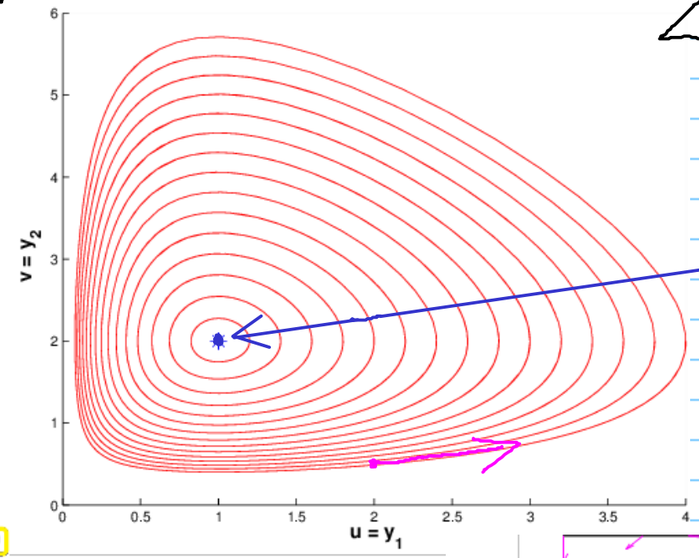


Fig. 390

Solution curves for (11.1.10)

△ trajectories (closed curves)

Stationary point $\bar{y}^*$
$\underline{f}(\bar{y}^*) = 0$

prey

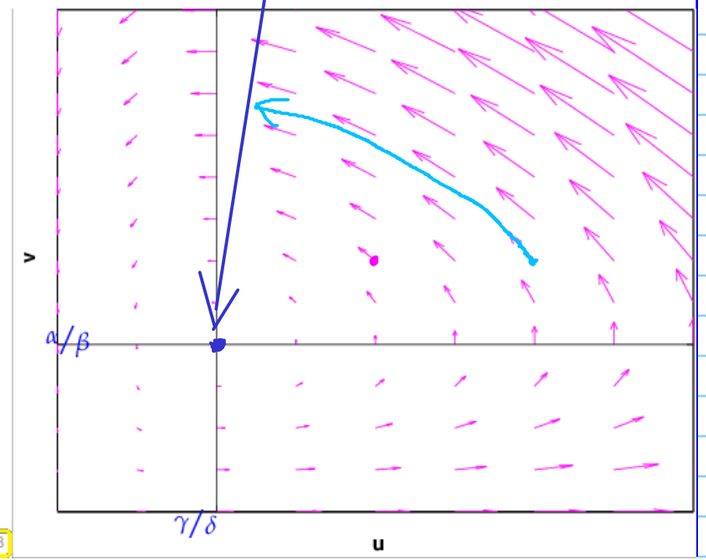


Fig. 388

→ $\hat{=}$ $\underline{f}\left(\begin{bmatrix} u \\ v \end{bmatrix}\right)$

ODE $\hat{=}$ movement of particles in velocity field described by $\underline{f}$

11.1.2. Theory of IVP

Remark: Autonomization

$$\dot{\bar{y}} = \underline{f}(t, \bar{y}) \quad : \quad \bar{z}(t) = \begin{bmatrix} \bar{y}(t) \\ t \end{bmatrix} \in \mathbb{R}^{d+1}$$

$$\dot{\bar{z}}(t) = \begin{bmatrix} \dot{\bar{y}}(t) \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \underline{f}(z_{d+1}, t), \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix} \\ 1 \end{bmatrix}}_{=: \underline{g}(\bar{z})}$$

$$\underbrace{\dot{\bar{z}} = \underline{g}(\bar{z})}_{\text{autonomous ODE}}$$

Remark: Conversion to 1st-order ODE

$$\ddot{\bar{y}} = f(\bar{y}) \qquad [\text{2nd-order ODE}]$$

$$\bar{z}(t) = \begin{bmatrix} \bar{y}(t) \\ \dot{\bar{y}}(t) \end{bmatrix} \in \mathbb{R}^{2d} \Rightarrow \dot{\bar{z}}(t) = \begin{bmatrix} \dot{\bar{y}} \\ \ddot{\bar{y}} \end{bmatrix} = \underbrace{\begin{bmatrix} z_{d+1} \\ \vdots \\ z_{2d} \\ \underline{f}\left(\begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix}\right) \end{bmatrix}}_{=: \underline{g}(\bar{z})}$$

$$\dot{\bar{z}} = \underline{g}(\bar{z})$$

(4)

Remark : Intrinsic Domain of definition of solutions of IVP

Example : $\dot{y} = y^2$, $y(0) = y_0$

We find the solutions

$$y(t) = \begin{cases} \frac{1}{y_0^{-1} - t} & \text{, if } y_0 \neq 0 \,, \\ 0 & \text{, if } y_0 = 0 \,, \end{cases} \tag{11.1.37}$$

with domains of definition

$$J(y_0) = \begin{cases}]-\infty, y_0^{-1}[& \text{, if } y_0 > 0 \,, \\ \mathbb{R} & \text{, if } y_0 = 0 \,, \\ ]y_0^{-1}, \infty[& \text{, if } y_0 < 0 \,. \end{cases}$$

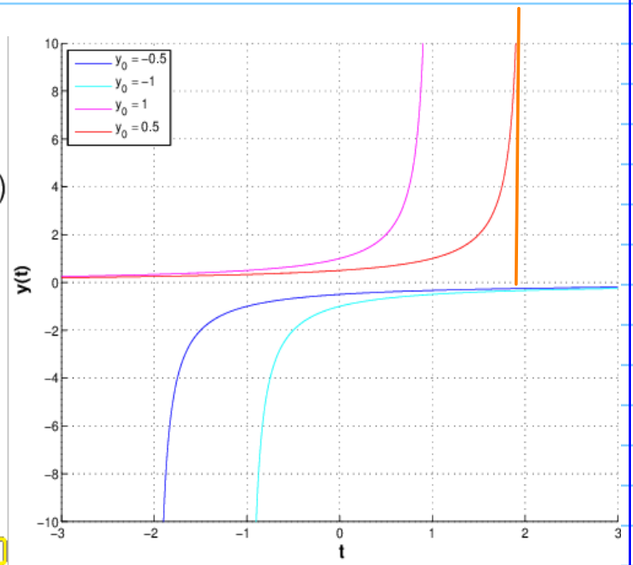


Fig. 396

Blow-up after finite time

"built into the solution"

11.1.3. Evolution operators [for $\dot{y} = f(y)$]

[Assume: Existence $\forall t \in \mathbb{R}$, $\forall y_0 \in D$]

