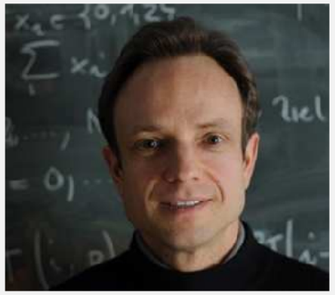
ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Course Video
Section 11.5: Adaptive Stepsize Control

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 14, 2022

(C) Seminar für Angewandte Mathematik, ETH Zürich



Tablet notes may have been corrected and/or extended after the recording of the video!

Dependencies. Section 7.6 and Section 11.4

Duration: minutes

XI Numerical Integration – Single-Step Methods

11.5. Adaptive Stepsize Control

11.5.1. The Need for Timestep Adaptation

Ex. 11.5.1.1. (Oregonator reaction)

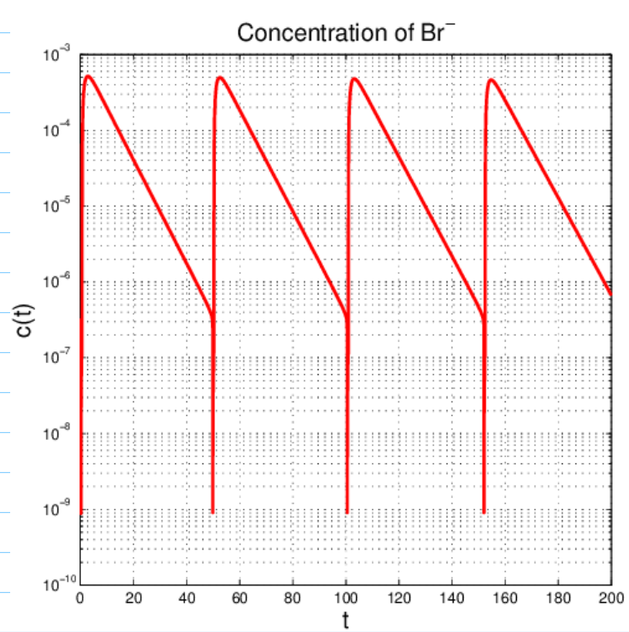
This is a special case of an "oscillating" Zhabotinski-Belousov reaction :

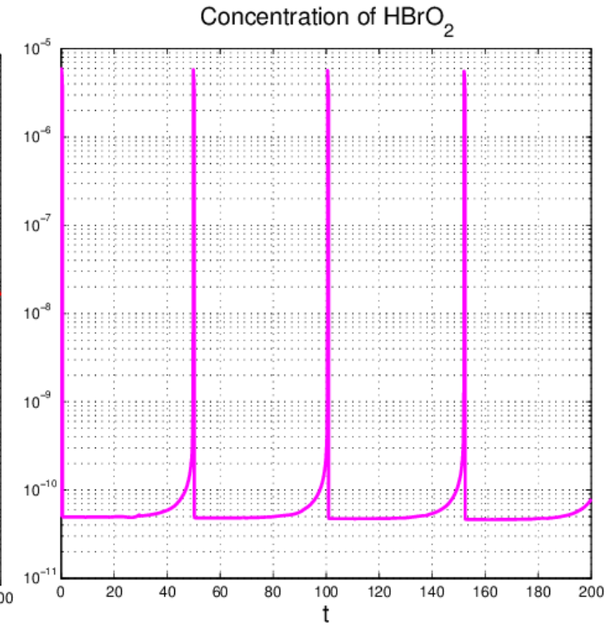
$$\begin{aligned} BrO_3^- + Br^- &\mapsto HBrO_2 \\ HBrO_2 + Br^- &\mapsto Org \\ BrO_3^- + HBrO_2 &\mapsto 2\,HBrO_2 + Ce(IV) \\ 2\,HBrO_2 &\mapsto Org \\ Ce(IV) &\mapsto Br^- \end{aligned} \tag{6.5.0.2}$$

ODE model for reaction kinetics :

$$\begin{aligned} y_1 := c(BrO_3^-): \quad \dot{y}_1 &= -k_1 y_1 y_2 - k_3 y_1 y_3 \,, \\ y_2 := c(Br^-): \quad \dot{y}_2 &= -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5 \,, \\ y_3 := c(HBrO_2): \quad \dot{y}_3 &= k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2 \,, \\ y_4 := c(Org): \quad \dot{y}_4 &= k_2 y_2 y_3 + k_4 y_3^2 \,, \\ y_5 := c(Ce(IV)): \quad \dot{y}_5 &= k_3 y_1 y_3 - k_5 y_5 \,, \end{aligned} \tag{6.5.0.3}$$

