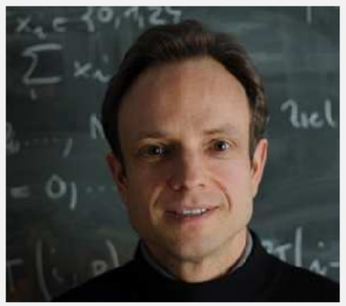
ETH Lecture 401-0674-00L Numerical Methods for Differential Equations

Course Video
Section 6.5: Adaptive Stepsize Control

Prof. R. Hiptmair, SAM, ETH Zurich

Date: March 27, 2021



Prerequisites.
- Knowledge about adaptive numerical quadrature

Dependencies. [Lecture → Section 6.4]

Duration: minutes

Video and accompanying tablet notes may not match completely!

[Corrections and updates may have been made in tablet notes.]

VI. Numerical Integration – Single-Step Methods

6.5. Adaptive Stepsize Control

6.5.1. The Need for Timestep Adaptation

Ex. 6.5.1.1. (Oregonator reaction)

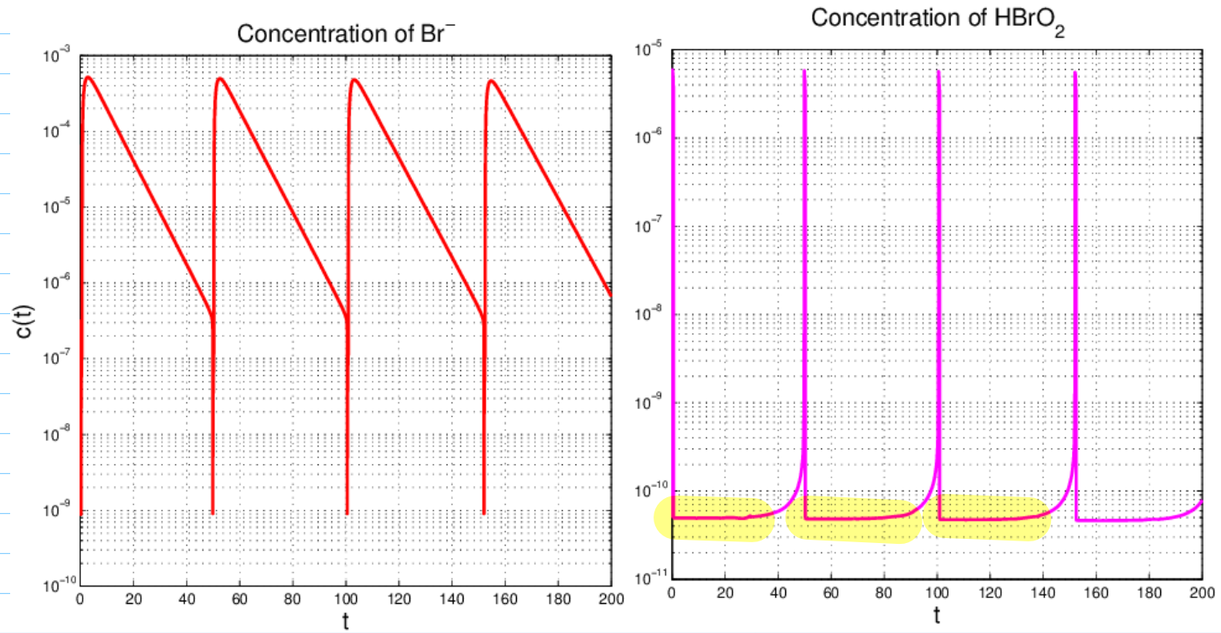
This is a special case of an "oscillating" Zhabotinski-Belousov reaction :

$$\begin{aligned} \mathrm{BrO}_3^- + \mathrm{Br}^- &\mapsto \mathrm{HBrO}_2 \\ \mathrm{HBrO}_2 + \mathrm{Br}^- &\mapsto \mathrm{Org} \\ \mathrm{BrO}_3^- + \mathrm{HBrO}_2 &\mapsto 2\,\mathrm{HBrO}_2 + \mathrm{Ce(IV)} \\ 2\,\mathrm{HBrO}_2 &\mapsto \mathrm{Org} \\ \mathrm{Ce(IV)} &\mapsto \mathrm{Br}^- \end{aligned} \tag{6.5.0.2}$$

