

Adaptive numerical solvers

for Ordinary Differential Equations

Bruno Guindani
Michele Vidulis



POLITECNICO
MILANO 1863

July 22, 2019

<https://github.com/poliprojects/apc-project>

Introduction

Ordinary differential equations (ODE)

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

Initial Value Problem (IVP):

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

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

(first order ODE)

Existence and uniqueness guaranteed under *Lipschitz continuity* of f

Iterative methods

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

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

Iterative methods

- Discretization of time into N intervals: $t_0, t_1, \dots, t_N = t_F$ through a discretization step h : $t_{n+1} = t_n + h$, with $n = 0, 1, \dots, N$

- $\mathbf{y}(t_n)$ is numerically approximated by \mathbf{u}_n :

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

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

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

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

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

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

Runge-Kutta methods

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

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

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

- *Butcher tableau*:

c_1	a_{11}	\dots	a_{1s}
\vdots		\ddots	
c_s	a_{s1}		a_{ss}
	b_1	\dots	b_s

$$\text{with } c_i = \sum_j a_{ij}$$

Runge-Kutta methods

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

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

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

- *Butcher tableau*:

c_1	a_{11}	\dots	a_{1s}
\vdots		\ddots	
c_s	a_{s1}		a_{ss}
	b_1	\dots	b_s

$$\text{with } c_i = \sum_j a_{ij}$$

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

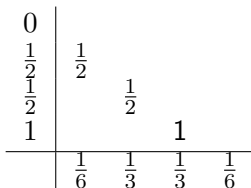
Examples of RK variants (1)

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

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

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

- **RK4 (standard)** (explicit):



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):

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

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
 - ▶ Compare two-round solution computed with step $\frac{h}{2}$, with single-round solution computed with step h
- No need for input of “correct” step
- Computational gain

Error computation for adaptive methods

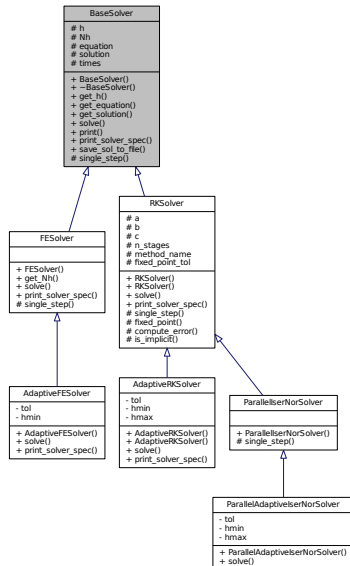
- Relative error of infinity norm is used:

$$\frac{\|\mathbf{u}_{h/2} - \mathbf{u}_h\|_\infty}{\|\mathbf{u}_h\|_\infty} < \frac{\varepsilon}{2} \quad (\text{tolerance})$$

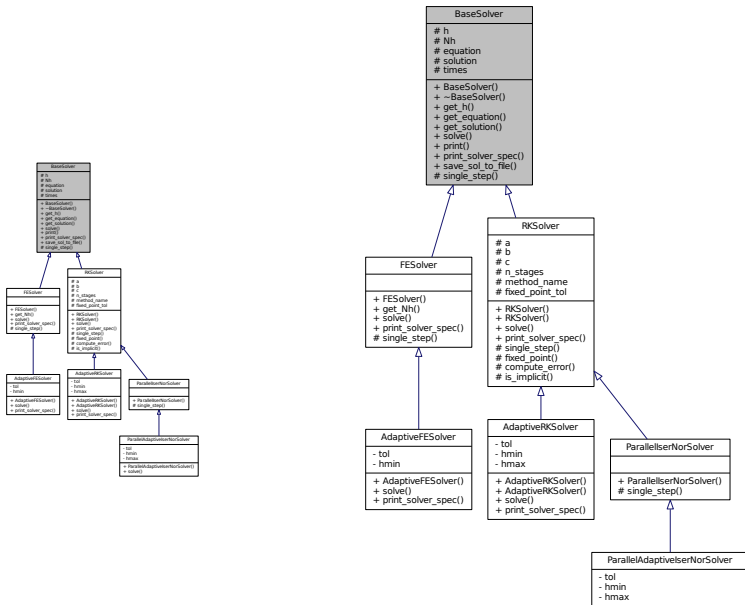
- This guarantees consistency (\implies convergence)
- At each iteration h can be doubled, halved, or unchanged
- h_{min} and h_{max} are required for some methods