2





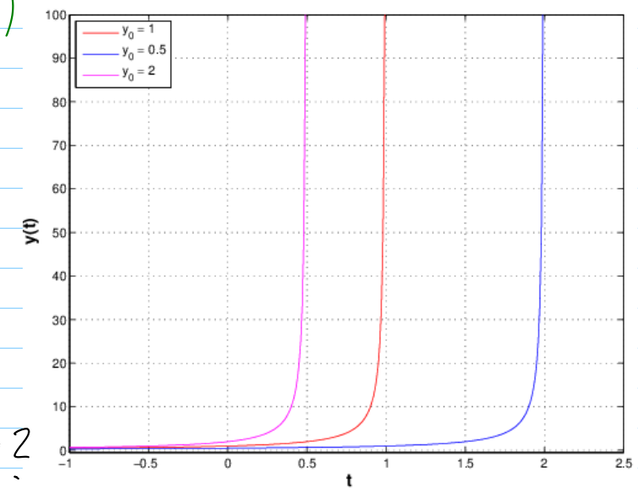
▷ Solution highly non-uniform in time

▷ Use highly non-uniform temporal mesh

Ex. 11.5.1.4 (Explosion ODE)

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

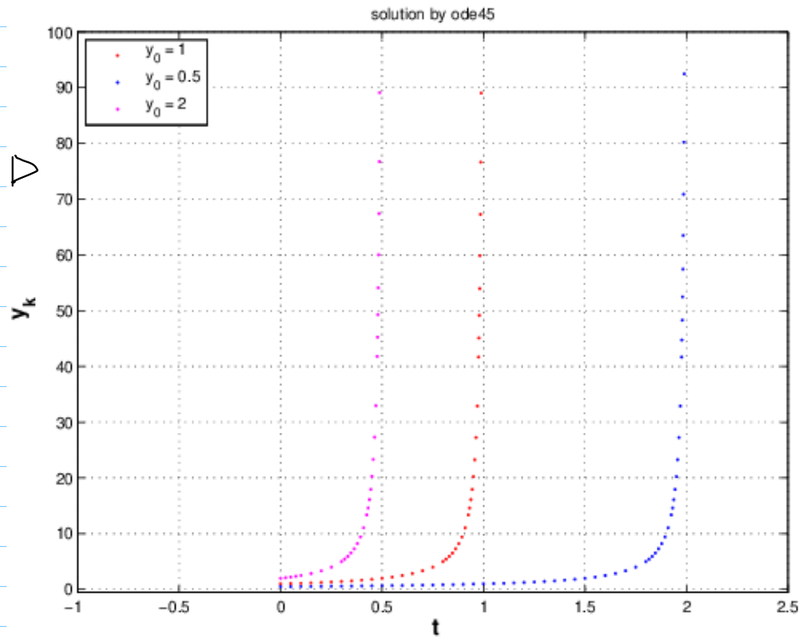
▷ Finite-time blow-up at $t = 1/y_0$



Uniform-h expl. Euler?

Sneak preview

(y_k) by adaptive explicit RK-SSM ▷



③

11.5.2 Local-in-time Stepsize Control

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

Principles: (→ Adaptive quadrature, Sect. 7.6)

efficient — reliable! — specified by user

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 $\|\mathbf{y}(T) - \mathbf{y}_M\| < \mathtt{TOL}$, $\mathtt{TOL}$ = 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

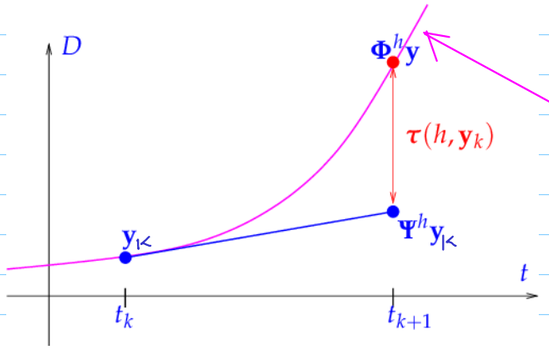
Tool: Local-in-time one-step error estimator (*a posteriori*, based on $\mathbf{y}_k, h_{k-1}$)

$= \Psi^{h_k} y_k - \Phi^{h_k} y_k$ [not accessible]

Can only be estimated!