ODE model for reaction kinetics :

$$\begin{aligned} y_1 := c(\mathrm{BrO}_3^-): \quad \dot{y}_1 &= -k_1 y_1 y_2 - k_3 y_1 y_3 \,, \\ y_2 := c(\mathrm{Br}^-): \quad \dot{y}_2 &= -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5 \,, \\ y_3 := c(\mathrm{HBrO}_2): \quad \dot{y}_3 &= k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2 \,, \\ y_4 := c(\mathrm{Org}): \quad \dot{y}_4 &= k_2 y_2 y_3 + k_4 y_3^2 \,, \\ y_5 := c(\mathrm{Ce(IV)}): \quad \dot{y}_5 &= k_3 y_1 y_3 - k_5 y_5 \,, \end{aligned} \tag{6.5.0.3}$$

(2)



▷ Solution highly non-uniform in time

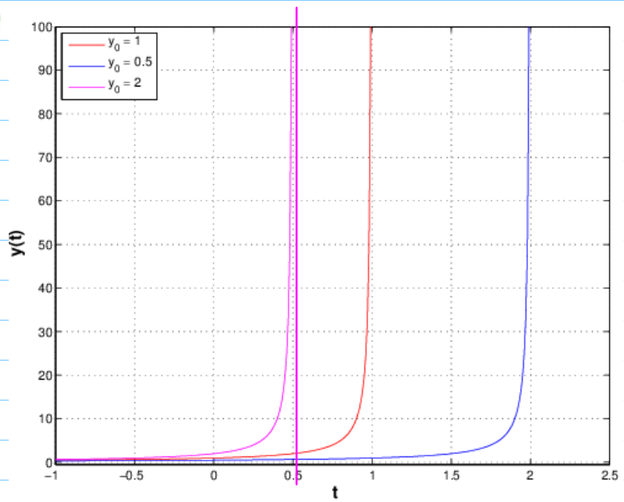
▷ Use highly non-uniform temporal mesh

Ex. 6.5.1.4 (Explosion ODE)

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

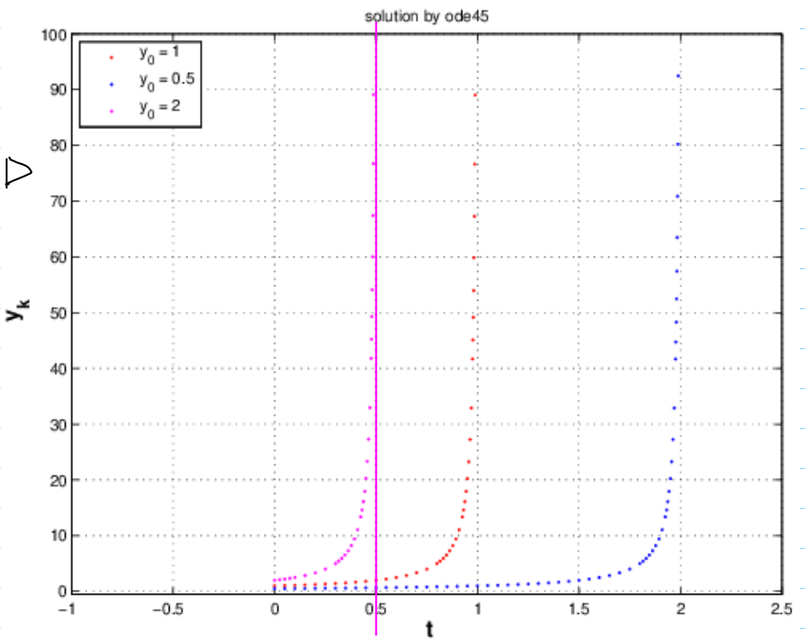
$$y(0) = y_0 > 0$$

▷ Finite-time blow-up at $t = 1/y_0$



Sneak preview

(y_k) by adaptive explicit RK-SSM ▷



③

6.5.2 Local-in-time Stepsize Control

≙ Algorithm building temporal mesh $\mathcal{M} = \{t_k\}_{k=0}^{M}$ during execution of SSM

