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 2015

URL: http://www.sam.ath.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf

XI. Numerical Integration

↳ Numerical solution of ODEs

$\dot{y} = f(t) \Rightarrow y(t) = \int_0^t f(\tau)\,d\tau$

11.1 Initial Value Problems (IVP) for ODEs

1st-order ODE in std. form: $\dot{y}(t) = f(t, y(t))$ (ODE)

$f : I \times D \to \mathbb{R}^d$, $I \subset \mathbb{R}$, $D \subset \mathbb{R}^d$ $\hat{=}$ right hand side (rhs)

$t \,\hat{=}$ "time variable"

$y(t) \,\hat{=}$ state $\in \mathbb{R}^d \to D \subset \mathbb{R}^d \,\hat{=}$ state space

Notation: $\dot{y}(t) := \frac{dy}{dt}(t)$

(ODE) $\Leftrightarrow$ $\begin{bmatrix} \dot{y}_1(t) \\ \vdots \\ \dot{y}_d(t) \end{bmatrix} = \begin{bmatrix} f_1(t, y_1(t), \dots, y_d(t)) \\ \vdots \\ f_d(t, y_1(t), \dots, y_d(t)) \end{bmatrix}$

Assume: f continuous

Definition 11.1.3. Solution of an ordinary differential equation

A solution of the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ with continuous right hand side function $\mathbf{f}$ is a continuously differentiable *function* "of time t" $\mathbf{y} : J \subset I \to D$, defined on an open interval J, for which $\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t))$ holds for all $t \in J$.

" f smooth [in t, y] $\Rightarrow$ y smooth [as a fnct. of t] "

Lemma 11.1.4. Smoothness of solutions of ODEs

Let $\mathbf{y} : I \subset \mathbb{R} \to D$ be a solution of the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ on the time interval I.

If $\mathbf{f} : I \times D \to \mathbb{R}^d$ is r-times continuously differentiable with respect to both arguments, $r \in \mathbb{N}_0$, then the trajectory $t \mapsto \mathbf{y}(t)$ is $r+1$-times continuously differentiable in the interior of I.

1.1.1. Examples

→ Population dynamics: growth of population with limited resources

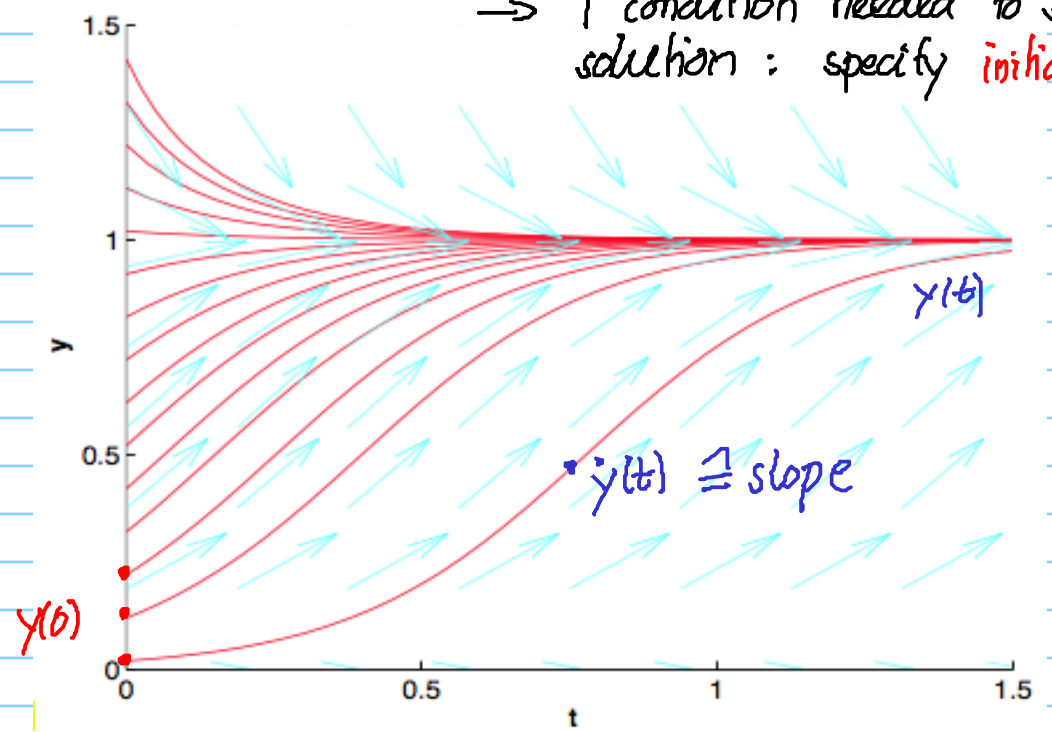
$\dot{y} = (1-y)\,y = f(y)$ [logistic ODE, $d = 1$]

state $y \mathrel{\hat=}$ density of population: State space $D = \mathbb{R}_0^+$

General solution: $y(t) = \dfrac{y(0)}{y(0) + (1-y(0))e^{-t}}$

↑ 1-parameter family of functions

→ 1 condition needed to select a unique solution: specify initial value/state $y(0) = y_0$



log. ODE $\mathrel{\hat=}$ r.h.s does not depend on t

Definition 11.1.7. 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.

Predator - prey model ($d = 2$)

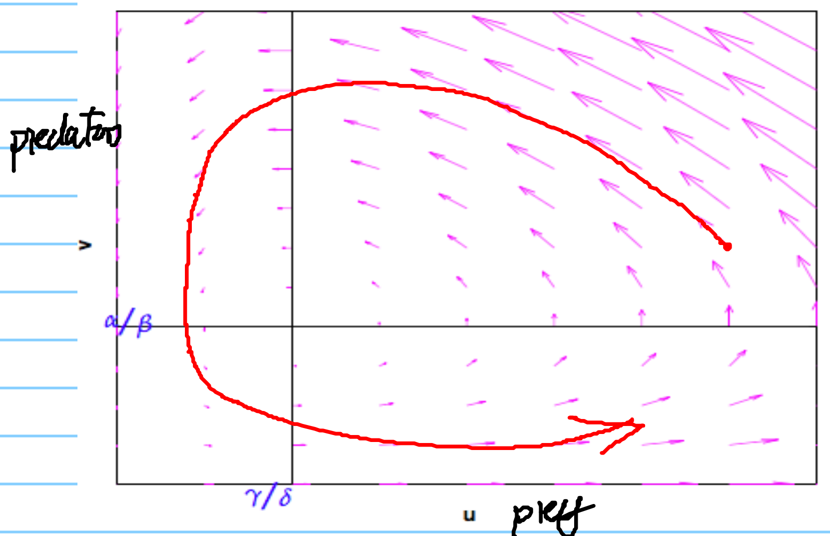
prey $\dot{u} = (1-v)\,u$

predator $\dot{v} = (u-1)\,v$

[Lotka - Volterra - ODE] autonomous

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

↑ vector field ("velocity field")



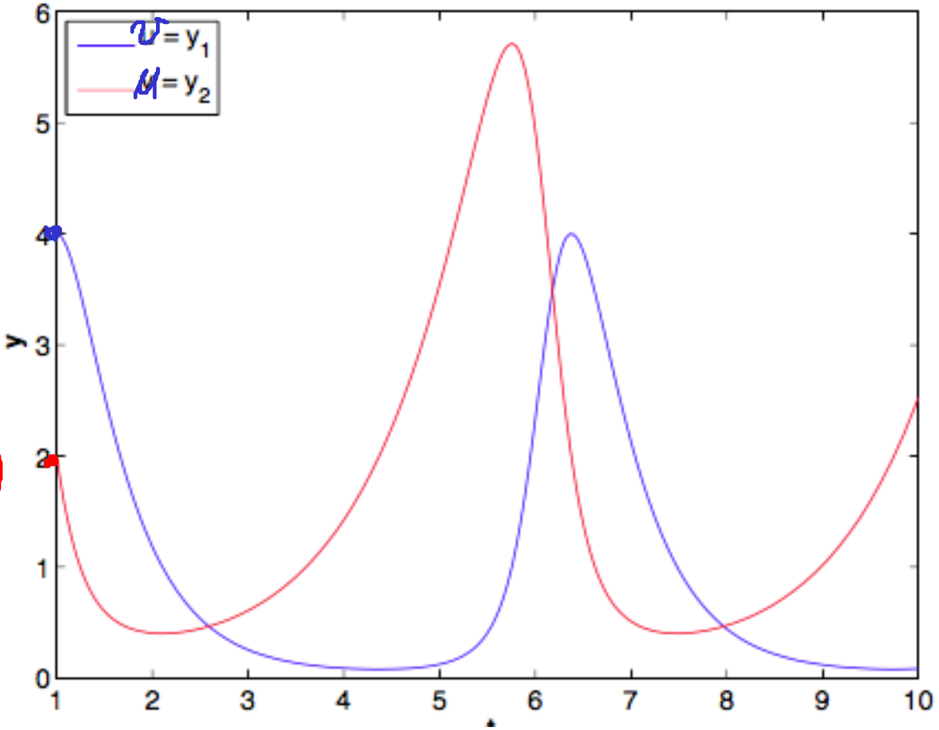
A solution* $t \to y(t)$ (trajectory)