Choose t_{k+1} only based on t_k, y_k, h_{k-1}

(at $O(1)$ cost/step)



◁ One-step error

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

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

Idea: *Estimation of one-step error*

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

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

$$\underbrace{\Phi^h \mathbf{y}_k - \Psi^h \mathbf{y}_k}_{\text{one-step error}} \approx \mathrm{EST}_k := \widetilde{\Psi}^h \mathbf{y}_k - \Psi^h \mathbf{y}_k \,. \qquad (11.5.2.3)$$

Heuristics for concrete $h > 0$ — A computable quantity!

error estimate

4

§ 11.5.2.4. (Local timestep adaptation)

Available : EST_k for timestep size h

Compare :

relative / absolute tolerance [User supplied]

$$EST_k \leq TOL := \max\{ RTOL \cdot \|y_k\| , ATOL \}$$

Yes → Accept current step (with $h = h_k$) ⇓ Advance to next step

- $y_{k+1} = \tilde{\Psi}^{h_k} y_k$
- $h \leftarrow \alpha h$, $\alpha \approx 1.1$

No → Reject current step ⇓ Repeat with $h \leftarrow h/2$

"ground to a halt" → e.g. in blow-up cases

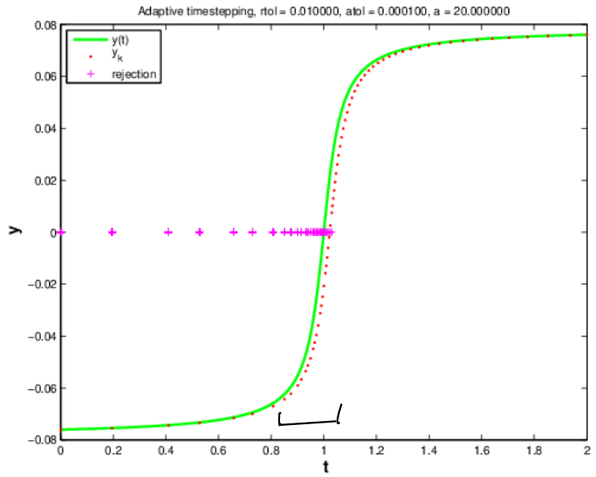
C++ code 11.5.2.6: Simple local stepsize control for single step methods → GITLAB

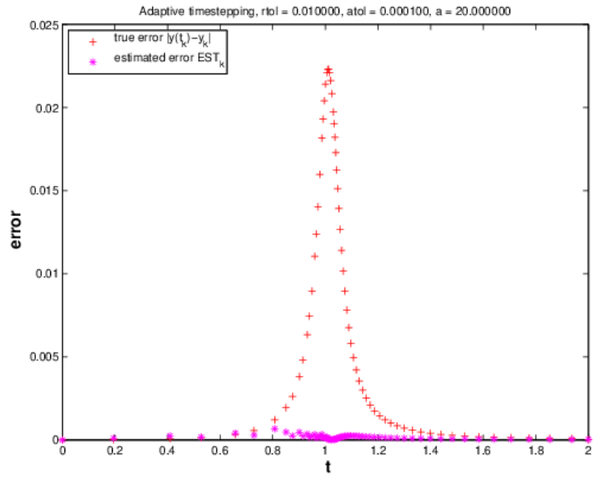
```
// Auxiliary function:  default norm for an EIGEN vector type
template <class State>
double _norm(const State &y) { return y.norm(); }

// Adaptive numerical integrator based on local-in-time stepsize control
template <class DiscEvolOp, class State,
          class NormFunc = decltype(_norm<State>)>
std::vector<std::pair<double, State>> odeintadapt(
    DiscEvolOp &&Psilow, 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
  double h = h0;  // timestep to start with
  // vector of times/computed states:  (t_k, y_k)_k
  std::vector<std::pair<double, State>> states;
  states.push_back({t, y});  // initial time and state
  // 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);  // local error estimate EST_k
    if (est < std::max(reltol * norm(y), abstol)) {  // step accepted
      y = yh;            // use high order approximation
      t = t + std::min(T - t, h);  // next time t_k   → advance
      states.push_back({t, y});    // store approximate state
      h = 1.1 * h;  // try with increased stepsize
    } else {        // step rejected
      h = h / 2;    // try with half the stepsize   → repeat
    }
    // Numerical integration has ground to a halt !
    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 = " << t << "." << std::endl;
    }
  } // end main loop
  return states; // ok thanks to return value optimization (RVO)
}
```

