Runge-Kutta guaranteed integration of ODEs

Olivier Bouissou
olivier.bouissou@cea.fr

CEA Saclay

Taylor Methods Workshop 2006
Boca Raton, Florida

Content of this talk.

Outline

Motivation

Context of this work: Validation of embedded systems (avionics, automotive).

- Hybrid Systems: composed of two distinct parts
  - *discrete subsystem:* a discrete transition system (finite automata, C program).
  - *continuous subsystem:* a switched system of differential equations.
- Validation of such systems:
  - computes overapproximation of all reachable states.
  - needs rigourous bounds on the all the possible values of the continuous variables.

Motivation

Context of this work: Validation of embedded systems (avionics, automotive).

- Hybrid Systems: composed of two distinct parts
  - *discrete subsystem:* a discrete transition system (finite automata, C program).
  - *continuous subsystem:* a switched system of differential equations.
- Validation of such systems:
  - computes overapproximation of all reachable states.
  - needs rigourous bounds on the all the possible values of the continuous variables.

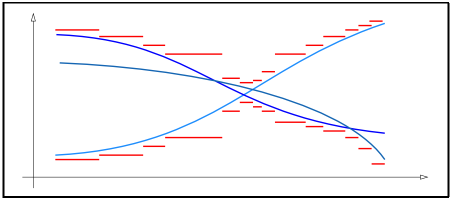
What we need: a method for computing representable functions which are guaranteed to enclose all the possible continuous dynamics.

Objectives.

Suppose you have a switched dynamical system:

$$b \rightarrow (\dot{y} = f(y)) \Box\ b' \rightarrow (\dot{y} = g(y))$$

We want to compute two functions that are guaranteed to enclose all the possible values of y.

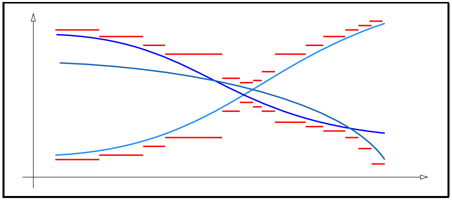


Objectives.

Suppose you have a switched dynamical system:

$$b \rightarrow (\dot{y} = f(y)) \Box \; b' \rightarrow (\dot{y} = g(y))$$

We want to compute two functions that are guaranteed to enclose all the possible values of y.



What we really need: given $\dot{y} = f(y)$, a set of enclosures $[\mathbf{y_n}]$ such that $\forall t_n, y(t_n) \in [\mathbf{y_n}]$.

What already exists.

- On the one side, validated integration using Taylor methods:
    - Taylor series expansion w.r.t. time only: AWA, VNODE
    - Taylor series expansion w.r.t. time and initial values: COSY VI
    - They mainly differ in the representation of the computed enclosures (intervals or Taylor models).
- On the other side, there are non validated integration methods:
    - Euler, Runge-Kutta,...
    - They have been intensively used for simulation and engineers often know how to tune them.

Outline

Interval Taylor series methods

We start from the Interval Initial Value problem:

$$\dot{y} = f(y), \quad y(t_0) \in [y_0] \tag{3.1}$$

- The goal of the integration is to compute a sequence of interval enclosures $[y_j]$ such that $y(t_j) \in [y_j]$.

We start from the real valued Taylor series expansion:

$$y_{j+1} = y_j + \sum_{k=1}^{N-1} f^{[k-1]}(y_j)h_j^k + h_j^N f^{[N-1]}\big(y(x_s)\big)$$

Interval Taylor series methods

We start from the Interval Initial Value problem:

$$\dot{y} = f(y), \quad y(t_0) \in [y_0] \tag{3.1}$$

- The goal of the integration is to compute a sequence of interval enclosures $[y_j]$ such that $y(t_j) \in [y_j]$.