⇔ path of a floating particle in velocity field f

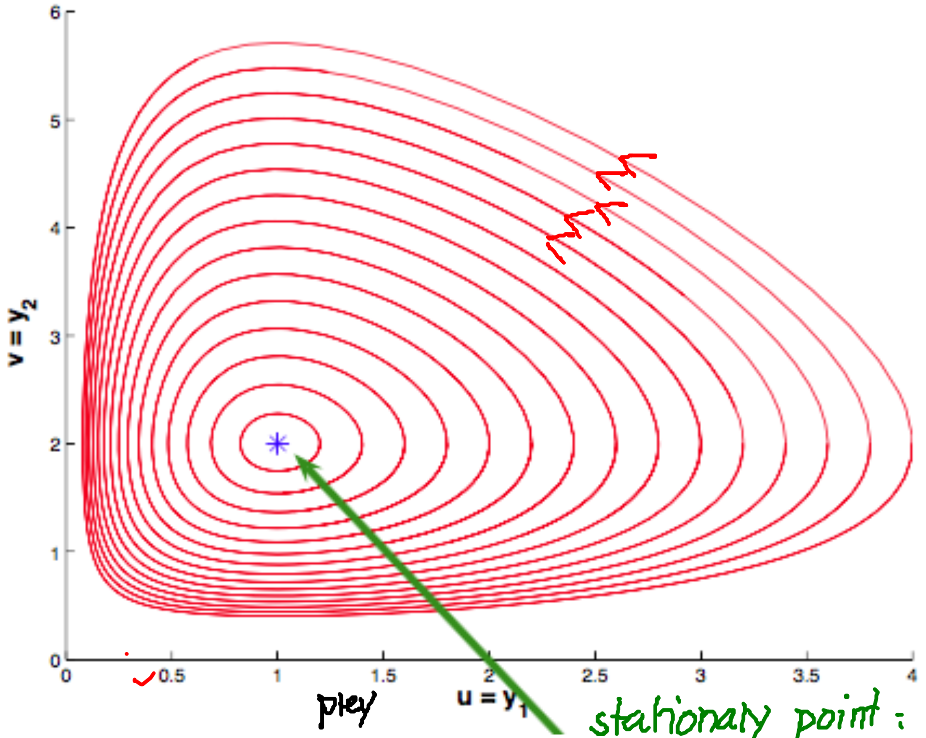
* a curve in state space

$v(0)$

$u(0)$



▷ periodic solutions



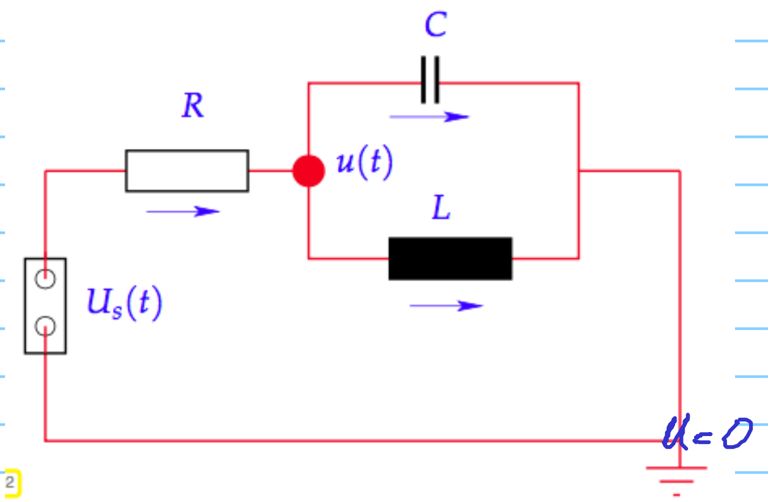
$\underline{y}(t) = \underline{y}^*$

stationary point : $f(\underline{y}^*) = 0$

Example : Transient circuit modeling

Kirchhoff's law

$$i_R(t) - i_C(t) - i_L(t) = 0$$



resistor: $\quad i_R(t) = R^{-1} u_R(t)\,,$

capacitor: $\quad i_C(t) = C\dfrac{du_C}{dt}(t)\,,$

coil: $\quad u_L(t) = L\dfrac{di_L}{dt}(t)\,.$

$\frac{d}{dt}$ →

$$\frac{d}{dt} i_R - \frac{d}{dt} i_C - \frac{d}{dt} i_L = 0$$

$$\frac{1}{R}\frac{d}{dt} u_R - C\frac{d^2}{dt^2} u_C - \frac{1}{L} u_L = 0$$

Introduce time-dependent nodal potentials :

$$\frac{1}{R}\frac{d}{dt}\left(U_s(t) - u(t)\right) - C\frac{d^2}{dt^2} u(t) - \frac{1}{L} u(t) = 0$$

→ 2nd-order ODE

11.1.2. Theory (of IVP)

$$\dot{y} = y \quad \Rightarrow \quad y(t) = Ce^t, \; C \in \mathbb{R}$$

[1-parameter family]

→ Need to fix unique solution: specify initial value

→ IVP: $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ + $\mathbf{y}(t_0) = \mathbf{y}_0$

r.h.s. $\mathbf{f} : I \times D \to \mathbb{R}^d$

$\mathbf{f}(t, \mathbf{y}) = \mathbf{f}(\mathbf{y})$: autonomous ODE / IVP

Autonomization by introducing t as an additional state component

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \; : \quad \mathbf{z}(t) := \begin{bmatrix} \mathbf{y}(t) \\ t \end{bmatrix} \text{ solves } \dot{\mathbf{z}} = \underbrace{\begin{bmatrix} \mathbf{f}(z_{d+1}, z_1, \dots, z_d) \\ 1 \end{bmatrix}}_{\text{equivalent autonomous ODE}}$$

Conversion to 1-st order ODE:

$$\ddot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \dot{\mathbf{y}}) \; : \quad \mathbf{z}(t) = \begin{bmatrix} \mathbf{y}(t) \\ \dot{\mathbf{y}}(t) \end{bmatrix} \text{ solves } \dot{\mathbf{z}} = \underbrace{\begin{bmatrix} [z_{d+1}, \dots, z_{2d}]^T \\ \mathbf{f}(z_1, \dots, z_d, z_{d+1}, \dots, z_{2d}) \end{bmatrix}}_{\text{equivalent 1st-order ODE}}$$

For 2nd-order ODEs: Initial values required for $\mathbf{y}(t_0)$, $\dot{\mathbf{y}}(t_0)$

$\Omega := I \times D$

Theorem 11.1.30. Theorem of Peano & Picard-Lindelöf [4, Satz II(7.6)], [54, Satz 6.5.1], [10, Thm. 11.10], [32, Thm. 73.1]

If the right hand side function $\mathbf{f} : \hat{\Omega} \mapsto \mathbb{R}^d$ is locally Lipschitz continuous (→ Def. 11.1.27) then for all initial conditions $(t_0, \mathbf{y}_0) \in \hat{\Omega}$ the IVP (11.1.19) has a unique *solution $\mathbf{y} \in C^1(J(t_0, \mathbf{y}_0), \mathbb{R}^d)$ with maximal (temporal) domain of definition $J(t_0, \mathbf{y}_0) \subset \mathbb{R}$.*

$\mathbf{y} \mapsto \mathbf{f}(t, \mathbf{y})$ has finite slope around every point $(t, \mathbf{y}) \in \Omega$

$$\exists L > 0 : \|\mathbf{f}(t, \mathbf{w}) - \mathbf{f}(t, \mathbf{z})\| \leq L \|\mathbf{w} - \mathbf{z}\| \quad \text{locally}$$

Not locally L.C.: $t \to \sqrt{t}$ on $[0, 1]$

Example: Temporal domain of definition

Explosion equation $\dot{y} = y^2$, $y(0) = y_0$

finite-time blow-up ▷

We find the solutions

$$y(t) = \begin{cases} \frac{1}{y_0^{-1} - t} & , \text{ if } y_0 \neq 0 \, , \\ 0 & , \text{ if } y_0 = 0 \, , \end{cases} \tag{11.1.35}$$

with domains of definition

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

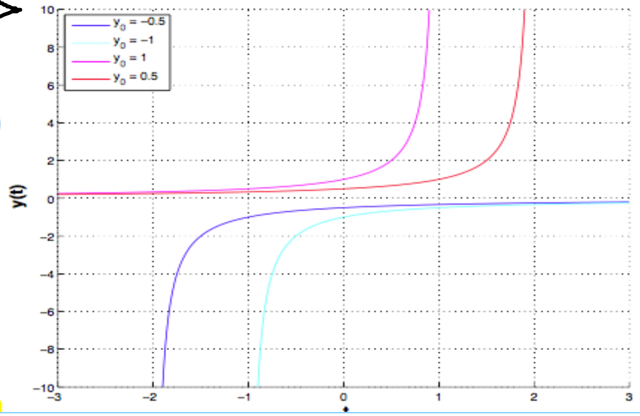


Fig. 293

Remark: Autonomous ODEs: always use $t_0 = 0$

$y(t)$ solution $\Rightarrow$ $y(t-\tau)$ also a solution

11.1.3. Evolution operators

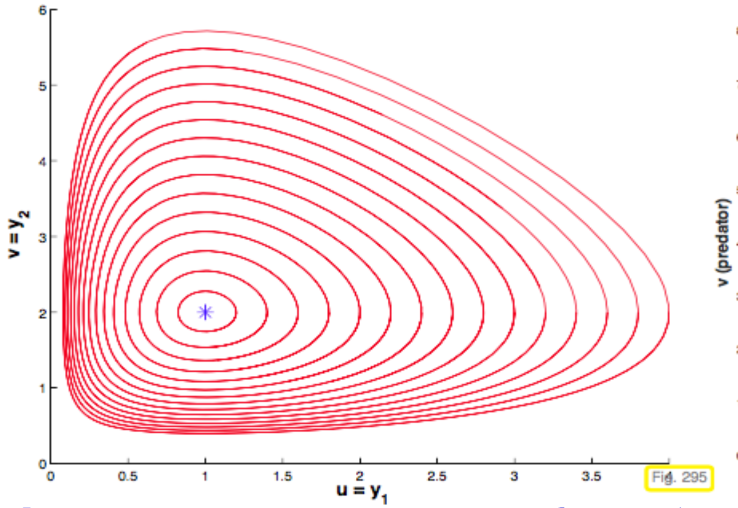
Autonomous ODE: $\dot{y} = f(y)$

[Assume: Global solution: $J(0,y_0) = \mathbb{R}$]
Existence & uniqueness

$y(t;y_0)$ solution of IVP: $\dot{y} = f(y),\ y(0) = y_0$

Evolution map: $\Phi : \begin{cases} \mathbb{R}\times D \to D \\ (t,y_0) \to y(t;y_0) \end{cases}$