[similar to adaptive quadrature]

Principles: efficient & reliable

Stepsize adaptation for single step methods

[Dream]

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

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

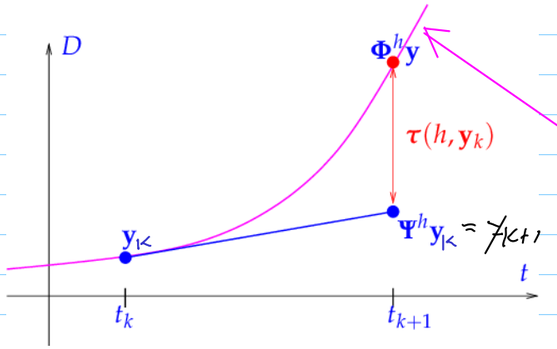
Tool: Local-in-time one-step error estimator (*a posteriori*, based on $\mathbf{y}_k, h_{k-1}$)

$= \Psi^{h_k} y_k - \Phi^{h_k} y_k$

not accessible !

can only be estimated (also locally-in-time)

Choose t_{k+1} only based on t_k, y_k, h_k (at $O(1)$ cost/step)



△ One-step error

Exact solution of IVP

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

§ 6.5.2.2. (Local-in-time error estimation)

Idea: *Estimation of one-step error*

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

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

$$\underbrace{\Phi^h y_k}_{\text{one-step error}} - \Psi^h y_k \approx \mathrm{EST}_k := \tilde{\Psi}^h y_k - \Psi^h y_k . \qquad (6.5.2.3)$$

↑ replaces exact solution

Heuristics for concrete $h > 0$

computationally available

(4) § 6.5.2.4. (Local timestep adaptation)

Available : EST_k for timestep size h

abs. tol / rel. tol. [user supplied]

Compare: $\mathrm{EST}_k \lessgtr \mathrm{TOL} := \max\{\mathrm{ATOL}, \mathrm{RTOL}\cdot\|y_k\|\}$

YES → Accept step

Advance to next step

($h \leftarrow 1.1\cdot h$)

NO → Reject stepsize

Repeat w/ $h \leftarrow \frac{1}{2}h$

"grinding to halt"

→ e.g. blow-up detected

C++11 code 6.5.2.6: Simple local stepsize control for single step methods ⇢ GITHUB

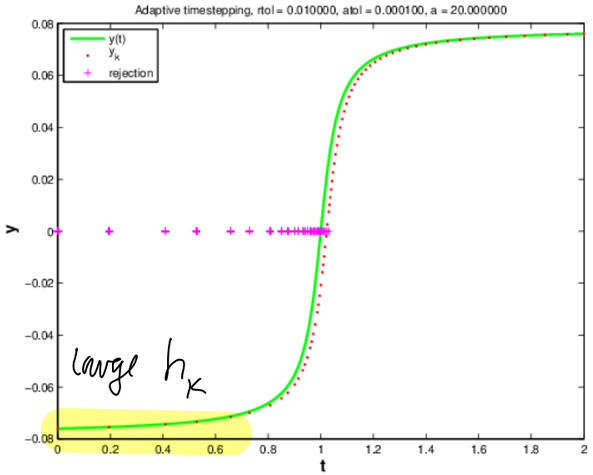
```
2  // Auxiliary function:  default norm for an EIGEN vector type
3  template <class State>
4  double _norm(const State &y) { return y.norm(); }
5
6  // Adaptive numerical integrator based on local-in-time stepsize control
7  template <class DiscEvolOp, class State, class NormFunc = decltype(_norm<State>)>
8  std::vector<std::pair<double, State> >
9  odeintadapt(DiscEvolOp &&Psilow, DiscEvolOp &&Psihigh,
10             const State &y0,double T, double h0,
11             double reltol, double abstol, double hmin,
12             NormFunc &norm = _norm<State>) {
13   double t = 0; // initial time t_0 = 0
14   State y = y0; // current state
15   double h = h0; // timestep to start with
16   std::vector<std::pair<double,State>> states; // vector of times/computed
         states:  (t_k, y_k)_k
17   states.push_back({t, y}); // initial time and state
18
19   while ((states.back().first < T) && (h >= hmin))  { //
20     State yh = Psihigh(h, y); // high order discrete evolution Ψ̃^h
21     State yH = Psilow(h, y); // low order discrete evolution Ψ^h
22     double est = norm(yH-yh); // local error estimate EST_k
23
24     if (est < std::max(reltol*norm(y), abstol)) { // step accepted
25       y = yh; // use high order approximation
26       t = t + std::min(T-t,h); // next time t_k
27       states.push_back({t,y}); //
28       h = 1.1*h;  // try with increased stepsize
29     }
30     else { // step rejected
31       h = h/2; // try with half the stepsize
32     }
33     // Numerical integration has ground to a halt !
34     if (h < hmin) {
35       std::cerr << "Warning: Failure at t=" << states.back().first
36         << ". Unable to meet integration tolerances without reducing the step "
37         << "size below the smallest value allowed (" << hmin << ") at time t."
38       << std::endl;
39     }
40   }
41   return states;
42 }
```