Exp. 11.5.2.8 (Simple adaptive stepsize control)

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





Statistics: `66 timesteps, 131 rejected timesteps`

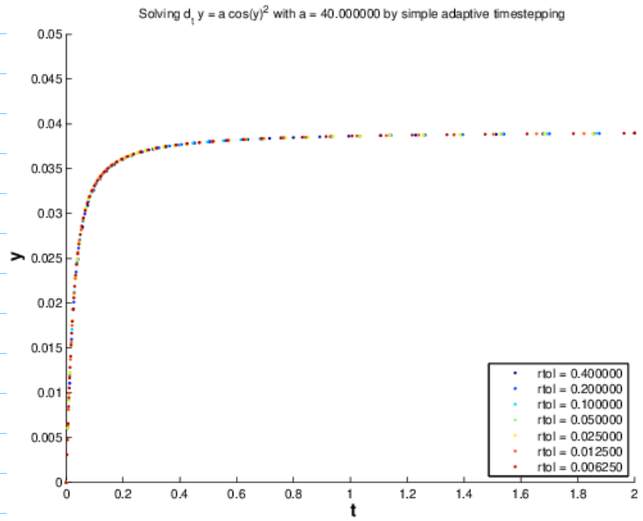
$\downarrow$

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

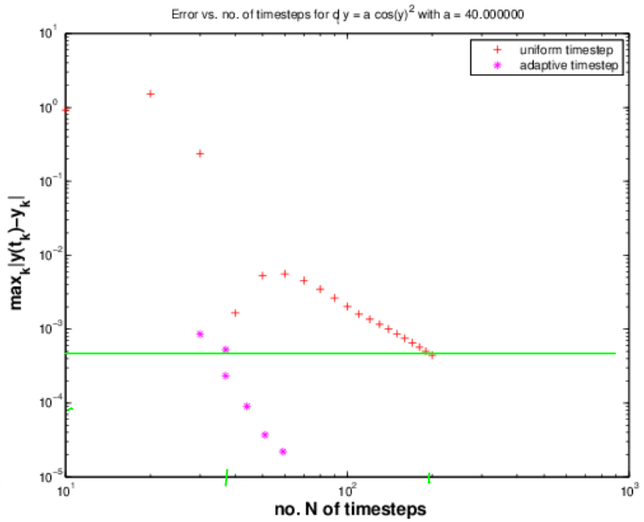
Nevertheless: EST_k useful for timestep control

Exp. 11.5.2.9 (Gain through adaptivity)

$\rightarrow$ Setting of Exp. 11.5.2.8, $y(0) = 0$, RTOL varied



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

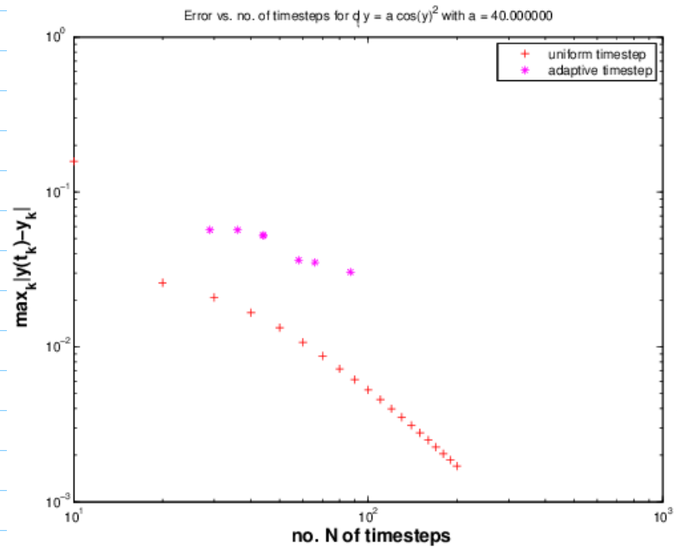


Error vs. computational effort

Exp. 11.5.2.10 ("Failure" of adaptive timestepping)