$y_0 \in D \Rightarrow$ Unique solution $\dot{y}(t) = f(y(t))$, $y(0) = y_0$

↳ $\hat{=}$ curve in state space

$t = t^*$ fixed: $y_0 \longrightarrow y(t^*)$

Definition 11.1.39. Evolution operator/mapping

Under Ass. 11.1.38 the mapping

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

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/mapping for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

$t \longrightarrow \Phi^t y_0 \hat{=}$ trajectory : $\frac{d}{dt} \Phi^t y = f(\Phi^t y)$

$y \longrightarrow \Phi^t y \hat{=}$ mapping of state space

Φ encodes the complete set of solutions of the ODE

(5)

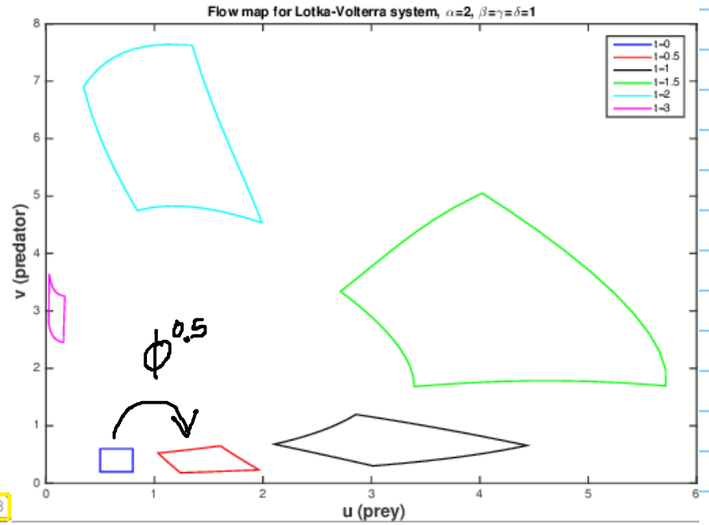


Fig. 398

state mapping $\mathbf{y} \mapsto \Phi^t \mathbf{y}$

11.2. Polygonal Approximation Methods

ODE: $\dot{y} = f(y)$

Goal: Approximate model for evolution operator Φ

Focus on discrete set of times (temporal mesh)

$$\mathcal{M} = \{ 0 = t_0 < t_1 < \dots < t_N = T \}$$

$t_N = T$: final time

11.2.1 Explicit* Euler method

Difference quotient on $\mathcal{M}$

$h_k := t_{k+1} - t_k$

$$\dot{y} = f(y)$$

"forward difference quotient"

$$\frac{y_{k+1} - y_k}{h_k} = f(y_k)$$

+ set $y_0 = y(0)$ [initial value]

$\Rightarrow$ sequence: $\{ y_0, y_1, \dots, y_N \}$ with $y_k \approx y(t_k)$

* created by f-eval above

Geometry: polygonal approximation of trajectories ▷

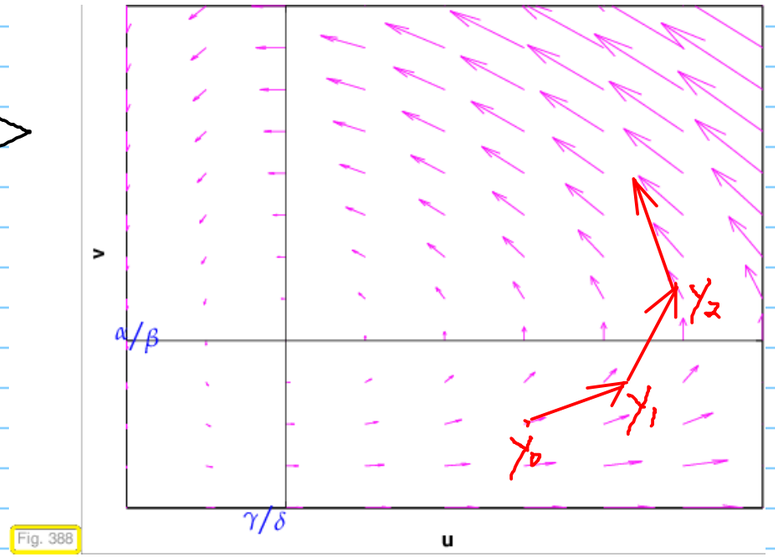


Fig. 388

11.2.2. Implicit Euler method

$$\dot{y} = f(y) \quad \Longrightarrow \quad \frac{y_{k+1} - y_k}{h_k} = f(y_{k+1}) \quad [1]$$

↑ "backward difference quotient"

to build sequence $(y_0, y_1, \ldots, y_N)$ ← given initial value

⇒ solve system of d equ. in each step for d unknowns

11.2.3. Implicit midpoint method

$$\dot{y} = f(y) \quad \Longrightarrow \quad \frac{y_{k+1} - y_k}{h_k} = \left[f\left(y\left(\frac{t_k + t_{(k+1)}}{2} \right) \right) \right]$$

↑ d.q.

↑ not available

polygonal approx: $y\left(\frac{t_k + t_{(k+1)}}{2}\right) \approx \frac{1}{2} y_k + \frac{1}{2} y_{k+1}$

▷

$$\frac{y_{k+1} - y_k}{h_k} = f\left(\frac{1}{2} (y_k + y_{k+1}) \right) \quad [2]$$

Remark: Implicit methods ⇒ for small h_k [1], [2] can be solved for y_{k+1},

III.3. General Single-Step Methods

Expl. Euler: $y_{k+1} = y_k + h_k f(y_k)$ $\longrightarrow$ $\Psi^h y = y + hf(y)$

Impl. Euler: $y_{k+1} = y_k + h_k f(y_{k+1})$ $\longrightarrow$ "$\Psi^h y := z : z = y + hf(z)$"

$$y_{k+1} = \Psi(h_{k+1}, y_k) =: \Psi^h y_k$$

What makes a good Ψ?

$$y(t_1) = \Phi^{h_1} y_0 \quad \Longleftrightarrow \quad y_1 = \Psi^{h_1} y_0$$

$$\Psi : I \times D \to \mathbb{R}^d \approx \Phi : I \times D \to \mathbb{R}^d$$

must approximate the evolution operator

At least the direction of $h \to \Psi^h y$ in $h = 0$ should be $f(y)$:

Definition 11.3.5. Single step method (for autonomous ODE) → [?, Def. 11.2]

Given a discrete evolution $\Psi : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^d$, an initial state $\mathbf{y}_0$, and a temporal mesh $\mathcal{M} := \{0 =: t_0 < t_1 < \cdots < t_N := T\}$ the recursion

$$\mathbf{y}_{k+1} := \Psi(t_{k+1} - t_k, \mathbf{y}_k) , \quad k = 0, \ldots, N-1 , \tag{11.3.6}$$

defines a single step method (SSM) for the autonomous IVP $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ on the interval $[0, T]$.