Lotka-Volterra ODE, $d=2$:



294

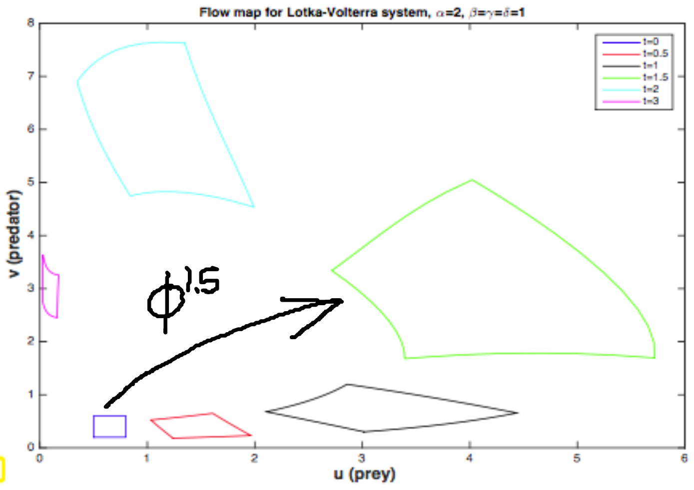


Fig. 295

y_0 fixed: trajectories $t \mapsto \Phi^t y_0 = \Phi(t,y_0)$

Fix t: state mapping $y \mapsto \Phi^t y$

ODE $\Longleftrightarrow$ Evolution map

$$\frac{\partial \Phi}{\partial t}(t,y) = f(\Phi(t,y)) \quad \forall t\in\mathbb{R},\ y\in D$$

$$f(z) = \frac{\partial \Phi}{\partial t}(t,(\Phi^t)^{-1}(z)) \ , \quad \forall z \in D$$

11.2. Polygonal approximation methods

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

We want
- $\to$ $y(T)$ for some "final time" $T > t_0$
- $\to$ approximate trajectory $t \to y_h(t) \approx y(t)$

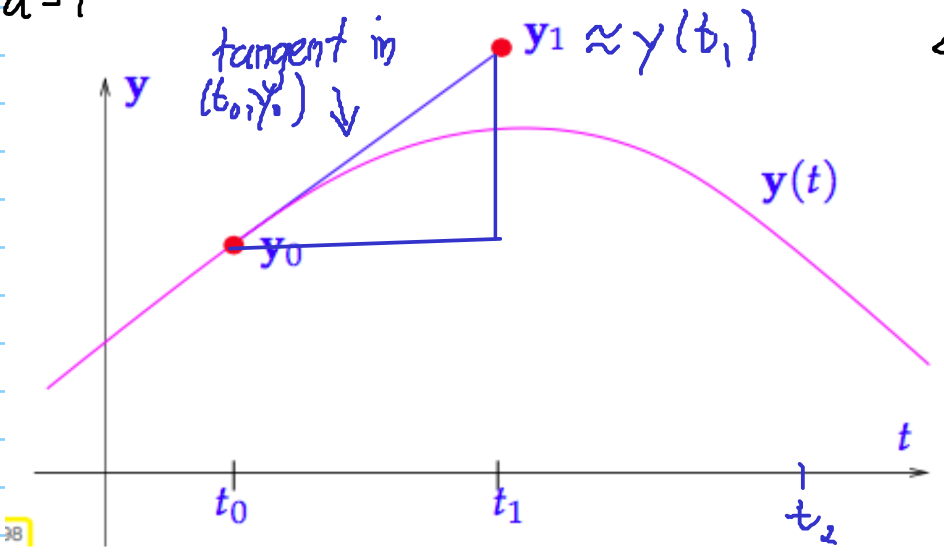
Temporal mesh: $\mathcal{M} = \{t_0 < t_1 < t_2 < \dots < t_N = T\}$

11.2.1. Explicit Euler method

Idea: Follow tangents over short times

$\dot{y} = f(t,y) \Rightarrow f(t,y(t))$ gives slope in t

$d = 1$



$\triangleleft$ 1st step

Then continue with y_1

(timestepping)

$\downarrow$

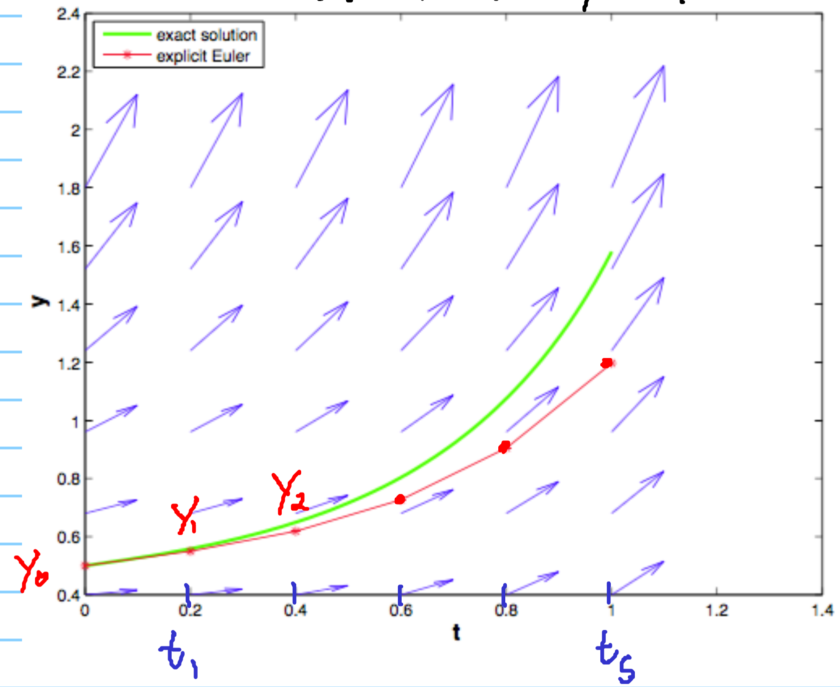
produces a sequence $y_0, y_1, y_2, \dots, y_N$

$y_h(t)$ by p.w. linear interpolation

$\Downarrow$

— $\hat{=}$ polygon

Ricati ODE: $\dot{y} = y^2 + t^2$



Recursion formula $\quad y_{k+1} = y_k + (t_{k+1} - t_k)\, f(t_k, y_k)$

$$\underbrace{\frac{y_{k+1} - y_k}{t_{k+1} - t_k}}_{\text{Difference quotient}} = f(t_k, y_k)$$

$\downarrow$ anchor time

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

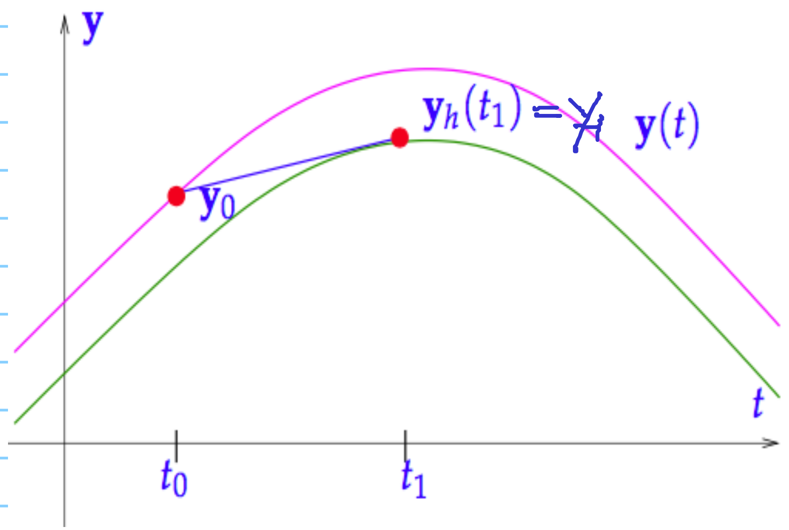
II.2.2. Implicit Euler method

Choose anchor point t_{k+1}

$$\cancel{f(t_{k+1}, y(t_{k+1}))}$$

$$\underbrace{\frac{y_{k+1} - y_k}{t_{k+1} - t_k}}_{\text{Difference quotient}} = f(t_{k+1}, y_{k+1}) \qquad \text{(IE)}$$

$\to$ $d \times d$ (non-linear) system of equations for y_{k+1}



Geometry of implicit Euler method:

Approximate solution through (t_0, y_0)

- straight line through (t_0, y_0)
- with slope $f(t_1, y_1)$

$\triangleleft$ — $\hat{=}$ trajectory through (t_0, y_0),

— $\hat{=}$ trajectory through (t_1, y_1),

— $\hat{=}$ tangent at — in (t_1, y_1).

⑦ Can we solve for y_{k+1}? $\quad h_k := t_{k+1} - t_k$

(IE) $\Rightarrow \quad y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$

y_{k+1} solves: $G(h_k, y_{k+1}) = 0$, $\quad G(h,y) = y - y_k - h f(y)$

[For autonomous ODE, $f \in C^1$]

Does $y \to G(h_k, y)$ have a unique zero?

- $G(0, y_k) = 0$
- $D_y G(0, y) = I \to$ invertible

Implicit function theorem: for small $|h|$: $G(h,y) = 0$ defines a function $h \mapsto y(h)$: $G(h, y(h)) = 0$

$\to$ y_{k+1} exists for sufficiently small h.