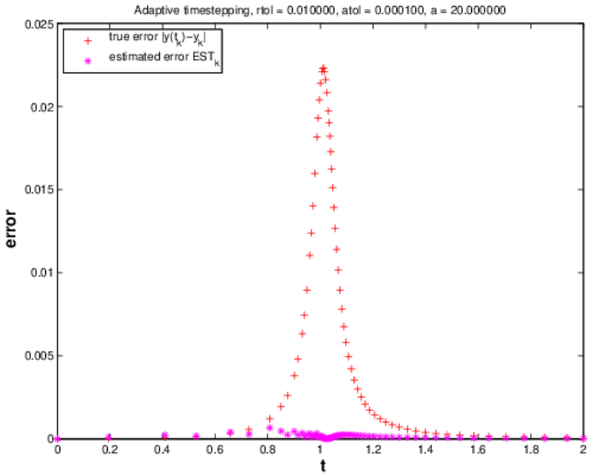
Accept

⑤

Exp. 6.5.2.8 (Simple adaptive stepsize control)

- Scalar IVP: $\dot{y} = \cos(\alpha y)^2$, $y(0) < 0$, $\alpha = 20$
- $\Psi^h \leftrightarrow$ expl. Euler, $\widetilde{\Psi}^h \leftrightarrow$ exp. trapezoidal method
  (order = 1) (order = 2)