Consistent discrete evolution → Minimal requirement

The discrete evolution Ψ defining a single step method according to Def. 11.3.5 and (11.3.6) for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ invariably is of the form

$$\Psi^h \mathbf{y} = \mathbf{y} + h\psi(h, \mathbf{y}) \quad \text{with} \quad \psi : I \times D \to \mathbb{R}^d \text{ continuous}, \ \psi(0, \mathbf{y}) = \mathbf{f}(\mathbf{y}) . \tag{11.3.9}$$

$$\frac{d}{dh} \Psi^h \Big|_{h=0} = 0 + \psi(0, y) + 0 \cdot \dot{\psi}(h, y) = f(y)$$

11.3.2. Asymptotic Convergence of SSM

IVP + SSM : | study $\max_k \|y_k - y(t_k)\|$ [$\|y_N - y(T)\|$]

↓

for family of meshes $(\mathcal{M}_\ell)$ [e.g. equid. meshes]

$h_\ell := \max_k |t_{k+1}^\ell - t_k^\ell| \to 0$ for $\ell \to \infty$

Example: $\dot{y} = \lambda(1-y)y$ [logistic ODE]

$y_0 \in [0,1] \Rightarrow$ solution exists on $\mathbb{R}$

Error at final time $T = 1$, equid. meshes

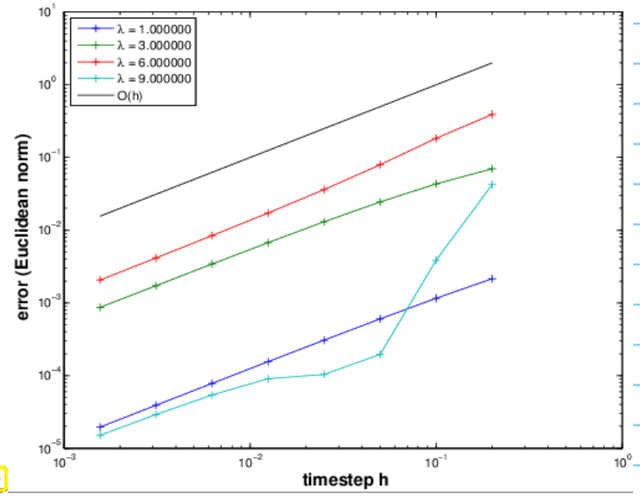


Fig. 405

explicit Euler method

△ Alg. cvg.

$err = O(h^\alpha)$

$\alpha = 1$

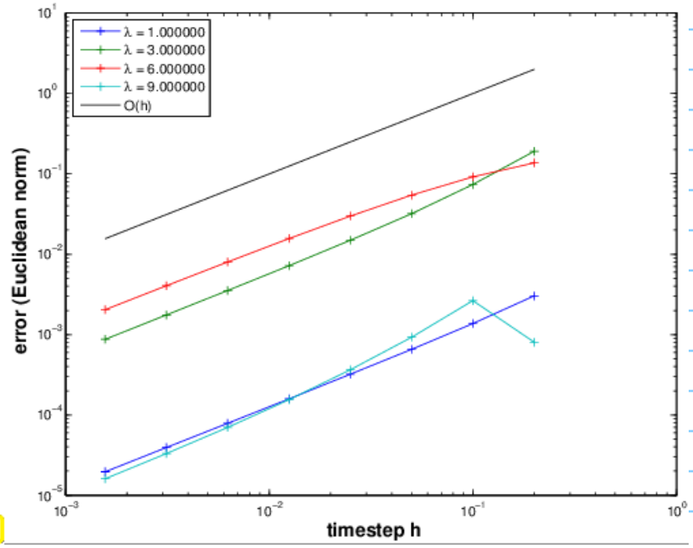


Fig. 406

implicit Euler method

△ Alg. cvg.

order = 1

impl. midp. method

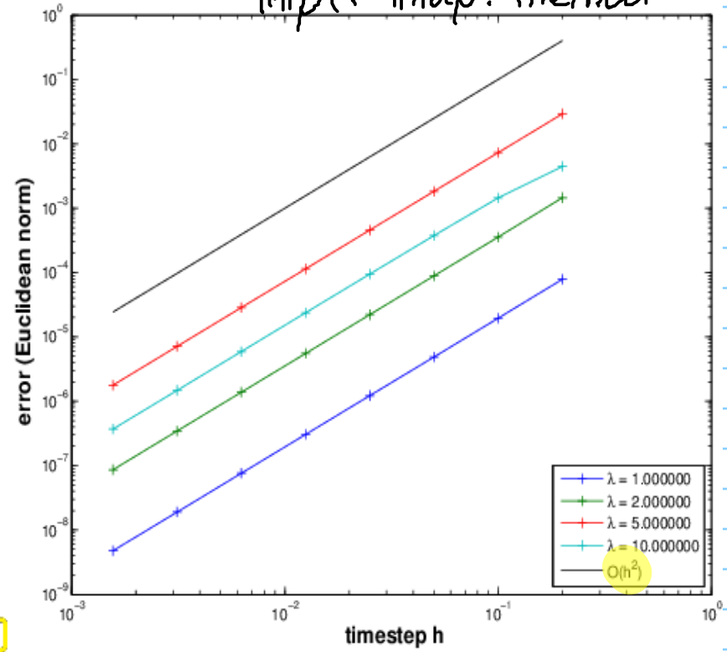


Fig. 407

△ Alg. cvg.

order = 2

All SSM converge algebraically in meshwidth h with some order $p \in \mathbb{N}$: $\text{err} = O(h^p)$

an important characteristic of the method

Repetition:

$$\dot{y} = f(y) \quad , \quad f : D \subset \mathbb{R}^d \to \mathbb{R}^d$$

Evolution operator $\Phi : \mathbb{R} \times D \to \mathbb{R}^d$

Numerical integrators $\Rightarrow$ Discrete evolution $\Psi : \mathbb{R} \times D \to \mathbb{R}^d$

$\Downarrow$

Single-step method on temporal mesh $\mathcal{M} = \{0 = t_0 < t_1 < \dots < t_N = T\}$

$\hookrightarrow$ $y_{k+1} = \Psi^{h_{k+1}} y_k$, $h_{k+1} := t_{k+1} - t_k$

Goal: $y_k \approx y(t_k)$, where $t \to y(t)$ solves IVP: $\dot{y} = f(y), y(0) = y_0$

How to get a function $y_h : [0,T] \to \mathbb{R}^d$ from $(y_k)_k$?