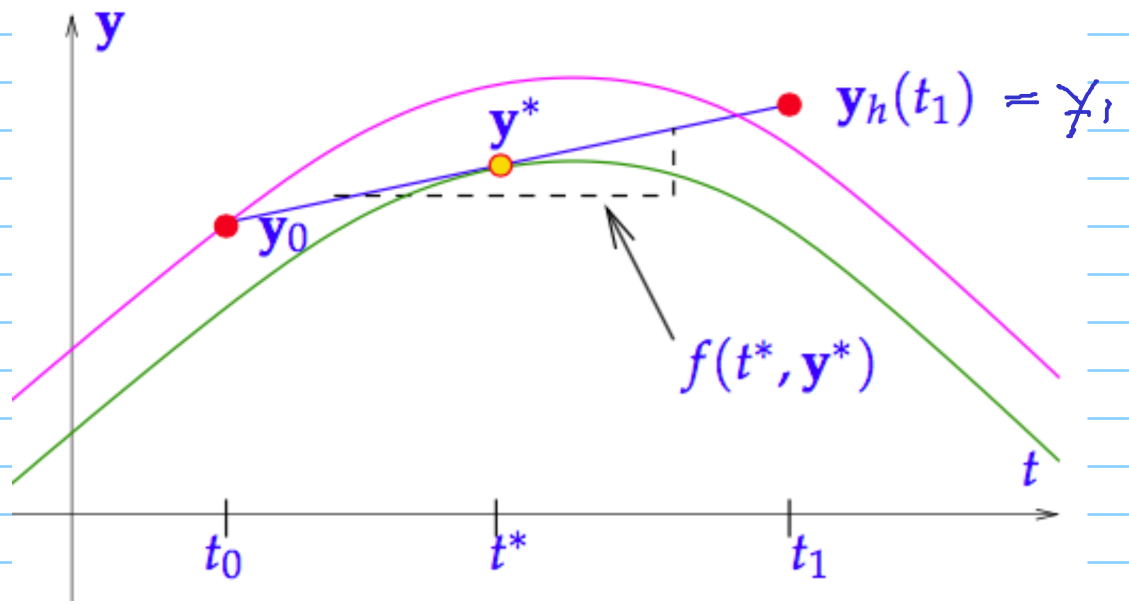
$\to$ $y_0, y_1, \dots, y_N$ well defined on sufficiently fine meshes.

II.2.3. Implicit midpoint method

$$\cancel{f\left(\frac{t_{k+1}+t_k}{2}, y\left(\frac{t_{k+1}+t_k}{2}\right)\right)}$$

$$\approx$$

$$\underbrace{\frac{y_{k+1} - y_k}{t_{k+1} - t_k}}_{\text{Difference quotient}} = \underbrace{f\left(\frac{t_{k+1}+t_k}{2}, \frac{y_{k+1}+y_k}{2}\right)}_{\text{polygonal approximation}}$$



(IMP) $\quad y_{k+1} = y_k + h_k f\left(\frac{t_k + t_{k+1}}{2}, \frac{y_k + y_{k+1}}{2}\right)$

All three polygonal methods (for sufficiently small h_k) define recursions

$$y_{k+1} = \Psi(h_k, y_k)$$

$\to$ Sequence $y_0, y_1, \dots, y_N$ (timestepping)

Hope: $\quad y_k \approx y(t_k)$

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

$\hookrightarrow$ evolution map

11.3. General Single Step Methods (SSM)

Euler methods / implicit midpoint method for IVP $\dot{y} = f(y)$, $y(0) = y_0$:

→ recursion $\quad y_{k+1} = \Psi(h_k, y_k)$ ↳ discrete evolution

$\approx \updownarrow$

$y(t_{k+1}) = \Phi(h, y(t_k))$ ← ↳ evolution map

Goal: $\Psi \approx \Phi$

Definition 11.3.5. Single step method (for autonomous ODE) → [45, Def. 11.2]

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

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

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

Notation: $\quad y_{k+1} = \Psi^{h_k} y_k \quad , \quad h_k := t_{k+1} - t_k$

SSM can also be defined through first step: $y_1 = \Psi^h y_0$

Consistent discrete evolution

The discrete evolution $\boldsymbol{\Psi}$ 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

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

Consistent SSM look like explicit Euler

Expl. Euler: $\Psi^h y = y + h f(y)$

Impl. midpoint method:

$$\Psi^h y_0 := y_1 = y_0 + h f(\tfrac{1}{2}(y_0 + y_1))$$
$$= y_0 + h \underbrace{f(y_0 + \tfrac{1}{2} h f(\ldots))}_{=: \psi(h,y)}$$

Impl. fct. thm: $y_1(h, y_0)$ is continuous ⇒ ψ continuous

$\psi(0, y_0) = f(y_0)$

Convergence *

Setting: $\dot{y} = f(y)$, $y(0) = y_0$

Assume: $\|f(z) - f(w)\| \le L\|z - w\| \quad \forall z, w \in D$

Error measure: $\max_j \|y_j - y(t_j)\|$

sequence produced by SSM $\qquad$ exact solution

* need family of meshes $(\mathcal{M}_\ell)_\ell$: $h_\ell := \max_j |t^\ell_{j+1} - t^\ell_j| \to 0$ as $\ell \to \infty$

h-refinement $\qquad$ meshwidth

(9)

Example: $\dot{y} = \lambda y(1-y)$, $y(0) = 0.01$

Final time $T = 1 \to h = \frac{1}{N}$, $N = \#\mathcal{M}$

Measured: $|y_N - y(T)|$

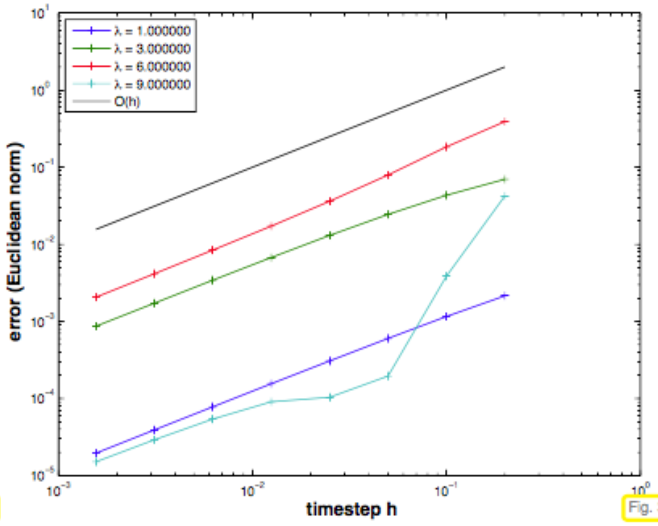
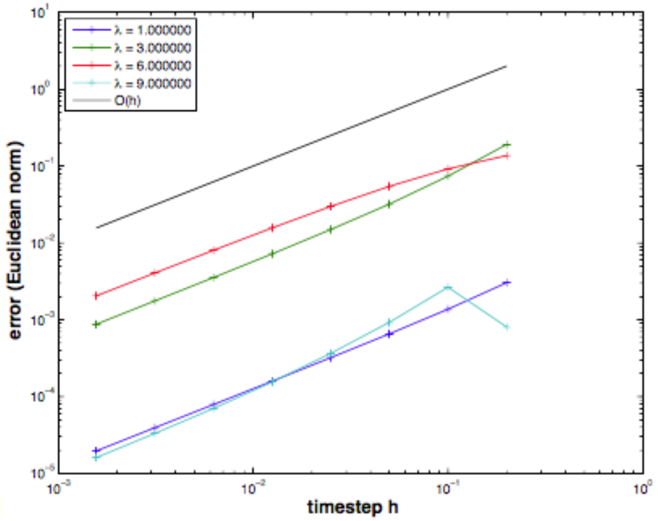


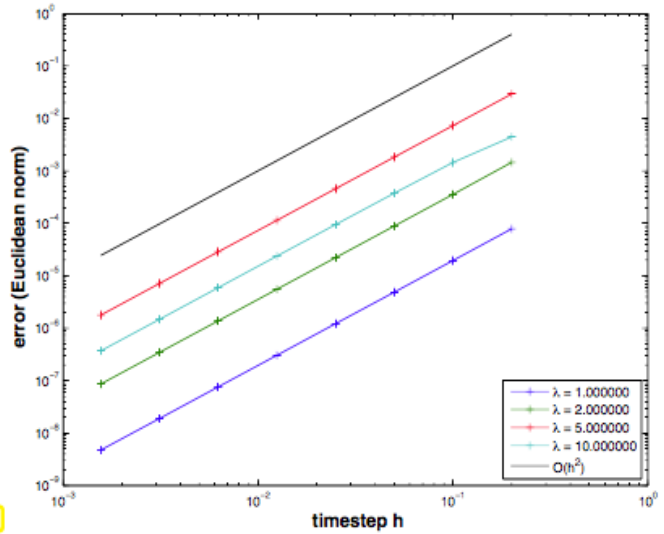
Fig. 335

explicit Euler method



implicit Euler method

$\to$ algebraic cvg. for $h \to 0$: $O(h)$ "first order"



◁ Implicit midpoint method $O(h^2)$

In general* consistent SSM converge algebraically for meshwidth $h \to 0$: error $= O(h^p)$, $p \in \mathbb{N}$

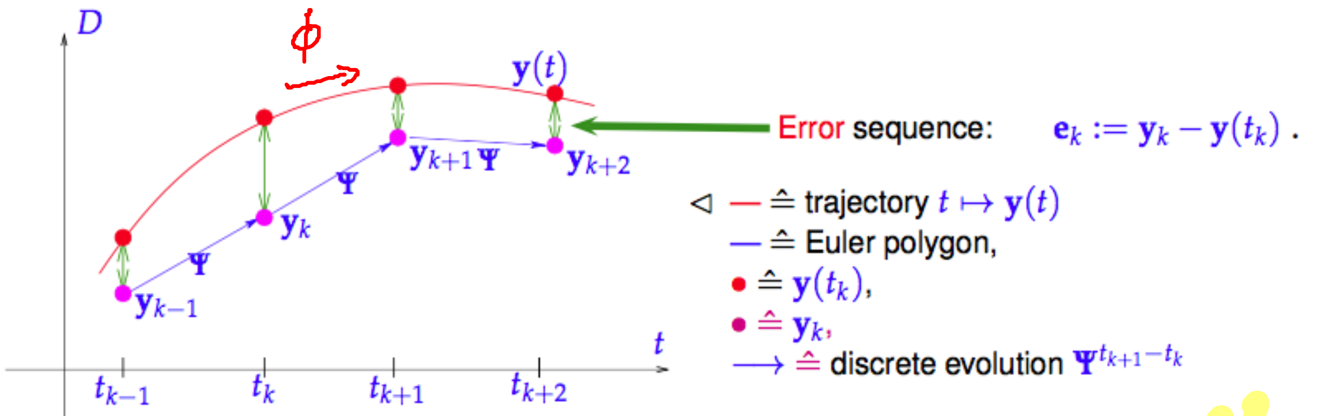
$p \mathrel{\hat{=}}$ order of SSM

* for smooth trajectories

Analysis for expl. Euler (for smooth $f \to$ smooth $y(t)$)