Code structure

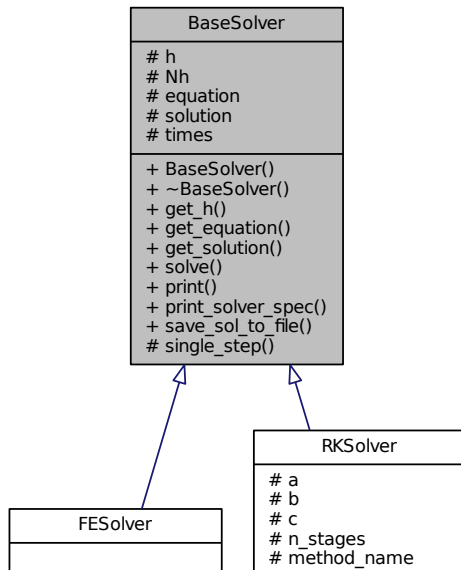
Code structure



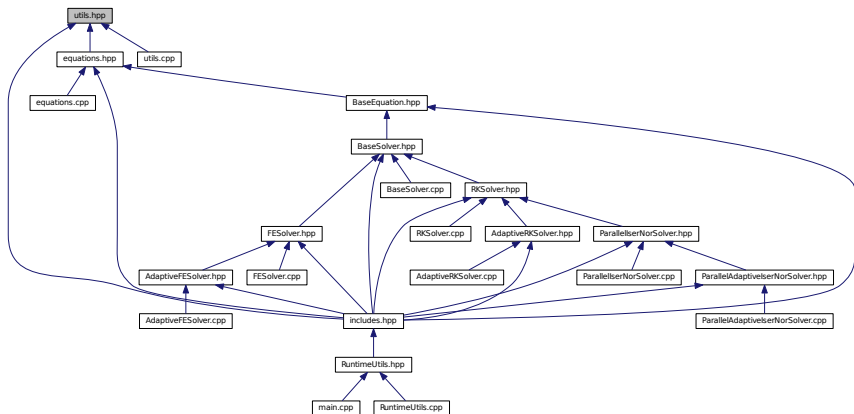
Base classes



Adaptive classes

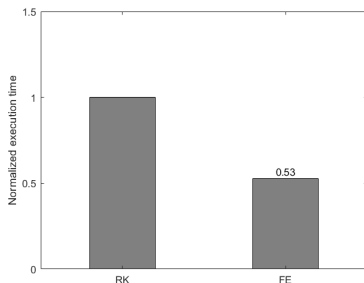


Dependencies



Implementation choices

- Seven test functions are provided
- Separate FE class is much more efficient than RK specialized class:



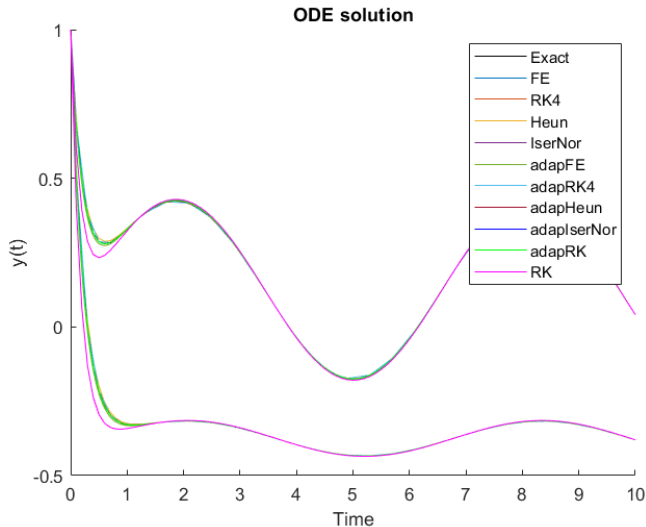
- Adaptive `single_step()` class methods are not efficient

Parallel Iserles-Nørsett

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

Results

Comparison between methods



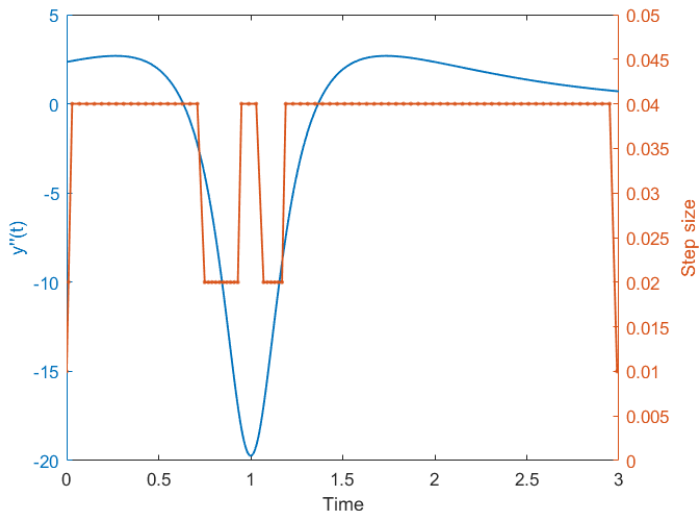
(components of test #4)