$\to$ by interpolation: Euler method $\to$ p.w. linear interpolation

better: piece cubic interpolation based on $y_k, f(y_k) \approx \dot{y}(t_k)$

11.4. Explicit Runge-Kutta-Methods

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

Apply num. quad!

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

$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^{s} b_i \mathbf{f}(t_0 + c_i h, \mathbf{y}(t_0 + c_i h)) \quad , \quad h := t_1 - t_0 \, . \tag{11.4.3}$$

Obtain these values by bootstrapping

approximate by some other "lower-order" discrete evolution

Quadrature formula = trapezoidal rule (7.2.5):

$$Q(f) = \tfrac{1}{2}(f(0) + f(1)) \quad \leftrightarrow \quad s = 2: \; c_1 = 0, c_2 = 1 \, , \; b_1 = b_2 = \frac{1}{2} \, , \tag{11.4.5}$$

and $\mathbf{y}(t_1)$ approximated by explicit Euler step (11.2.7)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0) \, , \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1) \, , \quad \mathbf{y}_1 = \mathbf{y}_0 + \tfrac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2) \, . \tag{11.4.6}$$

(11.4.6) = explicit trapezoidal method (for numerical integration of ODEs).

$$y_1 = y_0 + h\left[\tfrac{1}{2} f(y_0) + \tfrac{1}{2} f(y_1)\right] = y_0 + \tfrac{1}{2} h \left(f(y_0) + f(y_0 + h f(y_0))\right)$$

(10)

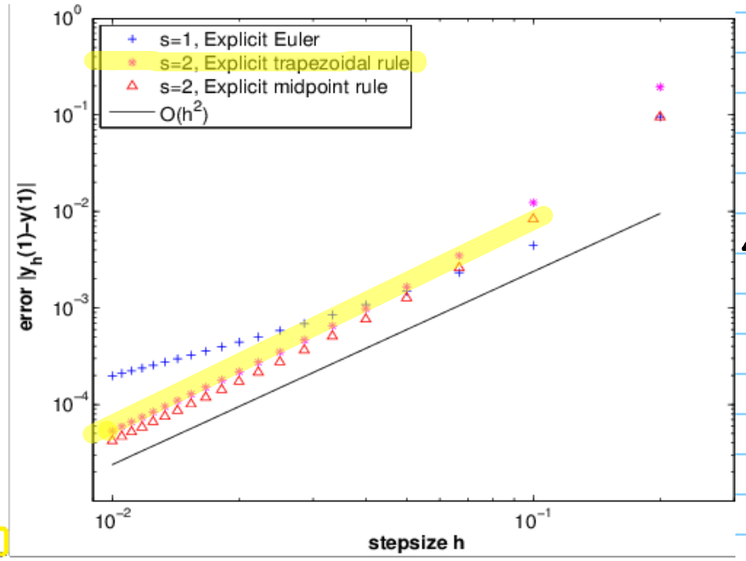


Fig. 413

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

(Logistic ODE)

◁ Alg. conv. order = 2

↳ f-eval enough to compute Ψ^h !

Definition 11.4.9. Explicit Runge-Kutta method

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i,j = 1,\ldots,s$, $s \in \mathbb{N}$, an s-stage explicit Runge-Kutta single step method (RK-SSM) for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y})$, $\mathbf{f}: \Omega \to \mathbb{R}^d$, is defined by ($\mathbf{y}_0 \in D$)

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

The vectors $\mathbf{k}_i \in \mathbb{R}^d$, $i = 1,\ldots,s$, are called increments, $h > 0$ is the size of the timestep.

≙ coefficient

Impl.: $k_1 = f(t_0 + c_1 h, y_0)$ → k_2 → k_3 → .. → k_s

Notation: Butcher scheme

Butcher scheme notation for explicit RK-SSM

Shorthand notation for (explicit) Runge-Kutta methods [?, (11.75)]

Butcher scheme ▷

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

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|ccccc} c_1 & 0 & & \cdots & & 0 \\ c_2 & a_{21} & \ddots & & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ c_s & a_{s1} & & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & & \cdots & b_{s-1} & b_s \end{array} \qquad (11.4.11)$$

$$y(t) \approx \sum_{i=1}^{s} w_i(t)\, k_i \quad \text{"dense output"}$$

RK-SSM consistent ?

Consistent discrete evolution

The discrete evolution Ψ defining a single step method according to Def. 11.3.5 and (11.3.6) for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ invariably is of the form

$$\Psi^h \mathbf{y} = \mathbf{y} + h\psi(h,\mathbf{y}) \quad \text{with} \quad \begin{array}{l} \psi : I \times D \to \mathbb{R}^d \text{ continuous,} \\ \psi(0,\mathbf{y}) = \mathbf{f}(\mathbf{y})\,. \end{array} \qquad (11.3.9)$$

Discr. Evol.: $\Psi^h y = y + h \underbrace{\sum_{i=1}^{s} b_i K_i(y)}_{\psi(h,y)}$

for $h = 0$: $K_i = f(y_0)$

$$\Rightarrow \quad \Psi(0,y) = \sum_{i=1}^{s} b_i f(y) \overset{!}{=} f(y) \Leftrightarrow \sum_{i=1}^{s} b_i = 1$$

an order condition — Consistency conditions for RK-SSM

Examples :

Explicit Euler method (11.2.7): $s = 1$

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \quad \rightarrow \quad \text{order} = 1$$

explicit trapezoidal rule (11.4.6): $s = 2$

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad \rightarrow \quad \text{order} = 2$$

Classical 4th-order RK-SSM:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} \quad \rightarrow \quad \text{order} = 4$$

Goal : High order few stages

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

Why is high order desirable ?

→ For the sake of efficiency !

$s \hat{=}$ no. of stages

Order-p method : assume $\text{err}(h) \approx Ch^p$

Cost: $W \approx s/h \approx$ no. of equidistant timesteps

to achieve error reduction by factor $\rho > 1$

$$\frac{C h_{old}^p}{C h_{new}^p} \overset{!}{=} \rho$$

$$\Rightarrow \quad h_{new} = \rho^{-1/p} \cdot h_{old}$$

$$W_{new} = \rho^{1/p} \cdot W_{old}$$

The larger p, the less effort is needed for prescribed error reduction.