Assume L.C.: $\|f(z) - f(w)\| \le L \|z - w\| \quad \forall z, w \in D$

Expl. Euler: $\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k)$, $k = 1, \dots, N-1$. (11.2.7)

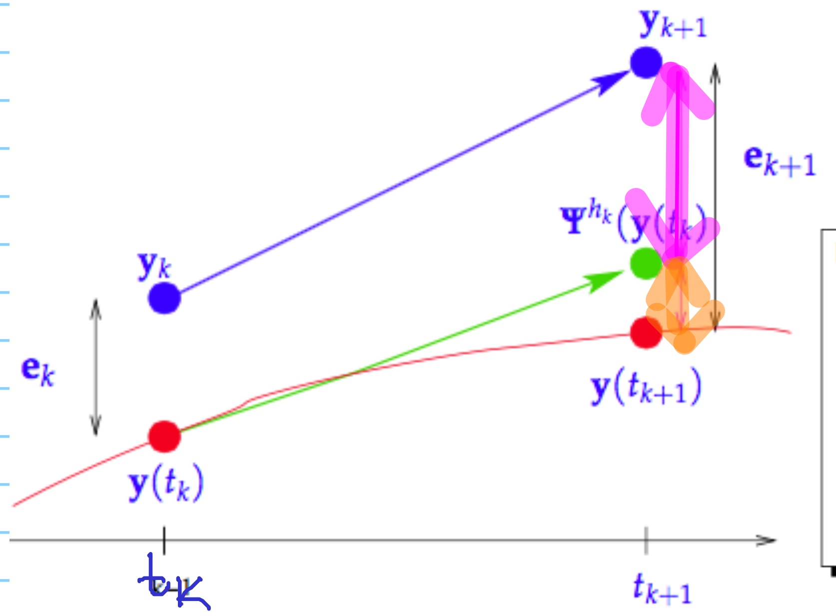


Discrete evol. $\Psi^h y = y + h f(y)$

Key idea: Error splitting

$e_{k+1} = \Psi^{h_k} y_k - \Phi^{h_k} y(t_k)$

$= \left(\Psi^{h_k} y_k - \Psi^{h_k} y(t_k)\right) + \left(\Psi^{h_k} y(t_k) - \Phi^{h_k} y(t_k)\right)$



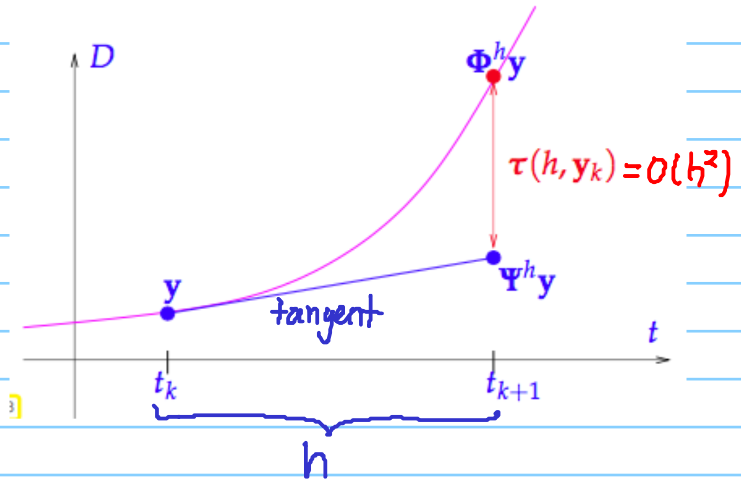
$\hat{=}$ propagated error

$\hat{=}$ one-step error

Fundamental error splitting

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

- One-step error :



Taylor :

$$y(t_k+h) =$$

$$= y(t_k) + \dot{y}(t_k)h + \tfrac{1}{2}h^2\ddot{y}(\xi)$$

$$= \Psi^h y(t_k) + \tfrac{1}{2}h^2 \ddot{y}(\xi)$$

$$t_k < \xi < t_{k+1}$$

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

- Propagated error :

$$\| \Psi^h(y(t_k)) - \Psi^h y_k \| = \| e_k + h(f(y(t_k)) - f(y_k)) \|$$

+ Lipschitz cont. $\leq \|e_k\| + hL\|e_k\|$

Combined $\Longrightarrow$ error recursion for $\varepsilon_k := \|e_k\|$

$$\varepsilon_{k+1} \leq (1+h_k L)\varepsilon_k + \delta_k \;, \quad \delta_k := \tfrac{1}{2}h_k^2 \ddot{y}(\xi_k)$$

$$\varepsilon_0 = 0$$

$$\longrightarrow \quad \varepsilon_k \leq \sum_{\ell=1}^{k} \prod_{j=1}^{\ell-1} (1+h_j L)\, \delta_\ell$$

$[\, 1+x \leq e^x \,]$ $\quad \leq \sum_{\ell=1}^{k} \left(\prod_{j=1}^{\ell-1} e^{h_j L} \right) \delta_\ell$

$$\leq \sum_{\ell=1}^{k} e^{(\underbrace{\sum_{j=1}^{\ell-1} h_j}_{\leq T})L} \cdot \delta_\ell$$

$$\leq e^{TL} \|\ddot{y}\|_{L^\infty([0,T])} \sum_{\ell=1}^{k} h_\ell \,(\tfrac{1}{2}\underbrace{h_\ell}_{\leq h})$$

for all $k=1,..,N$: $\leq \underbrace{e^{TL} \|\ddot{y}\|_{L^\infty([0,T])}\, h\, T}_{O(h)}$

Note: One-step-error $O(h^2)$ $\longleftrightarrow$ Total error $O(h)$

For SSM : One-step-error $O(h^{p+1})$ $\longrightarrow$ Order p

11.4. Explicit Runge-Kutta SSM

Goal: Explicit SSM with order $p > 1$

Rationale for high order: unknown constant

Discretization error $\approx C h^p$ [assume sharp]

Goal: error reduction by factor $\rho > 1$ by h-refinement

$$\frac{C h_0^p}{C h_1^p} = \rho \Rightarrow h_1 = \rho^{-1/p} \cdot h_0$$

Effort $\sim$ #{timesteps} $\sim h^{-1}$:

$$W_1 = \rho^{1/p} \cdot W_0$$

effort after refinement — effort before refinement

Less additional effort to achieve prescribed error reduction the larger p.

Construction

$$\left.\begin{array}{l} \dot{y} = f(y) \\ y(0) = y_0 \end{array}\right\} \Rightarrow y(h) = y_0 + \int_0^h f(y(\tau))\,d\tau$$

$$y_1 = y_0 + h \sum_{i=1}^{s} b_i f(y(c_i h))$$

Idea: Quadrature! s-pt. Q.F. on [0,1], weights b_i, nodes c_i

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

Idea: approximate the integral by means of s-point quadrature formula (→ Section 5.1, defined on the reference interval $[0,1]$) with nodes $c_1, \dots, c_s$, weights $b_1, \dots, b_s$.

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

Obtain these values by bootstrapping

Goal: One-step-error $O(h^{p+1})$

→ sufficient to approximate $y(t_0 + c_i h)$ with error $O(h^p)$, because of multiplication w/ h!

⇒ $y(t_0 + c_i h)$ approximate by SSM of order $p-1$ [Start with expl. Euler]

Examples:

- Quadrature formula = trapezoidal rule (5.2.5):

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

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

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

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

- Quadrature formula → simplest Gauss quadrature formula = midpoint rule (→ Ex. 5.2.3) & $\mathbf{y}(\frac{1}{2}(t_1+t_0))$ approximated by explicit Euler step (11.2.7)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0)\,, \quad \mathbf{k}_2 = \mathbf{f}(t_0 + \tfrac{h}{2}, \mathbf{y}_0 + \tfrac{h}{2}\mathbf{k}_1)\,, \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}_2\,. \qquad (11.4.7)$$

(11.4.7) = explicit midpoint method (for numerical integration of ODEs) [10, Alg. 11.18].

Definition 11.4.9. Explicit Runge-Kutta method

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i,j = 1,\dots,s$, $s \in \mathbb{N}$, 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 ($\mathbf{y}_0 \in D$)

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

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

Effort for one step : s f-eval.