$\rightarrow$ Setting of Exp. 11.5.2.8, $y(0) < 0$

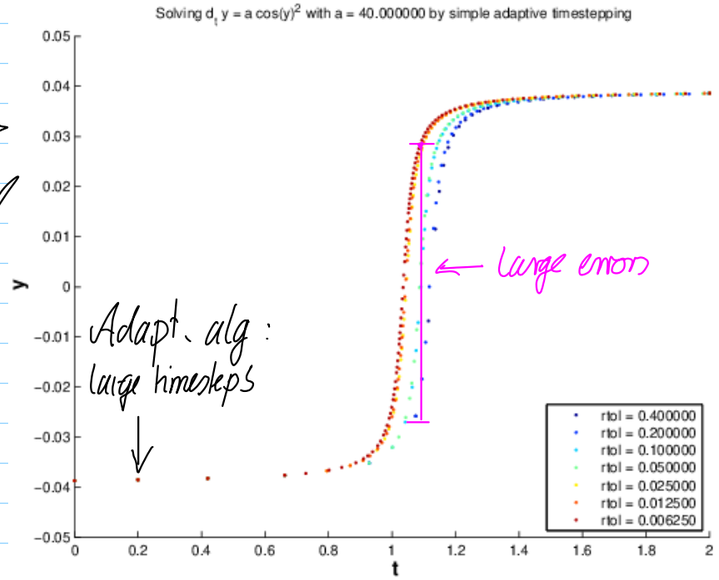
RTOL varied



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



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

Still stable:

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

"typical/acceptable" solution

§ 11.5.2.11 (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 $\mathrm{EST}_k : \mathrm{TOL}$

- $$\mathrm{EST}_k = \mathrm{EST}_k(h) := \underbrace{\tilde{\Psi}^h y_k}_{\text{Order } p+1} - \underbrace{\Psi^h y_k}_{\text{order } p}$$

▷ One-step errors:

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

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

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

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

7 + (*) $\quad \frac{EST_k}{h^{p+1}} \cdot h_*^{p+1} = TOL$

$\triangleright \quad h_* = h \left(\frac{TOL}{EST_k} \right)^{\frac{1}{p+1}}$

Use this h_* for

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

C++ code 11.5.2.18: Refined local stepsize control for single step methods ➙ GITLAB

```
// 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
  // Array for returning pairs of times/states (t_k, y_k)_k
  std::vector<std::pair<double, State>> states;
  states.push_back({t, y}); // initial time and state
  // 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 (11.5.2.17)
    if (est < tol) {  // step accepted
      // store next approximate state
      states.push_back({t = t + std::min(T - t, h), y = yh});
    }
    // New timestep size according to (11.5.2.17)
    h *= std::max(0.5, std::min(2., 0.9 * std::pow(tol / est, 1. / (p + 1)))); //
    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 = " << t << "." << std::endl;
    }
  } // end main loop
  return states;
}
```

8

11.5.3. Embedded Runge-Kutta Methods

Definition 11.4.0.11. Explicit Runge-Kutta single-step 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 $\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y})$, $\mathbf{f} : \Omega \to \mathbb{R}^N$, is defined by ($\mathbf{y}_0 \in D$)

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

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

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

Notation: Extended Butcher scheme

$$\begin{array}{c|ccccc}
0 & & & & & \\
\frac{1}{3} & \frac{1}{3} & & & & \\
\frac{1}{3} & \frac{1}{6} & \frac{1}{6} & & & \\
\frac{1}{2} & \frac{1}{8} & 0 & \frac{3}{8} & & \\
1 & \frac{1}{2} & 0 & -\frac{3}{2} & 2 & \\
\hline
y_1 & \frac{1}{6} & 0 & 0 & \frac{2}{3} & \frac{1}{6} \\
\hline
\hat{y}_1 & \frac{1}{10} & 0 & \frac{3}{10} & \frac{2}{5} & \frac{1}{5}
\end{array}
\qquad
b_i \left\{ \begin{array}{l} \\ \\ \\ \\ \\ \longrightarrow \Psi^h \\ \longrightarrow \tilde{\Psi}^h \end{array} \right.$$

Merson's embedded RK-SSM

▷ "Workhorses" in libraries for numerical integration

Typical interface:

Pass:
- Right-hand side function $\underline{f}$
- Initial value y_0
- Initial time t_0 & final time T
- Tolerances ATOL, RTOL

Rem 11.5.3.4 (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. $\longrightarrow$ Exp. 11.5.2.10

▷ Please try to answer Review Questions 11.5.3.10 without using any aids