$\sim$ # f-evaluations

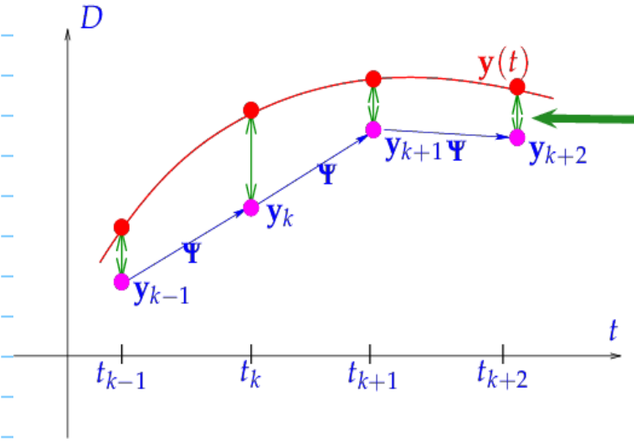
Convergence analysis: explicit Euler (equid. timestep)

Assume: • f smooth

• $\| f(x) - f(z) \| \leq L \| x - z \|$ (*)

Error: $e_k := y_k - y(t_k) \Rightarrow \Psi^h y = y + h f(y)$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) , \quad k = 1, \ldots, N-1 . \tag{11.2.7}$$



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

◁ — ≙ trajectory $t \mapsto \mathbf{y}(t)$

— ≙ Euler polygon,

• ≙ $\mathbf{y}(t_k)$,

• ≙ $\mathbf{y}_k$,

⟶ ≙ discrete evolution $\Psi^{t_{k+1}-t_k}$

$y_{k+1} = \Psi^{h_k} y_k$

Fundamental error splitting

$$\begin{aligned} \mathbf{e}_{k+1} &= \Psi^{h_k}\mathbf{y}_k - \Phi^{h_k}\mathbf{y}(t_k) \\ &= \underbrace{\Psi^{h_k}\mathbf{y}_k - \Psi^{h_k}\mathbf{y}(t_k)}_{\text{propagated error}} \\ &\quad + \underbrace{\Psi^{h_k}\mathbf{y}(t_k) - \Phi^{h_k}\mathbf{y}(t_k)}_{\text{one-step error}} . \end{aligned} \tag{11.3.25}$$

$y(t_{k+1}) = \Phi^{h_k} y(t_k)$

↑ exact solution

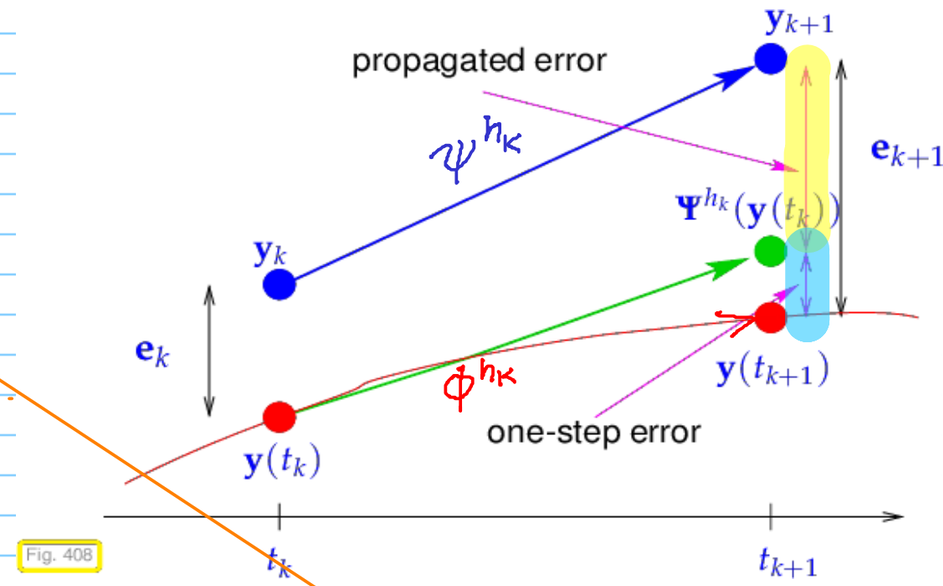


Fig. 408

Propagated error:

$$\begin{aligned} \| \Psi^h y_k - \Psi^h y(t_k) \| &= \| y_k + h f(y_k) - y(t_k) - h f(y(t_k)) \| \\ &= \| e_k + h (f(y_k) - f(y(t_k)) \| \\ &\overset{(*)}{\leq} (1 + hL) \| e_k \| \end{aligned}$$

One step error:

$$\begin{aligned} \Psi^h y(t_k) - y(t_k + h) &= y(t_k) + h f(y(t_k)) - y(t_k + h) \\ \text{Use ODE:} &= y(t_k) + h \dot{y}(t_k) - y(t_k + h) \\ \text{Taylor} &= \tfrac{1}{2} h^2 \ddot{y}(\tau) , \quad t_k \leq \tau \leq t_{k+1} \end{aligned}$$

③ Obtain *recursion* for error norms $\epsilon_k := \|\mathbf{e}_k\|$ by $\triangle$-inequality:

(13)

$$\epsilon_{k+1} \le (1+h_k L)\epsilon_k + \rho_k\,, \quad \rho_k := \tfrac{1}{2}h_k^2 \max_{t_k\le\tau\le t_{k+1}} \|\ddot{\mathbf{y}}(\tau)\| \,. \tag{11.3.30}$$

↳ bound for one-step error

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

$$\epsilon_k \le \sum_{l=1}^{k}\prod_{j=1}^{l-1}(1+Lh_j)\,\rho_l\,, \quad k=1,\ldots,N\,. \tag{11.3.31}$$

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

$$(11.3.31) \;\Rightarrow\; \epsilon_k \le \sum_{l=1}^{k}\prod_{j=1}^{l-1}\exp(Lh_j)\cdot\rho_l = \sum_{l=1}^{k}\exp(L\textstyle\sum_{j=1}^{l-1}h_j)\rho_l\,.$$

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

$y \in C^2$ required

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

$$\|\mathbf{y}_k - \mathbf{y}(t_k)\| \le T\exp(LT)\max_{l=1,\ldots,k}h_l\cdot\max_{t_0\le\tau\le t_k}\|\ddot{\mathbf{y}}(\tau)\|\,. \tag{11.3.32}$$

$\|\mathbf{y}_k - \mathbf{y}(t_k)\| = e_k$

1st-order alg. conv. $O(h)$

11.5. Adaptive Stepsize Control

Example:

This is a special case of an "oscillating" Zhabotinski-Belousov reaction [?]:

$$\begin{aligned} \mathrm{BrO_3^-} + \mathrm{Br^-} &\mapsto \mathrm{HBrO_2} \\ \mathrm{HBrO_2} + \mathrm{Br^-} &\mapsto \mathrm{Org} \\ \mathrm{BrO_3^-} + \mathrm{HBrO_2} &\mapsto 2\,\mathrm{HBrO_2} + \mathrm{Ce(IV)} \\ 2\,\mathrm{HBrO_2} &\mapsto \mathrm{Org} \\ \mathrm{Ce(IV)} &\mapsto \mathrm{Br^-} \end{aligned} \tag{11.5.2}$$