Butcher scheme

Shorthand notation for (explicit) Runge-Kutta methods [10, (11.75)]

Butcher scheme ▷

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

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

$b_i \triangleq$ quadrature weights *

* Apply RK-SSM to $\dot{y} = f(t)$

Remark: RK-SSM consistent, if $\sum_{i=1}^{s} b_i = 1$

Examples

- Explicit Euler method (11.2.7): $\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$ ➤ order = 1
- explicit trapezoidal rule (11.4.6): $\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$ ➤ order = 2
- explicit midpoint rule (11.4.7): $\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$ ➤ 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

Construction high-order RK-SSM by solving order conditions

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

Theory : $p \leq s$

```
[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 $\mathbf{y}_0 \in \mathbb{R}^d$

:

- `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)

```
function varargout = ode45(ode,tspan,y0,options,varargin)
% Processing of input parameters omitted
% :
% Initialize method parameters, c.f. Butcher scheme (11.4.11)
pow = 1/5;
A = [1/5, 3/10, 4/5, 8/9, 1, 1];
B = [
    1/5         3/40     44/45     19372/6561     9017/3168        35/384
    0           9/40     -56/15    -25360/2187    -355/33          0
    0           0        32/9      64448/6561     46732/5247
        500/1113
    0           0        0         -212/729       49/176           125/192
    0           0        0         0              -5103/18656
        -2187/6784
    0           0        0         0              0                11/84
    0           0        0         0              0                0
    ];
E = [71/57600; 0; -71/16695; 71/1920; -17253/339200; 22/525; -1/40];
% : (choice of stepsize and main loop omitted)
% ADVANCING ONE STEP.
hA = h * A;
hB = h * B;
f(:,2) = feval(odeFcn,t+hA(1),y+f*hB(:,1),odeArgs{:});
f(:,3) = feval(odeFcn,t+hA(2),y+f*hB(:,2),odeArgs{:});
f(:,4) = feval(odeFcn,t+hA(3),y+f*hB(:,3),odeArgs{:});
f(:,5) = feval(odeFcn,t+hA(4),y+f*hB(:,4),odeArgs{:});
f(:,6) = feval(odeFcn,t+hA(5),y+f*hB(:,5),odeArgs{:});

tnew = t + hA(6);
if done, tnew = tfinal; end    % Hit end point exactly.
h = tnew - t;          % Purify h.
ynew = y + f*hB(:,6);
% : (stepsize control, see Sect. 11.5 dropped
```

Butcher scheme

Compute k_i

$\Rightarrow$ Linear comb. of k_i

11.5. Adaptive Stepsize Control

Example: Oscillatory chemical reaction

BZ-reaction

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

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

Abrupt changes in concentrations over many orders of magnitude

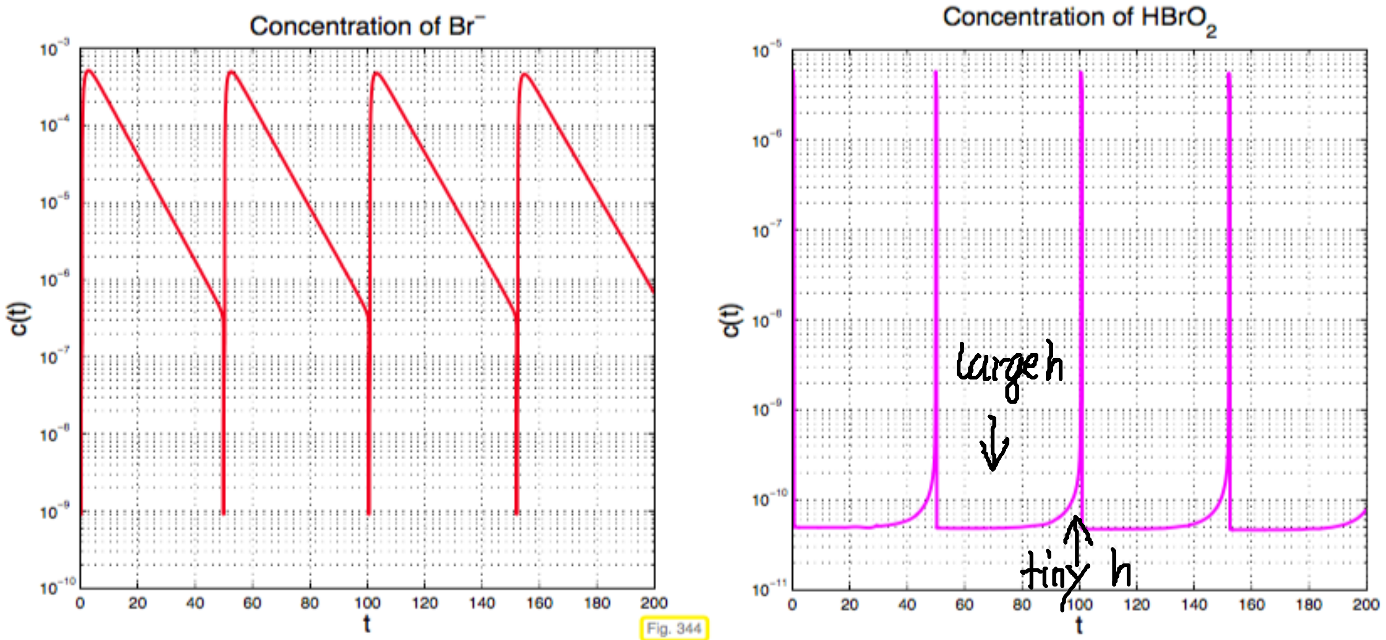


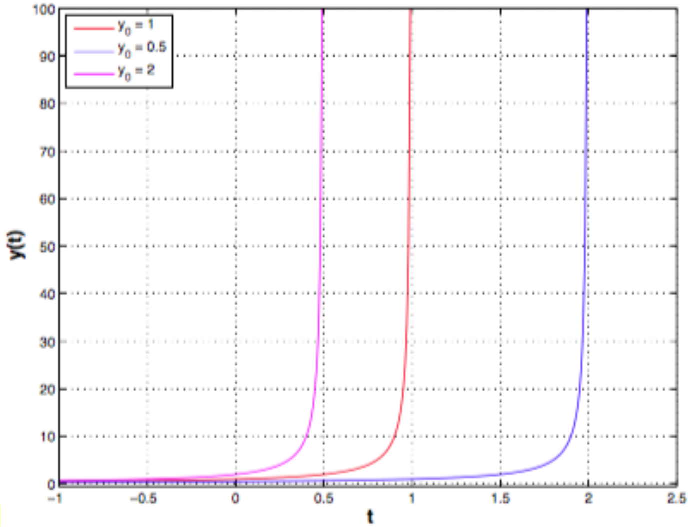
Fig. 344

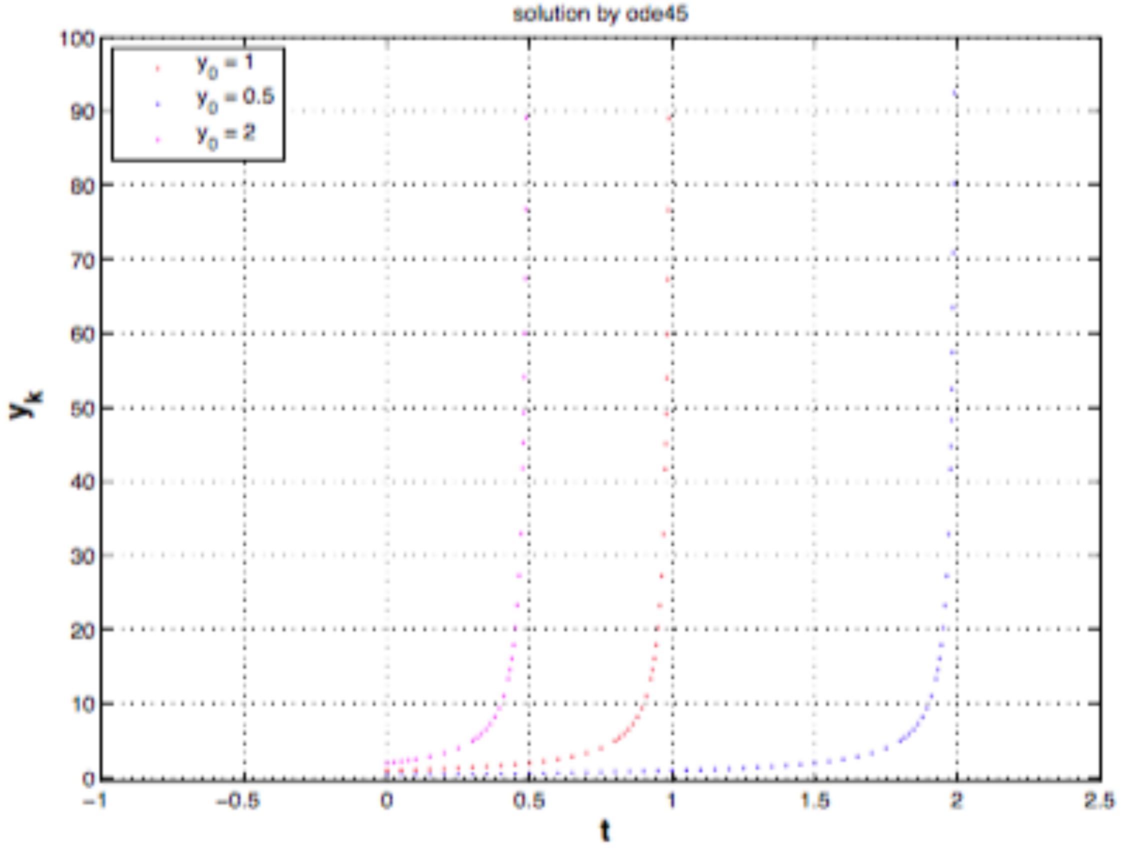
Example: blow-up

$$\dot{y} = y^2$$

blow-up after finite time ▷

impossible to solve with fixed timestep





◁ adaptive timestepping detects blow-up

Policy: (cmp. adaptive composite quadrature)

Be efficient! Be accurate!

Stepsize adaptation for single step methods

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

[Dream]

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})

Choose h_k based on y_{k-1} and h_{k-1} alone

Cheap & easy to implement

(I) ESTIMATE of one-step error

Idea: *Estimation of one-step error*

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

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

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

Heuristics for concrete h

(II) REFINE

$\mathrm{EST}_k \le \max(\mathrm{atol}, \mathrm{rtol}\cdot\|y_k\|)$?

Accept step, goto next step with $h_{k+1} = \alpha h_k$ for some $\alpha > 1$

Reject step, repeat with $h_k \leftarrow \frac{1}{2} h_k$

MATLAB-code 11.5.11: Simple local stepsize control for single step methods

