Adaptive numerical solvers

for Ordinary Differential Equations

Bruno Guindani
Michele Vidulis

POLITECNICO
MILANO 1863

July 22, 2019

Introduction

Ordinary differential equations (ODE)

Given $I = [t_0, t_F] \subset \mathbb{R}$, $\quad f(t, \boldsymbol{y}) : I \times \mathbb{R}^n \to \mathbb{R}^n$, $\quad f \in C^1$, and $t_0 \in I, \boldsymbol{y}_0 \in \mathbb{R}^n$:

Initial Value Problem (IVP):

find a C^1 function $\boldsymbol{y}(t) : I \to \mathbb{R}^n$ that solves

$$\begin{cases} \boldsymbol{y}'(t) = f(t, \boldsymbol{y}(t)) & \text{with } t \in I \\ \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \end{cases}$$

(first order ODE)

Existence and uniqueness guaranteed under *Lipschitz continuity* of f

Iterative methods

- Discretization of time into N intervals: $t_0,\ t_1,\ \ldots,\ t_N = t_F$ through a scretization step h: $t_{n+1} = t_n + h$, with $n = 0, 1, \ldots, N$
- $\boldsymbol{y}(t_n)$ is numerically approximated by $\boldsymbol{u}_n$:

$$\boldsymbol{u}_0 = \boldsymbol{y}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_N \in \mathbb{R}^n$$

Iterative methods

- Discretization of time into N intervals: $t_0,\ t_1,\ \ldots,\ t_N = t_F$ through a scretization step h: $t_{n+1} = t_n + h$, with $n = 0, 1, \ldots, N$
- $\boldsymbol{y}(t_n)$ is numerically approximated by $\boldsymbol{u}_n$:

$$\boldsymbol{u}_0 = \boldsymbol{y}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_N \in \mathbb{R}^n$$

- In **single-step methods**, $\boldsymbol{u}_{n+1}$ depends directly only on the one previous step $\boldsymbol{u}_n$:

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\ \phi(t_n,\ h,\ \boldsymbol{u}_n,\ \boldsymbol{u}_{n+1},\ f)$$

- In **explicit** methods, $\boldsymbol{u}_{n+1}$ does not appear in ϕ
- In **implicit** methods, $\boldsymbol{u}_{n+1}$ appears in ϕ

$\implies$ nonlinear equations

Runge-Kutta methods

- Family of **single-step** methods
- Weighted average of s evaluations (**stages**) of f:

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h \sum_{i=1}^{s} b_i \boldsymbol{K}_i \quad \text{with}$$

$$\boldsymbol{K}_i = f(t_0 + c_i h,\ \boldsymbol{u}_n + \sum_{j=1}^{s} a_{ij} \boldsymbol{K}_j)$$

Runge-Kutta methods

- Family of **single-step** methods
- Weighted average of s evaluations (**stages**) of f:

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h \sum_{i=1}^{s} b_i \boldsymbol{K}_i \quad \text{with}$$

$$\boldsymbol{K}_i = f\big(t_0 + c_i h,\ \boldsymbol{u}_n + \sum_{j=1}^{s} a_{ij} \boldsymbol{K}_j\big)$$

- *Butcher tableau*:

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & & \ddots & \\ c_s & a_{s1} & & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array} \qquad \text{with } c_i = \sum_j a_{ij}$$

Runge-Kutta methods

- Family of **single-step** methods
- Weighted average of s evaluations (**stages**) of f:

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h \sum_{i=1}^{s} b_i \boldsymbol{K}_i \quad \text{with}$$

$$\boldsymbol{K}_i = f(t_0 + c_i h,\ \boldsymbol{u}_n + \sum_{j=1}^{s} a_{ij} \boldsymbol{K}_j)$$

- *Butcher tableau*:

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & & \ddots & \\ c_s & a_{s1} & & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array} \qquad \text{with } c_i = \sum_j a_{ij}$$

- $O(sn^2)$ if f linear
- Explicit if $[a_{ij}]_{ij}$ is lower triangular

Examples of RK variants (1)

- **Forward Euler** (FE) (explicit):

$$a = 0,\ b = 1,\ c = 0$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + hf(t_n, \boldsymbol{u}_n)$$

- RK4 (standard) (explicit):

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & & \frac{1}{2} & & \\
1 & & & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}$$

Examples of RK variants (2)

- Heun (explicit):