Comparison of Mean Square Errors



TEST
IMAGE

Trend of step in adaptive methods



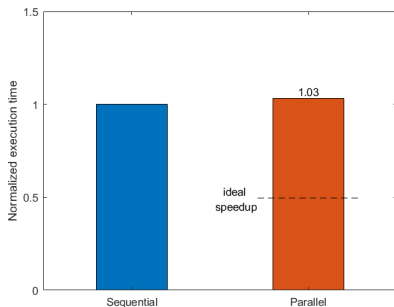
(test #7, adaptive Heun method)

Actual efficiency of parallelism

Speedup is heavily dependent on the problem function:

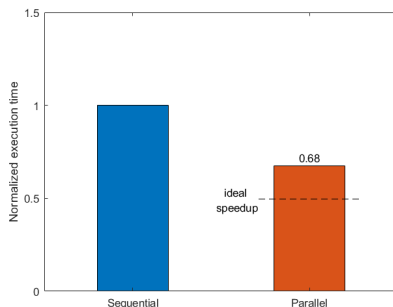
Test #5

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



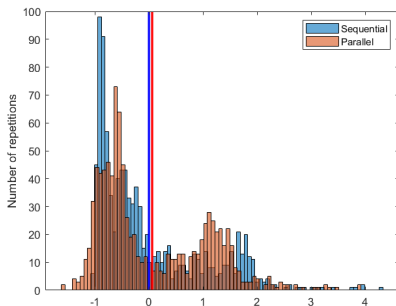
Test #6

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

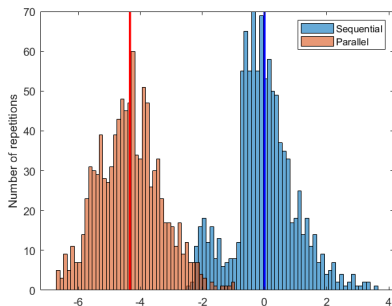


Multiple run results

Test #5



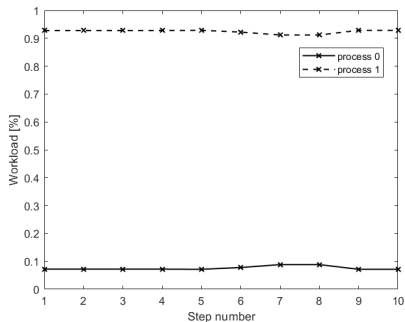
Test #6



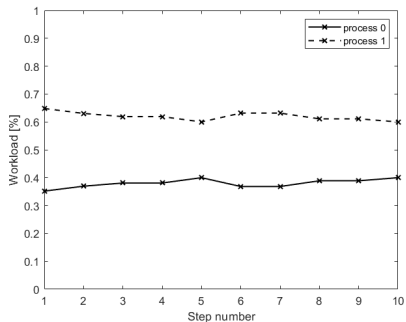
Why?

Workload distribution

Test #5



Test #6

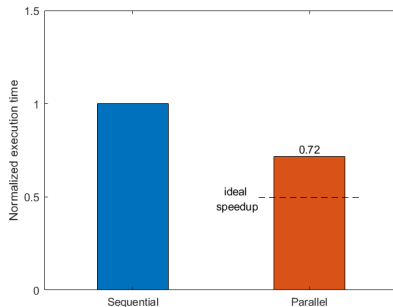
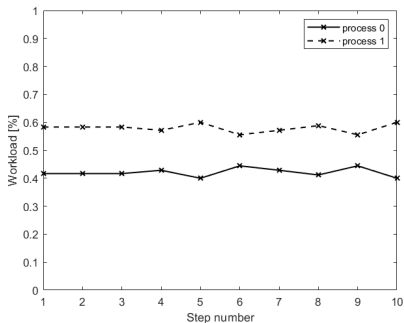


⇒ load imbalance





A vectorial example

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

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



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*