ODE

$$\begin{aligned} y_1 := c(\mathrm{BrO_3^-}): &\quad \dot{y}_1 = -k_1y_1y_2 - k_3y_1y_3\,, \\ y_2 := c(\mathrm{Br^-}): &\quad \dot{y}_2 = -k_1y_1y_2 - k_2y_2y_3 + k_5y_5\,, \\ y_3 := c(\mathrm{HBrO_2}): &\quad \dot{y}_3 = k_1y_1y_2 - k_2y_2y_3 + k_3y_1y_3 - 2k_4y_3^2\,, \\ y_4 := c(\mathrm{Org}): &\quad \dot{y}_4 = k_2y_2y_3 + k_4y_3^2\,, \\ y_5 := c(\mathrm{Ce(IV)}): &\quad \dot{y}_5 = k_3y_1y_3 - k_5y_5\,, \end{aligned} \tag{11.5.3}$$

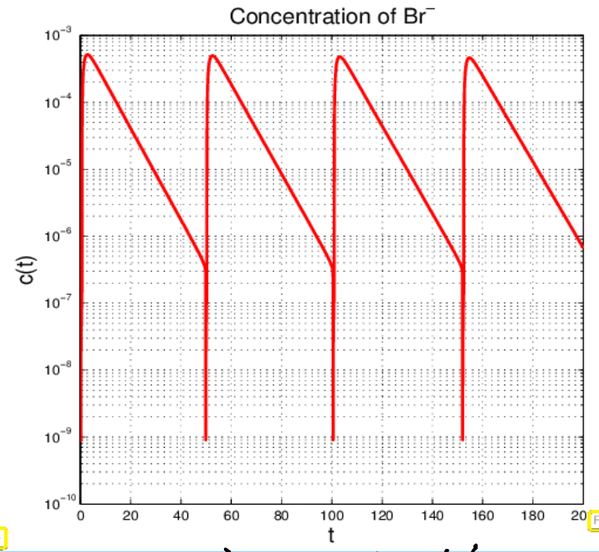


Fig. 414

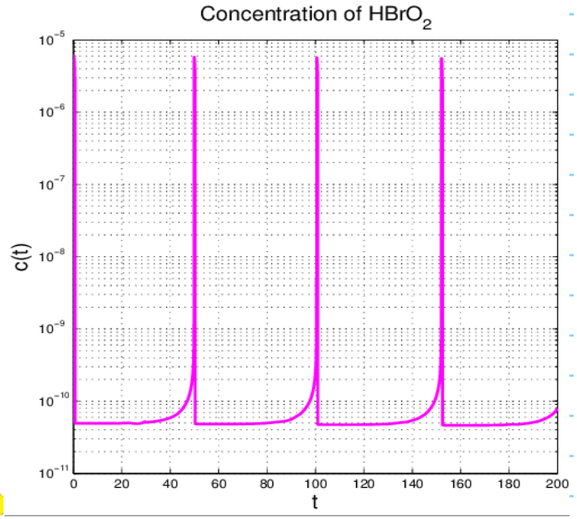


Fig. 415

$y(t)$ = highly non-uniform in time

(4)

→ suggest use of highly non-uniform temporal meshes

Example: Blow-up

We return to the "explosion ODE" of Ex. 11.1.35 and consider the scalar autonomous IVP:

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

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

As we have seen a solution exists only for finite time and then suffers a Blow-up, that is, $\lim_{t \to 1/y_0} y(t) = \infty$

: $J(y_0) =]-\infty, 1/y_0]$!

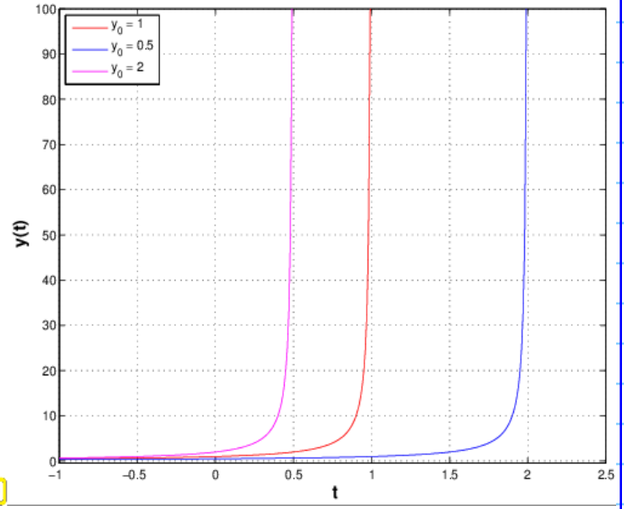


Fig. 416

Expl. Euler & uniform time step → no "numerical blow-up"

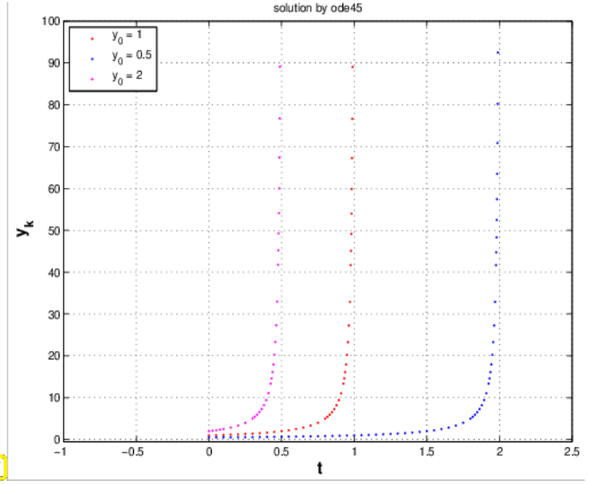


Fig. 417

Simulation with MATLAB's `ode45`:

MATLAB-code 11.5.5:

```
1 fun = @(t,y) y.^2;
2 t1,y1] = ode45(fun,[0 2],1);
3 [t2,y2] = ode45(fun,[0
    2],0.5);
4 [t3,y3] = ode45(fun,[0 2],2);
```

Adaptive stepsize selection

Stepsize adaptation for single step methods

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

use merely y_k & h_k → cheap

(I)

by comparing two SSM of different order

Idea: *Estimation of one-step error*

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

If $\mathrm{Order}(\widetilde{\Psi}) > \mathrm{Order}(\Psi)$, then we expect

$$\underbrace{\Phi^h \mathbf{y}(t_k) - \Psi^h \mathbf{y}(t_k)}_{\text{one-step error}} \approx \mathrm{EST}_k := \underbrace{\widetilde{\Psi}^h \mathbf{y}(t_k) - \Psi^h \mathbf{y}(t_k)}_{\text{computable}} . \qquad (11.5.8)$$

$= y(t_{k+1})$

replaces exact solution

Heuristics for concrete h

(II) Stepsize adaptation based on EST_k

(I)

$$EST_k \lesssim \max\{ATOL, RTOL \cdot \|y_k\|\}$$

absolute tol. relative tol.