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

- **Iserles-Nørsett** (implicit):

$$\begin{array}{c|cccc} \frac{1}{3} & \frac{1}{3} & & & \\ \frac{2}{3} & \frac{1}{3} & \frac{1}{3} & & \\ \frac{21+\sqrt{57}}{48} & & & \frac{21+\sqrt{57}}{48} & \\ \frac{21-\sqrt{57}}{48} & & & \frac{3-\sqrt{57}}{24} & \frac{21+\sqrt{57}}{48} \\ \hline & \frac{9+3\sqrt{57}}{16} & \frac{9+3\sqrt{57}}{16} & -\frac{1+3\sqrt{57}}{16} & -\frac{1+3\sqrt{57}}{16} \end{array}$$

Convergence analysis for RK

- **Convergence** $\rightarrow$ absolute error: $\|y_n - u_n\| \simeq O(h^q)$
- **Consistence** $\rightarrow$ truncation error: $\max_n \|\tau_n(h)\| \simeq O(h^q)$
- Under reasonable Lipschitz continuity assumptions on ϕ, a single-step method which is consistent is also convergent

Convergence analysis for RK

- **Convergence** $\rightarrow$ absolute error: $\|y_n - u_n\| \simeq O(h^q)$
- **Consistence** $\rightarrow$ truncation error: $\max_n \|\tau_n(h)\| \simeq O(h^q)$
- Under reasonable Lipschitz continuity assumptions on ϕ, a single-step method which is consistent is also convergent
- Runge-Kutta is consistent iff $\sum_i b_i = 1 \implies$ **convergent**
- Steep limitations on order of convergence:
  - Maximum order is the number of stages
  - If $s \geq 5$, equality cannot be achieved in explicit variants

order	5	6	7	8
minimum s	6	7	9	11

Adaptive methods

- Step h is **updated** at every iteration adaptively, i.e. based on the trend of the solution
  - Small h near steep slopes, large h near flat points
  - *A posteriori* **estimate of error** is needed
  - Compute two-round solution with $\frac{h}{2}$, with single-round solution with h
- No need for input of "correct" step
- Computational gain

Consistency of adaptive methods

Forward Euler:

- Truncation errors:

$$e_h = h^2/2y''(\xi), \qquad e_{h/2} = h^2/8y''(\eta) \cdot 2 + o(h^2)$$

- Error estimate:

$$|u_{h/2} - u_h| \simeq |e_{h/2}| \simeq h^2/4|y''(\widehat{\eta})| + o(h^2) \overset{\downarrow}{<} \frac{\varepsilon}{2} \quad \text{(tolerance)}$$

Consistency of adaptive methods

Forward Euler:

- Truncation errors:

$$e_h = h^2/2y''(\xi), \qquad e_{h/2} = h^2/8y''(\eta) \cdot 2 + o(h^2)$$

- Error estimate:

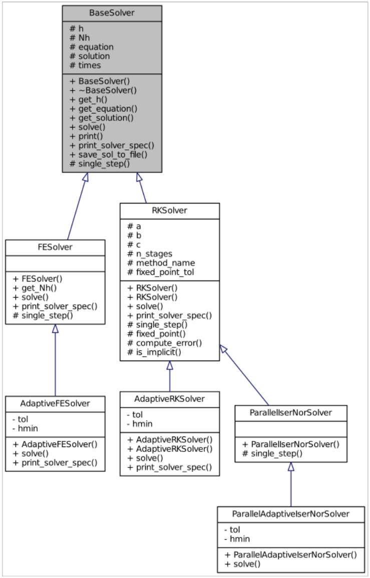
$$|u_{h/2} - u_h| \simeq |e_{h/2}| \simeq h^2/4|y''(\widehat{\eta})| + o(h^2) \overset{\downarrow}{<} \frac{\varepsilon}{2} \quad \text{(tolerance)}$$

Runge-Kutta:

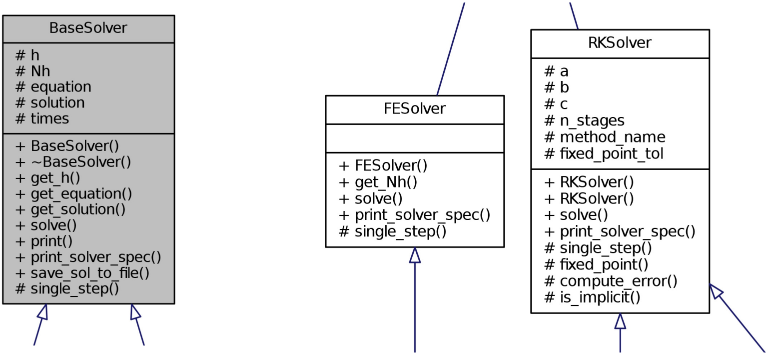
- ...