Statistics: 66 timesteps, 131 rejected timesteps

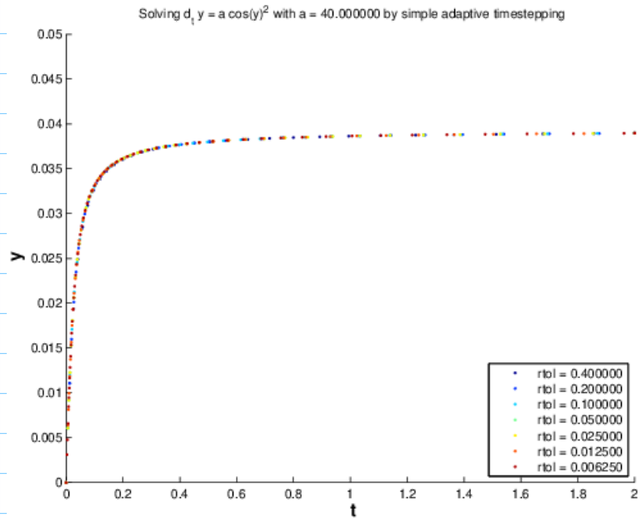
↓

Estimated error EST_k and true error $|y(t_k) - y_k|$ poorly correlated

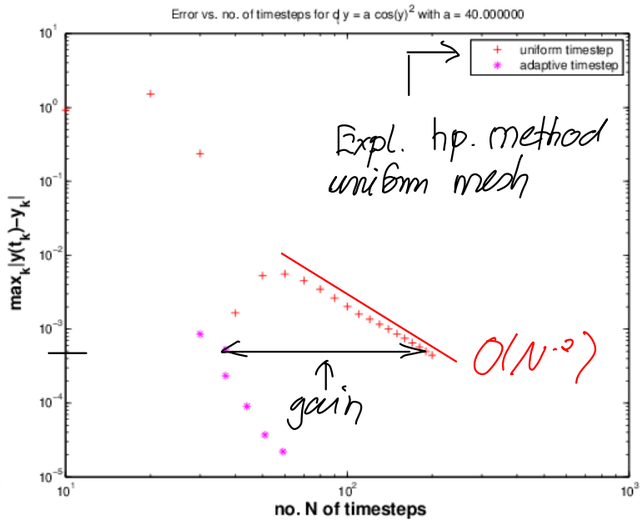
Nevertheless: EST_k useful for timestep control

Exp. 6.5.2.9 (Gain through adaptivity)

→ Setting of Exp. 6.5.2.8, $y(0) = 0$, RTOL varied



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



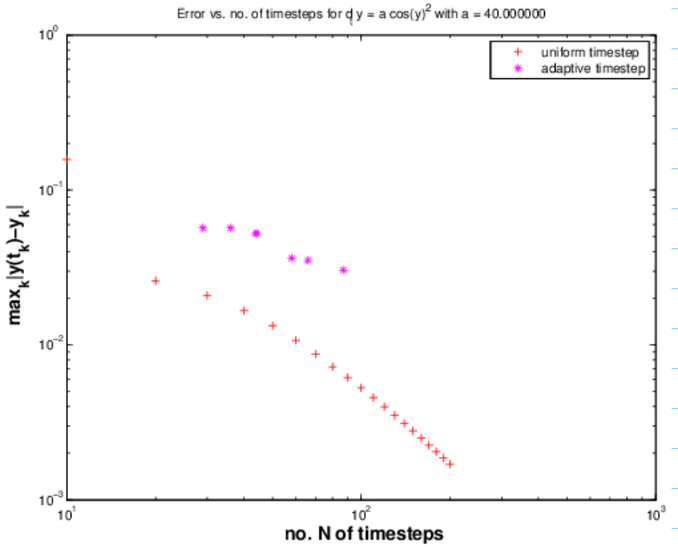
Error vs. computational effort

Exp. 6.5.2.10 ("Failure" of adaptive timestepping)

→ Setting of Exp. 6.5.2.8, $y(0) < 0$

RTOL varied

⑥



Error vs. computational effort

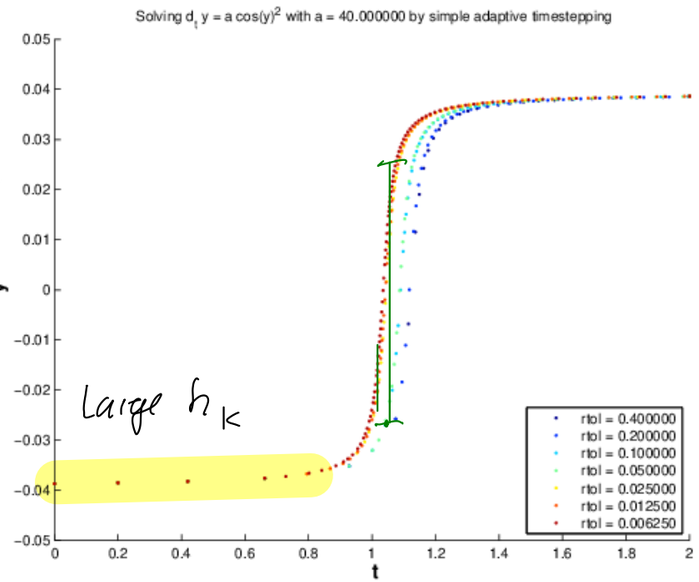
◁ Results by adaptive timestepping much worse than those on a uniform temporal mesh.