- YES → Step accepted
  - $h_{k+1} = \alpha \cdot h_k$ ($\alpha \approx 1.1$)
  - ↓ next step
- NO → Step rejected : repeat
  - $h_k := \frac{1}{2} \cdot h_k$ → (I)

Algorithm: Embedded RK-SSM

$\Psi, \widetilde{\Psi}$ ↔ same increments

0					
$\frac{1}{3}$	$\frac{1}{3}$				
$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$			
$\frac{1}{2}$	$\frac{1}{8}$	0	$\frac{3}{8}$		
1	$\frac{1}{2}$	0	$-\frac{3}{2}$	2	
y_1	$\frac{1}{6}$	0	0	$\frac{2}{3}$	$\frac{1}{6}$
$\widehat{y}_1$	$\frac{1}{10}$	0	$\frac{3}{10}$	$\frac{2}{5}$	$\frac{1}{5}$

→ $\widetilde{\Psi}$ (order 5)

→ Ψ (order 4)

Merson's embedded RK-SSM

Remark: $y_{k+1} = \widetilde{\Psi}^{h_k} y_k$

→ Estimate error of Ψ but use $\widetilde{\Psi}$ for timestepping } works!

Example: $\dot{y} = \cos(\alpha y)$ → arctan solution

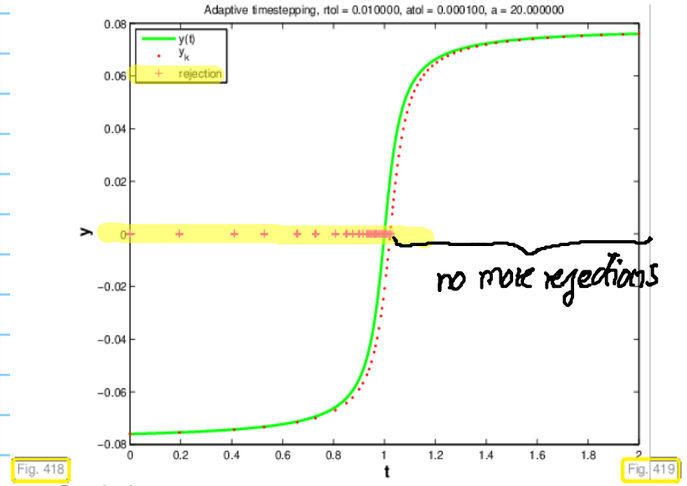


Fig. 418

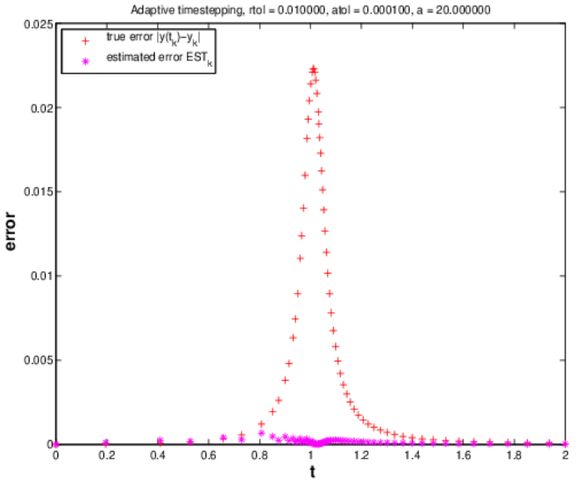


Fig. 419

Statistics: 66 timesteps, 131 rejected timesteps

EST_k are not related to total error!

C++11 code 11.5.11: Simple local stepsize control for single step methods

```
// Auxiliary function:  default norm for an EIGEN vector type
template <class State>
double _norm(const State &y) { return y.norm(); }

template <class DiscEvolOp, class State, class NormFunc =
    decltype(_norm<State>)>
std::vector<std::pair<double, State> >
odeintadapt(DiscEvolOp &Psilow, DiscEvolOp &Psihigh,
            State& y0,double T, double h0,
            double reltol, double abstol, double hmin,
            NormFunc &norm = _norm<State>) {
  double t = 0; // initial time
  State y = y0; // initial state
  double h = h0; // timestep to start with
  std::vector<std::pair<double,State>> states; // for output
  states.push_back({t, y});

  while ((states.back().first < T) && (h >= hmin))  { //
    State yh = Psihigh(h, y0); // high order discrete evolution Ψ̃^h
    State yH = Psilow(h, y0); // low order discrete evolution Ψ^h
    double est = norm(yH-yh); // local error estimate EST_k

    if (est < max(reltol*norm(y0), abstol)) { // step accepted
      y0 = yh; // use high order approximation
      t = t+min(T-t,h); // next time t_k
      states.push_back({t,y0}); //
      h = 1.1*h;  // try with increased stepsize
    }
    else // step rejected
      h = h/2; // try with half the stepsize
  }
  // Numerical integration has ground to a halt !
  if (h < hmin)  {
    cerr << "Warning: Failure at t=" << states.back().first
      << ". Unable to meet integration tolerances without reducing
         the step "
      << "size below the smallest value allowed ("<< hmin <<") at
         time t." << endl;
  }
  return states;
}
```