Code structure

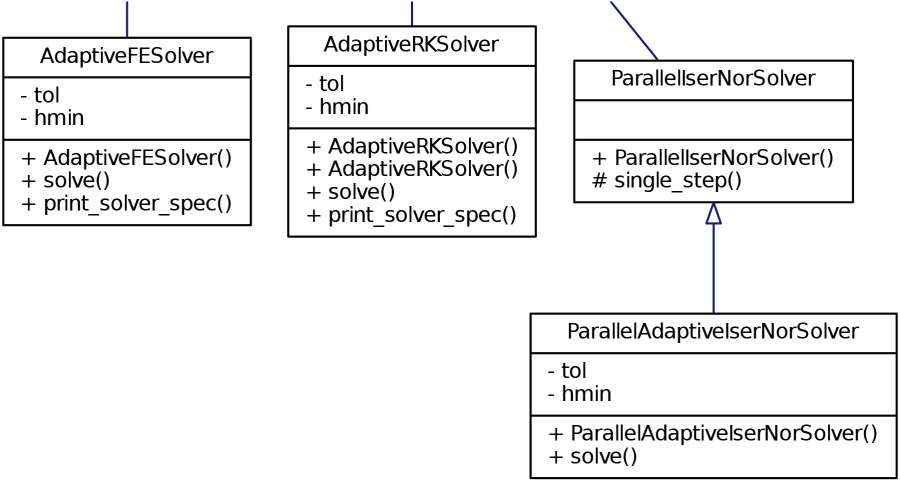
Code structure



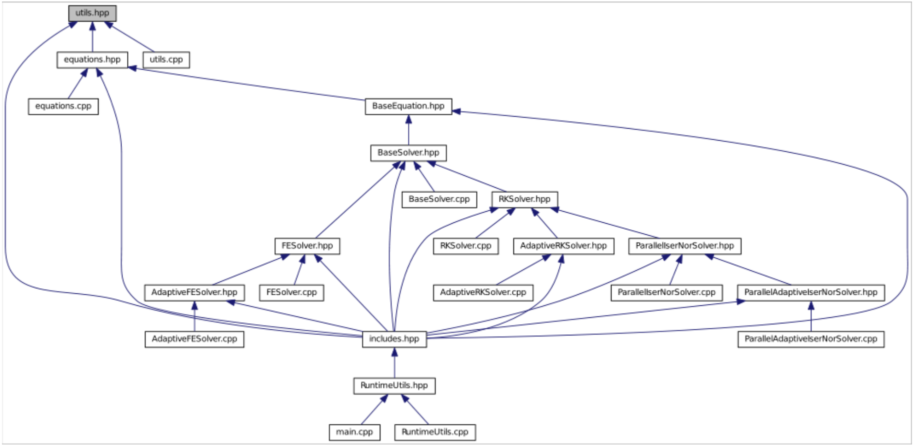
Base classes



Adaptive classes

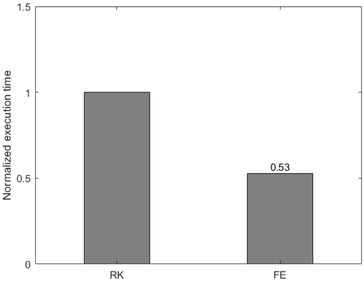


Dependences



Implementation choices

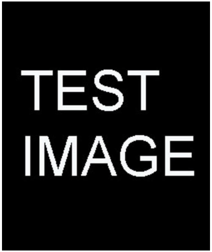
- Separate FE class is much more efficient than RK specialized class:



- Adaptive `single_step()` class methods are not efficient

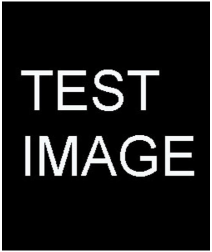
Parallel Iserles-Nørsett

- Exploits block-diagonal structure of Butcher array
- Runs in parallel on 2 processors, each dealing with one indpendent 2-by-2 block
- Fixed point algorithm was used for nonlinear equations