We start from the real valued Taylor series expansion:

$$y_{j+1} = y_j + \sum_{k=1}^{N-1} f^{[k-1]}(y_j)h_j^k + h_j^N f^{[N-1]}(y(x_s))$$

A naive transformation of this formula into interval arithmetics gives:

$$[y_{j+1}] = [y_j] + \sum_{k=1}^{N-1} f^{[k-1]}([y_j])h_j^k + h_j^N f^{[N-1]}([\tilde{y}_j])$$

Interval Taylor series methods

- *Computation of* $[\tilde{y}_j]$: Picard-Lindelöf operator (or higher order methods).
- *Avoiding* $[y_j]$ *to grow*: we compute the interval evaluations with the mean value form:

$$\begin{aligned}
[y_{j+1}] &= \hat{y}_j + \sum_{k=1}^{N-1} f^{[k-1]}(\hat{y}_j) h_j^k + h_j^N f^{[N-1]}([\tilde{y}_j]) + \\
&\quad \left(I + \sum_{k=1}^{N-1} J(f^{[k-1]}, [y_j]) h_j^k\right)([y_j] - \hat{y}_j) \\
&= y_{j+1} + h_j^N f^{[N-1]}([\tilde{y}_j]) + S_j.([y_j] - \hat{y}_j)
\end{aligned}$$

Interval Taylor series methods

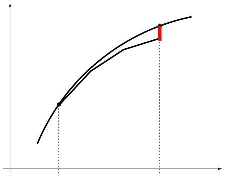
So, the enclosure at the next step is computed as the sum of :

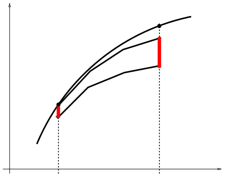
- a point: $y_{j+1} = \hat{y}_j + \sum_{k=1}^{N-1} f^{[k-1]}(\hat{y}_j) h_j^k$
- a local error term: $h_j^N f^{[N-1]}\big([\tilde{y}_j]\big)$
- an error propagation term: $S_j.([y_j] - \hat{y}_j)$

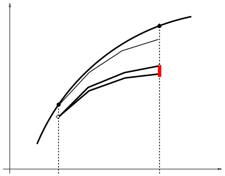
Wrapping effect occurs during the computation of the error propagation. To reduce it, you can use for example the **QR-factorization** method.

Our Method.

- *Compute the point approximation and the error independently.*
  - not validated approximation points are computed without any interval arithmetics.
  - errors are computed in a second time and compared to a user defined tolerance ϵ.
- The global error may be divided into three parts:
- Each error is computed independently:

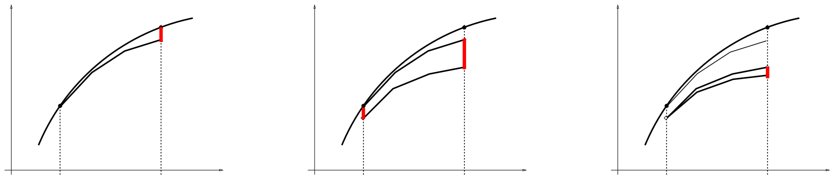






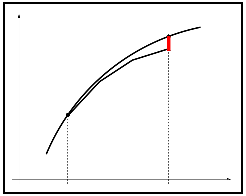
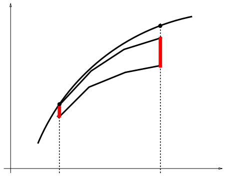
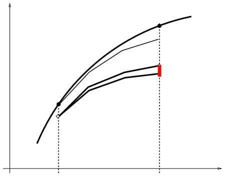
Our Method.

- Compute the point approximation and the error independently.
- *The global error may be divided into three parts:*
    - approximation error due to limitations of the method.
    - propagation of the previous error.
    - roundoff error due to machine finite precision.
- Each error is computed independently:



Our Method.