y(0) = 0

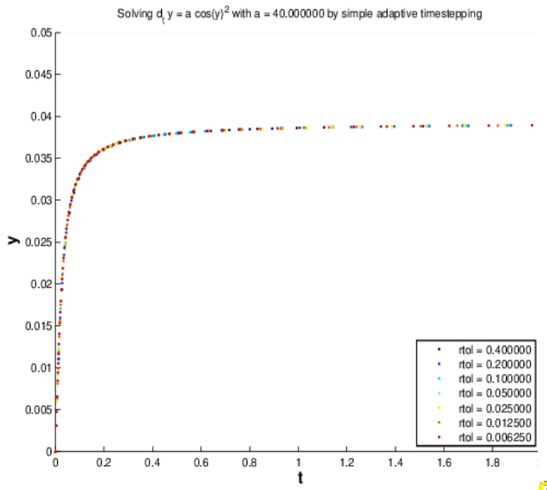


Fig. 420

Solutions $(y_k)_k$ for different values of `rtol`

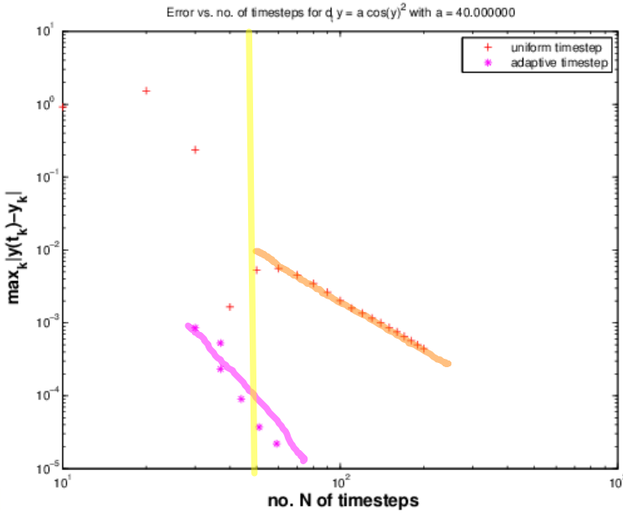


Fig. 421

Error vs. computational effort

∇ atol, rtol ✗ true size of error

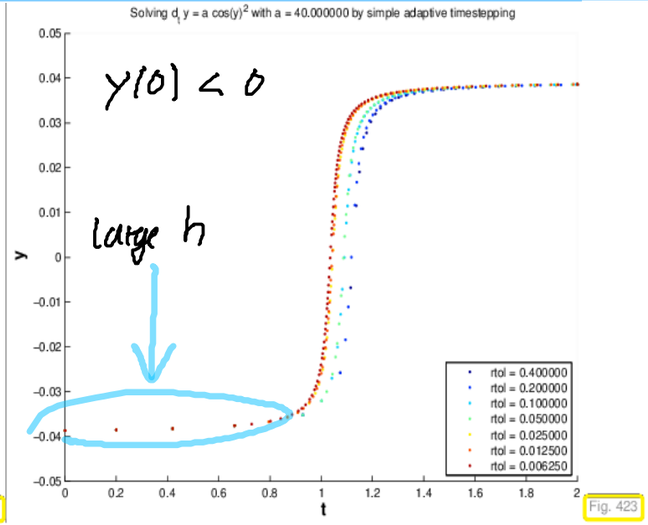


Fig. 422

Solutions $(y_k)_k$ for different values of `rtol`

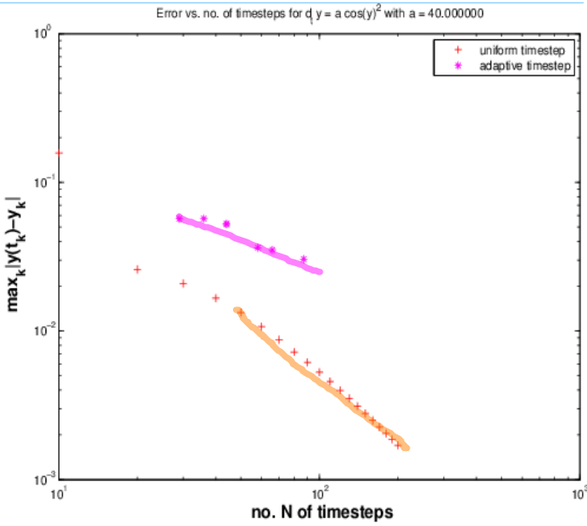


Fig. 423

Error vs. computational effort

Here: Position of step has sensitive dependence on $y(0)$

☺ "Wrong result" is still acceptable!
[→ Backward error analysis]

Stepsize prediction:

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

$\Psi \hat{=}$ order p $\qquad \widetilde{\Psi} \hat{=}$ order $p+1$

$$\mathrm{EST}_k := \widetilde{\Psi}^h y_k - \Psi^h y_k$$

$$\begin{aligned} \Psi^{h_k}\mathbf{y}(t_k) - \Phi^{h_k}\mathbf{y}(t_k) &\doteq ch_k^{p+1} + O(h_k^{p+2}), \\ \widetilde{\Psi}^{h_k}\mathbf{y}(t_k) - \Phi^{h_k}\mathbf{y}(t_k) &\doteq O(h_k^{p+2}). \end{aligned} \quad \Rightarrow \quad \boxed{\mathrm{EST}_k \doteq ch_k^{p+1}} \,. \qquad (11.5.18)$$

Efficient timestep h^*:

$$C = \left(\frac{h_k^{p+1}}{\mathrm{EST}_k}\right)^{-1} \Rightarrow \quad \begin{aligned} \mathrm{EST}_k^* &= \mathrm{TOL} \quad \text{(Goal)} \\ c(h^*)^{p+1} &= \mathrm{TOL} \end{aligned}$$

"Optimal timestep": (stepsize prediction)

$$h_* = h_k \sqrt[p+1]{\frac{\mathrm{TOL}}{\mathrm{EST}_k}} \,. \qquad (11.5.21)$$

C++11 code 11.5.22: Refined local stepsize control for single step methods

```
// Auxiliary function:  default norm for an EIGEN vector type
template <class State>
double _norm(const State &y) { return y.norm(); }

template <class DiscEvolOp, class State, class NormFunc =
    decltype(_norm<State>)>
std::vector<std::pair<double, State> >
odeintssctrl(DiscEvolOp &Psilow, unsigned int p, DiscEvolOp &Psihigh,
             State& y0,double T, double h0,
             double reltol, double abstol, double hmin,
             NormFunc &norm = _norm<State>) {
  double t = 0; // initial time
  State y = y0; // initial state
  double h = h0; // timestep to start with
  std::vector<std::pair<double,State>> states; // for output
  states.push_back({t, y});

  // Main timestepping loop
  while ((states.back().first < T) && (h >= hmin)) { //
    State yh = Psihigh(h, y0); // high order discrete evolution Ψ̃^h
    State yH = Psilow(h, y0); // low order discrete evolution Ψ^h
    double est = norm(yH-yh); // ↔ EST_k
    double tol = max(reltol*norm(y0), abstol); // effective tolerance

    // Optimal stepsize according to (11.5.21)
    h = h*max(0.5,min(2.,pow(tol/est,1./(p+1)))); //
    if (est < tol)  // step accepted
      states.push_back({t = t+min(T-t,h),y0 = yh}); // store next
         approximate state
  }
  if (h < hmin) {
    cerr << "Warning: Failure at t="
      << states.back().first
      << ". Unable to meet integration tolerances without reducing
         the step"
      << " size below the smallest value allowed ("<< hmin <<") at
         time t." << endl;
  }
  return states;
}
```