Results

Results

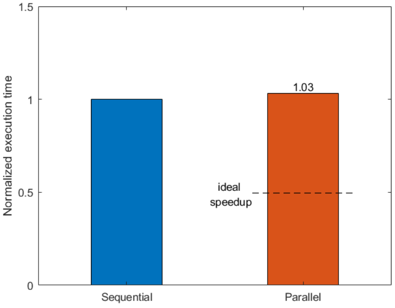


Actual efficiency of parallelism

Speedup is heavily dependent on the problem function:

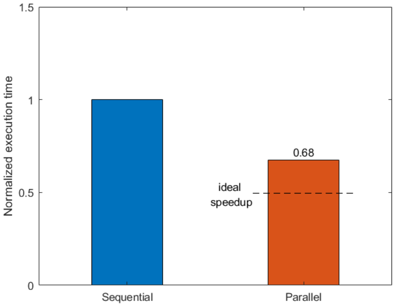
Test #5

$$f(t, y(t)) = -16y(t)$$



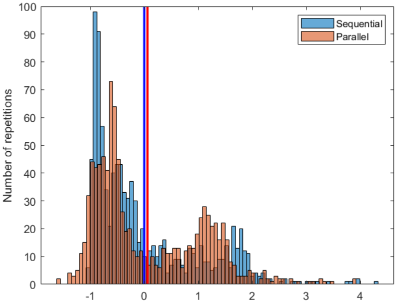
Test #6

$$f(t, y(t)) = \exp_2(-\frac{y(t)}{4} + 6 + 10t)$$

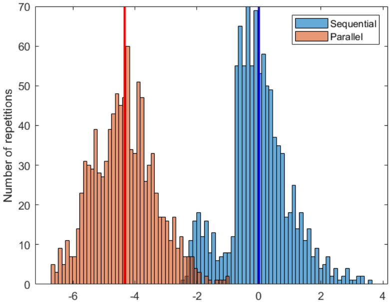


Multiple run results

Test #5



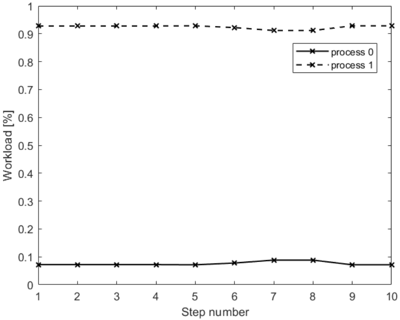
Test #6



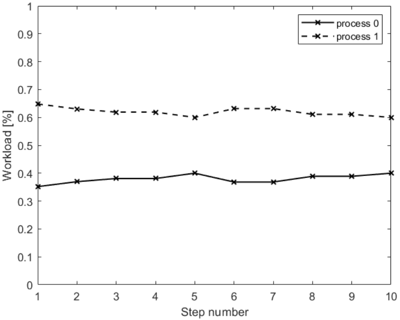
Why?

Workload distribution

Test #5



Test #6

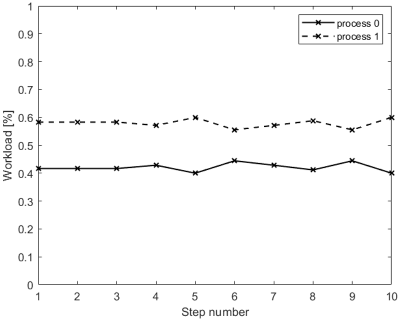


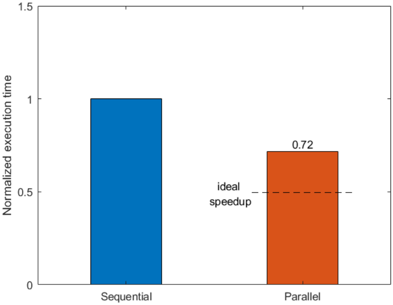
$\Longrightarrow$ **load imbalance**

A vectorial example

- Efficiency still depends on the function
- Here fixed point iterations are well-balanced among processors:

$$\text{Test } \#4: \qquad \boldsymbol{f}(t, \boldsymbol{y}(t)) = \begin{bmatrix} -3 & -1 \\ 1 & -5 \end{bmatrix} \boldsymbol{y}(t) + \begin{bmatrix} \sin(t) \\ -2t \end{bmatrix}$$





Bibliography

- Podhaisky, *Parallel two-step Runge-Kutta methods*
- Quarteroni, Saleri, *Calcolo scientifico*
- Quarteroni, Sacco, Saleri, Gervasio, *Matematica numerica*
- Solodushkin, Iumanova, *Parallel Numerical Methods for Ordinary Differential Equations: a Survey*