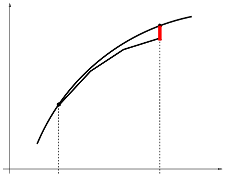
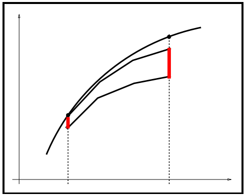
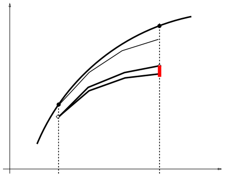
- Compute the point approximation and the error independently.
- The global error may be divided into three parts:
  - approximation error due to limitations of the method.
  - propagation of the previous error.
  - roundoff error due to machine finite precision.
- *Each error is computed independently:*

Picard-Lindelöf operator for method error.

Our Method.

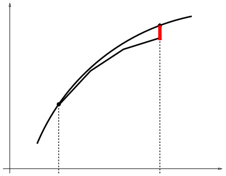
- Compute the point approximation and the error independently.
- The global error may be divided into three parts:
  - approximation error due to limitations of the method.
  - propagation of the previous error.
  - roundoff error due to machine finite precision.
- *Each error is computed independently:*

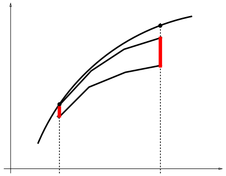




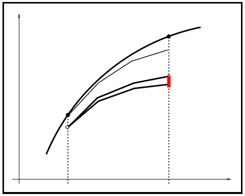
Löhner's factorization method for the propagation.

Our Method.

- Compute the point approximation and the error independently.
- The global error may be divided into three parts:
  - approximation error due to limitations of the method.
  - propagation of the previous error.
  - roundoff error due to machine finite precision.
- *Each error is computed independently:*







Global error arithmetics for the roundoff error.

Notations.

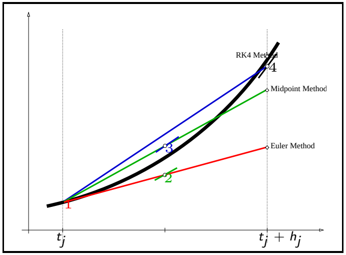
- Real numbers: $a \in \mathbb{R}$
- Floating point numbers: $\mathbf{a} \in \mathbb{F}$
- Intervals: $[a] = [\underline{a}, \overline{a}]$
- Floating point intervals: $\mathbf{[a]} = [\underline{\mathbf{a}}, \overline{\mathbf{a}}]$
- Initial value problem:

$$\dot{y} = f(y), \quad y(t_0) \in y_0 + [e_0] \quad \textit{with} \left\{ \begin{array}{l} y : \mathbb{R} \rightarrow \mathbb{R}^d \\ f : \mathbb{R}^d \rightarrow \mathbb{R}^d \end{array} \right. \tag{3.2}$$

Outline

The RK4 Method.

- Iterative method for computing approximation points of the solution of (3.2)
- Order 4 method, with adaptative step size control.
- Needs four evaluations of f for computing y_{j+1} out of y_j.

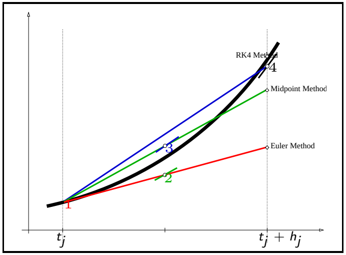


$$
\begin{aligned}
k_1 &= f(y_j) \\
k_2 &= f(y_j + h/2.k_1) \\
k_3 &= f(y_j + h/2.k_2) \\
k_4 &= f(y_j + hk_3) \\
y_{j+1} &= y_j + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)
\end{aligned}
$$

- The iteration of the scheme gives $\left(\mathbf{y_n}\right)_{n\in\mathbb{N}}$

The RK4 Method.