```
function [t,y] =
  odeintadapt(Psilow,Psihigh,T,y0,h0,reltol,abstol,hmin)
t = 0; y = y0; h = h0;                %
while ((t(end) < T) && (h > hmin)) %
  yh = Psihigh(h,y0);    % high order discrete evolution Ψ̃^h
  yH = Psilow(h,y0);     % low order discrete evolution Ψ^h
  est = norm(yH-yh);     % ↔ EST_k

  if (est < max(reltol*norm(y0),abstol))                %
    y0 = yh; y = [y,y0]; t = [t,t(end) + min(T-t(end),h)]; %
    h = 1.1*h;  % step accepted, try with increased stepsize
  else, h = h/2; end   % step rejected, try with half the stepsize
end
```

ESTIMATE

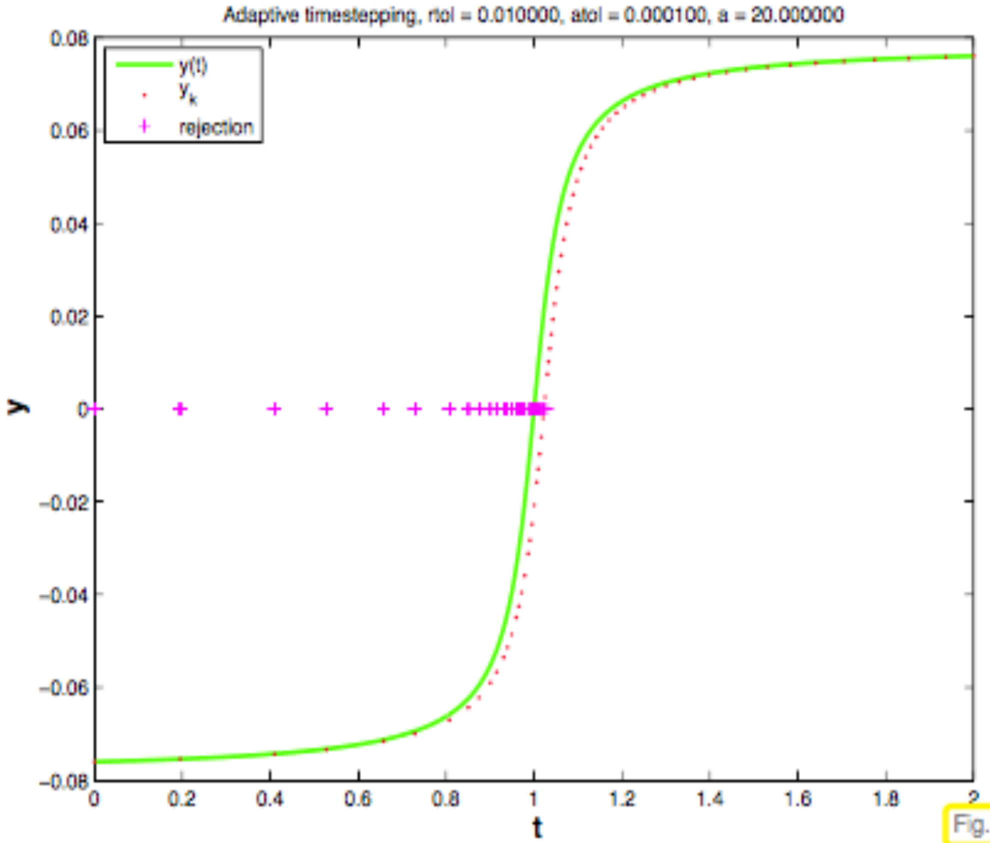
EST_k has next to nothing to do with $y(t_k) - y_k$!

$\hookrightarrow$ only $\approx$ one-step error

No global error control through local-in-time adaptive timestepping

The absolute/relative tolerances imposed for local-in-time adaptive timestepping do *not* allow to predict accuracy of solution!

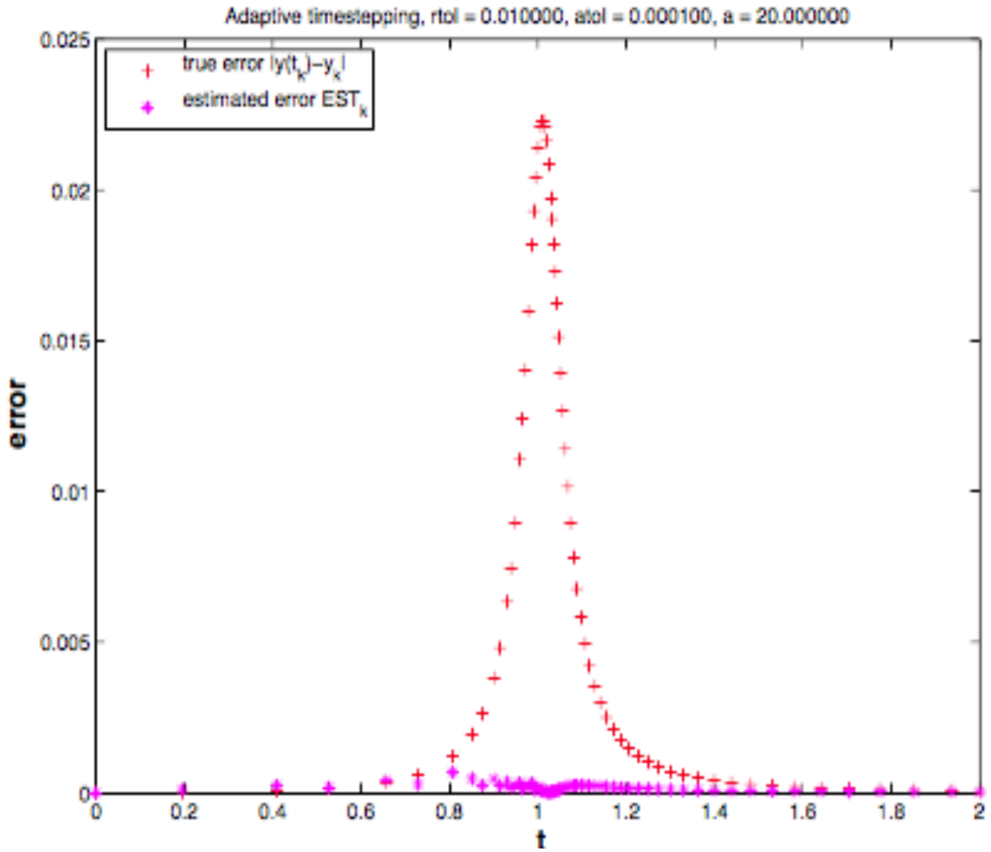
Example: $\dot{y} = \cos(\alpha y)$, Code 11.5.11 w/ expl. Eul. (order 1) & expl. trp. meth. (order 2)



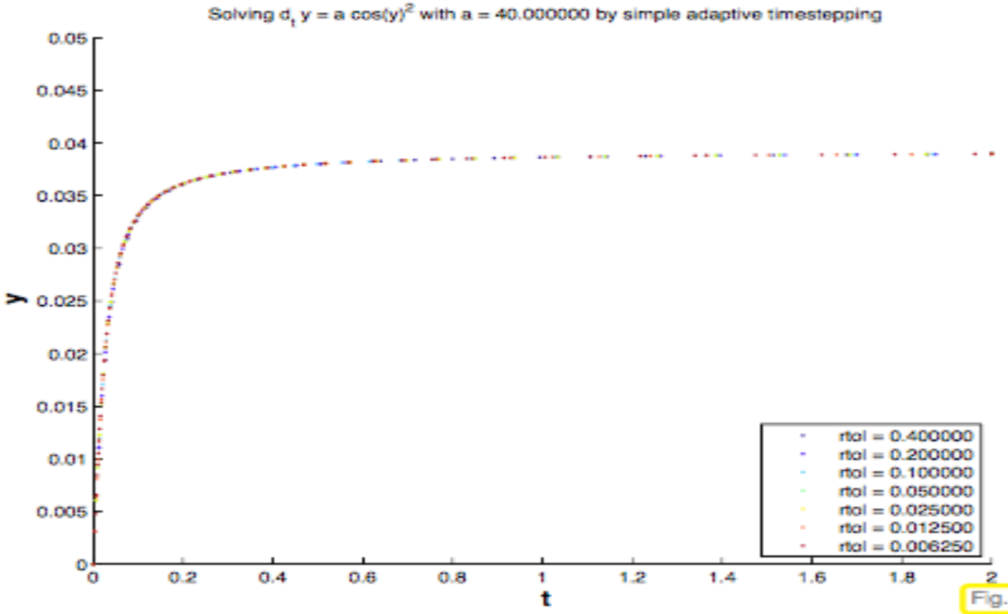
7

Fig. 3

$\lhd$ ASC works !



◁ $EST_k \not\approx$ true error at t_k

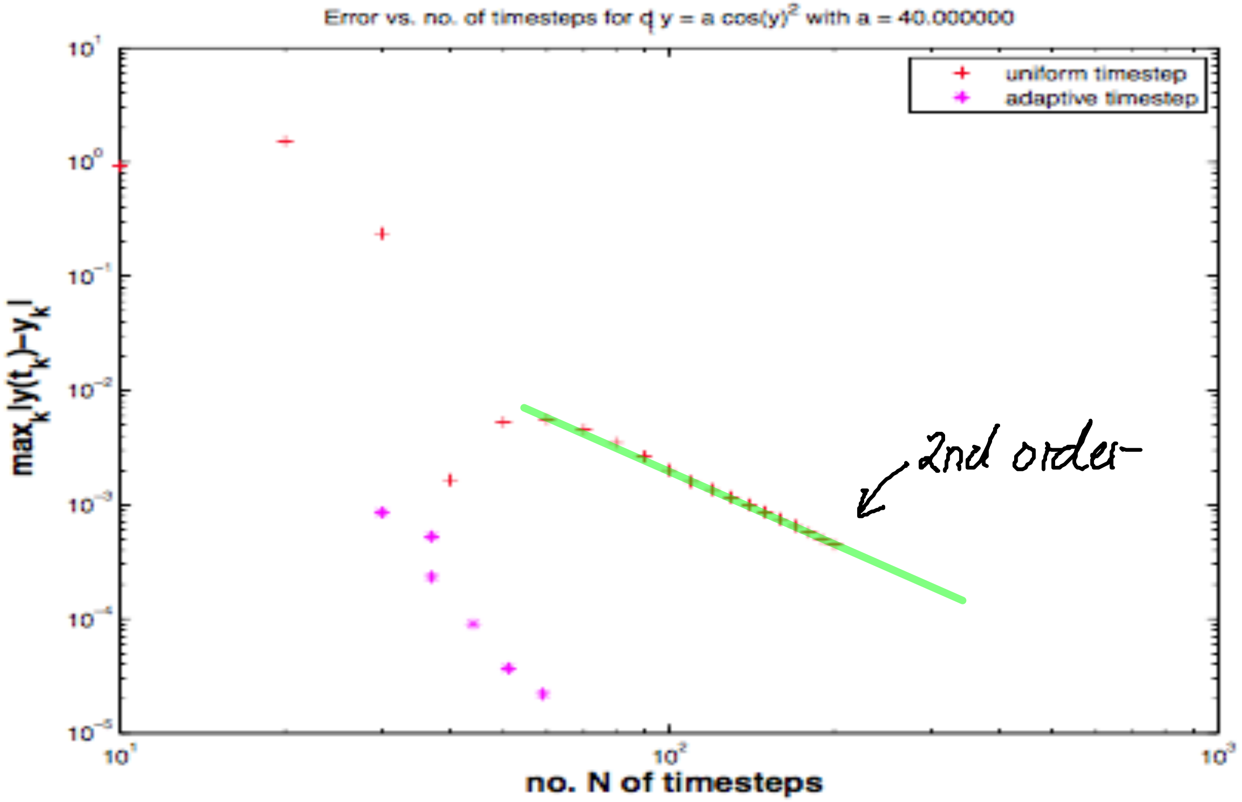


$y_0 = 0$
→ no step

349 Fig. 350

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

Gain through adaptivity



Error vs. computational effort

→ Adaptivity efficient !

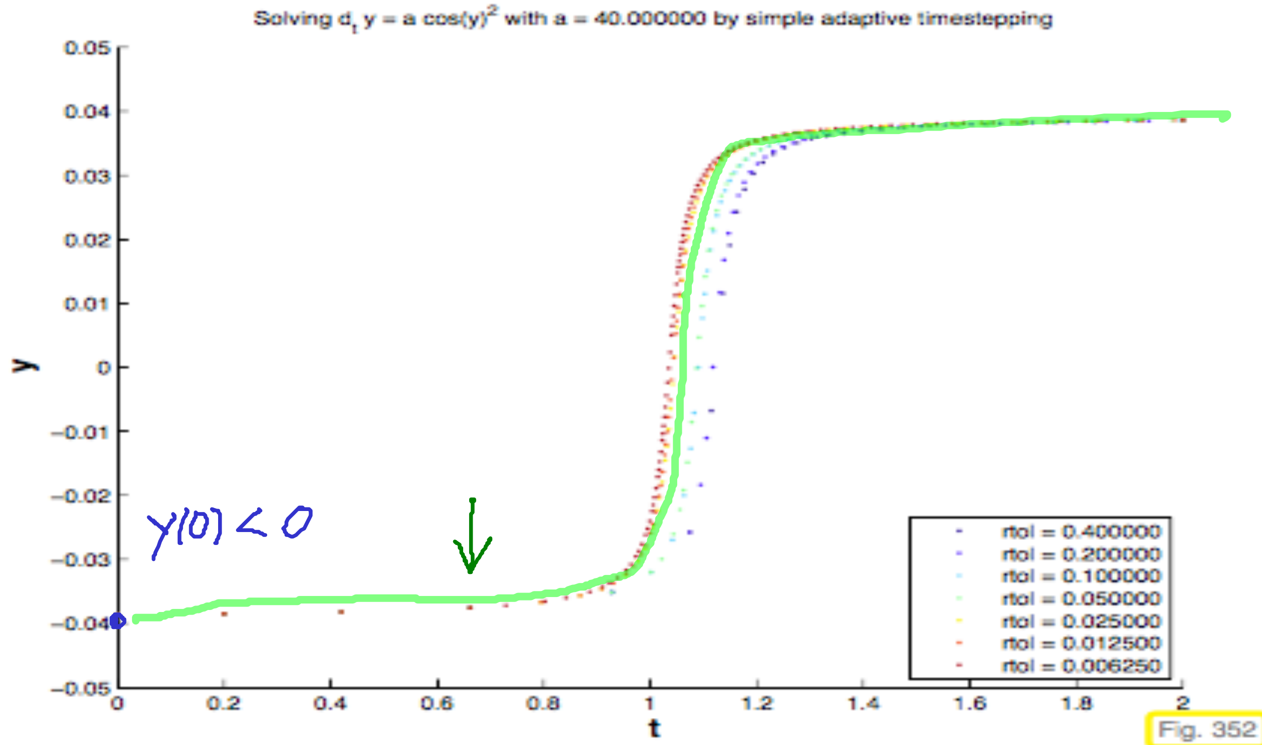
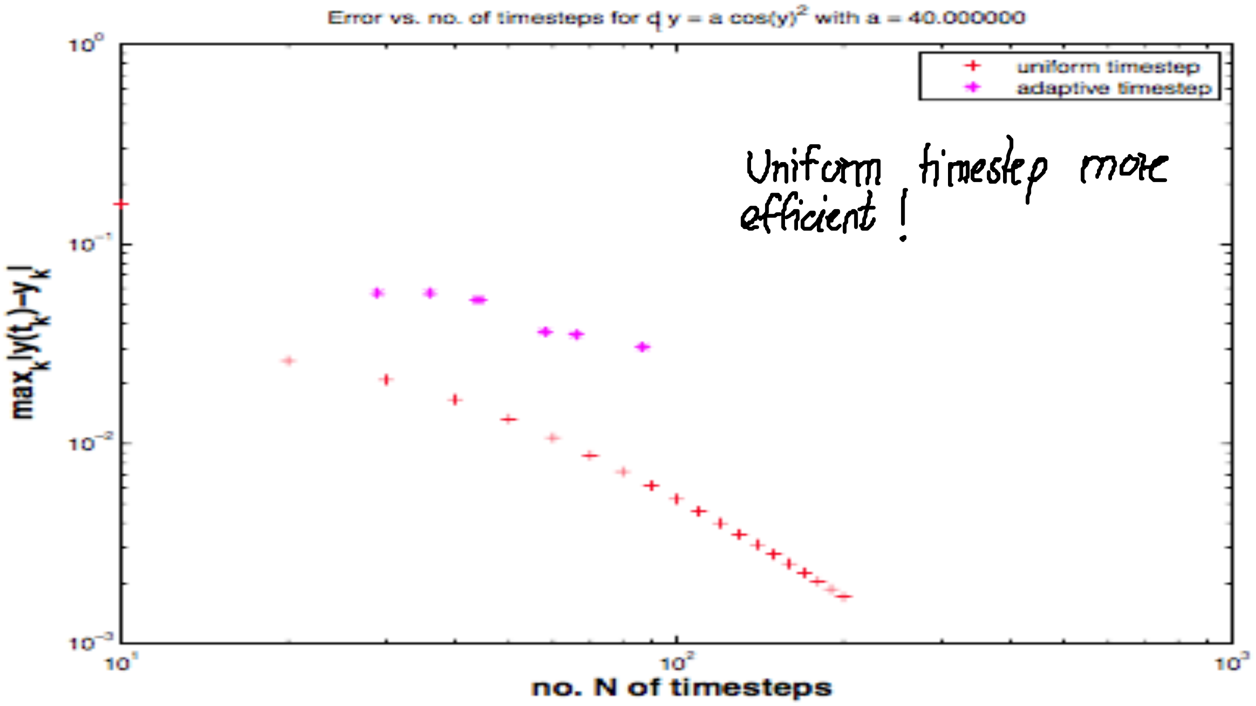


Fig. 352



Error vs. computational effort

Here: sensitive dependence of step position on $y(t)$ for small times