Sensitive dependence ▷ of position of step on initial value

Still stable

"Bad" solution produced by the adaptive algorithm is a good solution for slightly perturbed initial values!

↓ typical solution



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

§ 6.5.2.11 (Refined local stepsize control)

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

↓ Based on EST_k

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

Order $p+1$ ↖ ↗ order $= p$

▷ One-step error:

$$\Psi^h y_k - \Phi^h y_k = c h^{p+1} + O(h^{p+2})$$
$$\widetilde{\Psi}^h y_k - \Phi^h y_k = O(h^{p+2}) \quad \text{for } h \to 0$$

$$\Rightarrow \mathrm{EST}_k \doteq c h^{p+1} \quad [\text{up to } O(h^{p+2})]$$

↕ "known"

$$\Rightarrow \text{estimate}: \quad c \doteq \frac{\mathrm{EST}_k}{h^{p+1}} \qquad (*)$$

"Optimal timestep" h_* : $\mathrm{EST}_k(h_*) \overset{!}{=} \mathrm{TOL}$

$\doteq c h_*^{p+1}$

(7)

with (*): $h_* = h \left(\frac{TOL}{EST_k} \right)^{1/(p+1)}$

Use this h_* for

- repeated step in case $TOL < EST_k$
- next step, if $TOL > EST_k$

C++11 code 6.5.2.17: Refined local stepsize control for single step methods

→ GITHUB

```
// Auxiliary function:  default norm for an EIGEN vector type
template <class State>
double _norm(const State &y) { return y.norm(); }
// Adaptive single-step integrator
template <class DiscEvolOp, class State, class NormFunc = decltype(_norm<State>)>
std::vector<std::pair<double, State> >
odeintssctrl(DiscEvolOp &&Psilow, unsigned int p, DiscEvolOp &&Psihigh,
             const State &y0,double T, double h0,
             double reltol, double abstol, double hmin,
             NormFunc &norm = _norm<State>) {
  double t = 0; // initial time t_0 = 0
  State y = y0; // current state, initialized here
  double h = h0; // timestep to start with
  std::vector<std::pair<double,State>> states; // vector (t_k, y_k)_k
  states.push_back({t, y});

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

    // Optimal stepsize according to (6.5.2.16)
    if (est < tol)  { // step accepted
      states.push_back({t = t+std::min(T-t,h),y = yh}); // store next approximate
          state
      }
    h = h*std::max(0.5,std::min(2.,std::pow(tol/est,1./(p+1)))); //   h_*
    if (h < hmin) {
      std::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." << std::endl;
    }
  }
  return states;
}
```

6.5.3. Embedded Runge-Kutta Methods

Definition 6.4.0.9. 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 $\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y})$, $\mathbf{f}: \Omega \to \mathbb{R}^N$, 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 $\mathbf{k}_i \in \mathbb{R}^N$, $i = 1,\ldots,s$, are called **increments**, $h > 0$ is the size of the timestep.

Idea: Same increments for $\widetilde{\Psi}^h$ and Ψ^h, only weights b_i different

Notation:

Extended Butcher scheme

b_i {

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	$\frac{2}{3}$	$\frac{1}{6}$ → Ψ
$\widehat{y}_1$	$\frac{1}{10}$	0	$\frac{3}{10}$	$\frac{2}{5}$	$\frac{1}{5}$ → $\widetilde{\Psi}$

Merson's embedded RK-SSM

▷ "Workhorses" in libraries for numerical integration

Typical interface:

Pass:
- Right-hand side function $\mathbf{f}$
- Initial value $\mathbf{y}_0$
- Initial time t_0 & final time T
- Tolerances ATOL, RTOL

Rem 6.5.3.4 (Tolerances and accuracy)

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!

* in particular for IVPs with sensitive dependence on initial conditions. → Exp. 6.5.2.10

Please answer review questions

After you have watched the video take a short break and then try to answer the

review questions 6.5.3.9

You should do this without consulting other parts of the lecture document or tablet notes.