- Iterative method for computing approximation points of the solution of (3.2)
- Order 4 method, with adaptative step size control.
- Needs four evaluations of f for computing y_{j+1} out of y_j.



$$
\begin{aligned}
k_1 &= f(y_j) \\
k_2 &= f(y_j + h/2.k_1) \\
k_3 &= f(y_j + h/2.k_2) \\
k_4 &= f(y_j + hk_3) \\
y_{j+1} &= y_j + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

- The iteration of the scheme gives $\left(\mathbf{y_n}\right)_{n\in\mathbb{N}}$

$$\boxed{\textit{Goal:}\text{ Find an enclosure of } y(t_j) - \mathbf{y_j}.}$$

Some definitions.

We define the following functions:

$$\begin{aligned}
k_1(y,h) &= f(y)\\
k_2(y,h) &= f(y + h/2.k_1(y,h))\\
k_3(y,h) &= f(y + h/2.k_2(y,h))\\
k_4(y,h) &= f(y + hk_3(y,h))\\
\Phi(y,h) &= y + \frac{h}{6}\big(k_1(y,h) + 2k_2(y,h) + 2k_3(y,h) + k_4(y,h)\big)
\end{aligned}$$

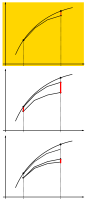
We then have:

$$y_{j+1} = \Phi(y_j, h_j)$$

We also define:

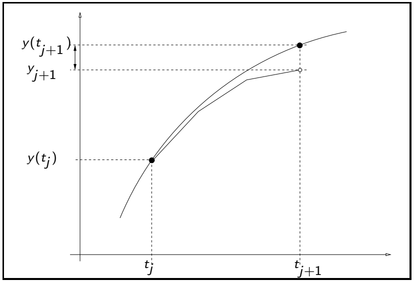
$$\varphi_j : t \mapsto \Phi\big(t - t_j, y(t_j)\big) \quad \psi_j : y \mapsto \Phi\big(h_j, y\big)$$

Computing the error: one step error

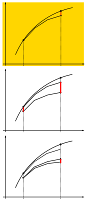


- Let us suppose that $y_j = y(t_j)$.
  - $y_{j+1} = \varphi_j(t_{j+1})$
  - $\forall i \in [0,4],\ \frac{d^i y}{dt^i}(t_j) = \frac{d^i \varphi_j}{dt^i}(t_j)$
  - Therefore, there exists $\xi \in [t_j, t_{j+1}]$ such that

$$y(t_{j+1}) - \varphi_j(t_{j+1}) = h_j^5 (y - \varphi_j)^{[5]}(\xi)$$

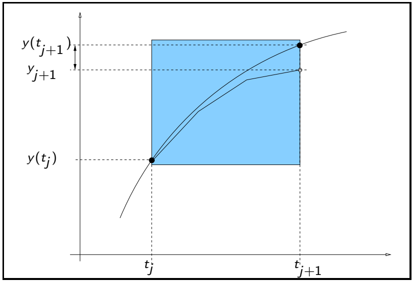


Computing the error: one step error

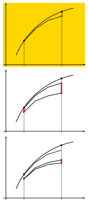


- Let us suppose that $y_j = y(t_j)$.
  - $y_{j+1} = \varphi_j(t_{j+1})$
  - $\forall i \in [0,4],\ \frac{d^i y}{dt^i}(t_j) = \frac{d^i \varphi_j}{dt^i}(t_j)$
  - Therefore, there exists $\xi \in [t_j, t_{j+1}]$ such that

$$y(t_{j+1}) - \varphi_j(t_{j+1}) = h_j^5 (y - \varphi_j)^{[5]}(\xi)$$



Computing the error: one step error (2)



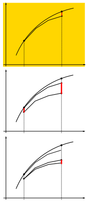
- We compute an apriori enclosure $[\tilde{y}_j]$ such that:

$$\forall t \in [t_j, t_{j+1}],\ y(t) \in [\tilde{y}_j]$$

- Picard-Lindelöf operator $P\big(R\big) = y_j + [0, h_j].f(R)$ or higher order methods.
  - Then, we have:

$$y(t_{j+1}) - \varphi_j(t_{j+1}) \in \frac{h_j^5}{120}\left(\frac{d^4 f}{dx^4}([\tilde{y}_j]) - \frac{d^5 \varphi_j}{dx^5}([t_j, t_{j+1}])\right)$$