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 $h_{k+1} = ?$

$$\mathrm{EST}_k := \| \tilde{\Psi}^h y_k - \Psi^h y_k \| , \quad h := h_k$$

order p+1   order p

One-step errors:

$$\Rightarrow \quad \| \Psi^h y_k - \Phi^h y_k \| = c h^{p+1} + O(h^{p+2})$$
$$\| \tilde{\Psi}^h y_k - \Phi^h y_k \| = O(h^{p+2})$$

known   exact solution

$$\Rightarrow \quad \mathrm{EST}_k \approx c h^{p+1} \quad \text{for small } h$$

Goal (efficiency!) $\quad \mathrm{EST}_k = \mathrm{TOL}$

$$h_{new} : \quad c h_{new}^{p+1} = \mathrm{TOL} \qquad (*)$$

$$EST_k \approx c h^{p+1} \Longrightarrow c \approx \frac{EST}{h^{p+1}}$$

$$(*) \Longrightarrow \boxed{h_{new} = h \cdot \sqrt[p+1]{\frac{TOL}{EST}}}$$

recommended new timestep

MATLAB-code 11.5.22: Refined local stepsize control for single step methods

```
function [t,y] =
   odeintssctrl(Psilow,p,Psihigh,T,y0,h0,reltol,abstol,hmin)
t = 0; y = y0; h = h0;                    %
while ((t(end) < T) && (h > hmin)) %
  yh = Psihigh(h,y0);   % high order discrete evolution Ψ̃^h  ⎫
  yH = Psilow(h,y0);    % low order discrete evolution Ψ^h   ⎬ ESTIMATE
  est = norm(yH-yh);    % ↔ EST_k                             ⎭

  tol = max(reltol*norm(y(:,end)),abstol); %
  h = h*max(0.5,*min(2,*(tol/est)^(1/(p+1)))); % Optimal stepsize
     according to (11.5.21)
  if (est < tol)                                          %
    y0 = yh; y = [y,y0]; t = [t,t(end) + min(T-t(end),h)]; % step
      accepted
  end
end
```

* safeguard against oscillating timesteps

Implementation $\hat{=}$ embedded RK methods (same coeffs. a_{ij}, different b_i for different orders)