Computing the error: one step error (2)



- We compute an apriori enclosure $[\tilde{y}_j]$ such that:

$$\forall t \in [t_j, t_{j+1}],\ y(t) \in [\tilde{y}_j]$$

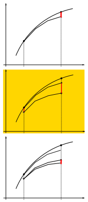
- Picard-Lindelöf operator $P(R) = y_j + [0, h_j].f(R)$ or higher order methods.
  - Then, we have:

$$y(t_{j+1}) - \varphi_j(t_{j+1}) \in \frac{h_j^5}{120}\left(\frac{d^4 f}{dx^4}([\tilde{y}_j]) - \frac{d^5\varphi_j}{dx^5}([t_j, t_{j+1}])\right)$$

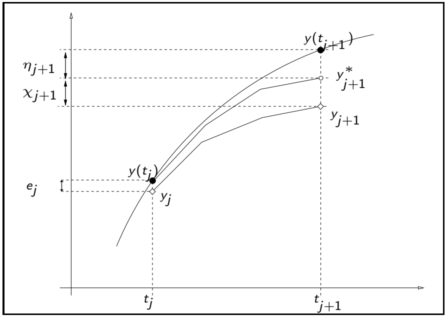
  - In Taylor series method, the local error is:

$$\frac{h_j^N}{N!}\frac{d^{N-1}f}{dx^{N-1}}([\tilde{y}_j])$$

Computing the error: propagation

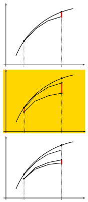


Now, what if there were an error: $y(t_j) \in \mathbf{y_j} + \mathbf{[e_j]}$



- We now have: $y_{j+1} = \psi_j(y_j)$ and $y^*_{j+1} = \psi_j(y(t_j))$
- So, $\boxed{y_{j+1} - y^*_{j+1} = Jac(\psi_j, \chi_j).\epsilon_j}$ with $\chi_j \in [y_j, y(t_j)]$

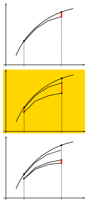
Computing the error: propagation



- This is overapproximated with interval arithmetic:

$$y_{j+1} - y^*_{j+1} \in Jac(\psi_j, y_j + [e_j]).[\epsilon_j]$$

Computing the error: propagation



- This is overapproximated with interval arithmetic:

$$y_{j+1} - y^*_{j+1} \in Jac(\psi_j, y_j + [e_j]).[\epsilon_j]$$

- In Taylor series method, the propagation of the previous error is given by:

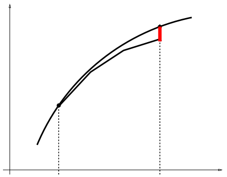
$$\left(I + \sum_{k=1}^{N-1} J(f^{[k-1]}, [y_j])h_j^k\right)([y_j] - \hat{y}_j)$$

- In both cases, the use of the QR preconditionning keeps the method stable.

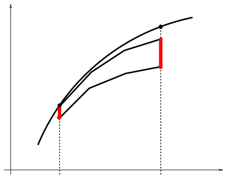
To sum up.

$$\boxed{\textit{Goal:}\ \text{give a rigourous bound on } y(t_j) - \mathbf{y_j}}$$

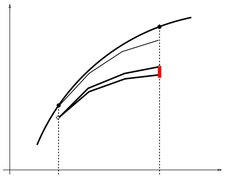
$$\begin{aligned}
y(t_{j+1}) &= \mathbf{y_{j+1}} + (y(t_{j+1}) - y^*_{j+1}) + (y^*_{j+1} - \mathbf{y_{j+1}})) \\
&= \mathbf{y_{j+1}} + (y - \varphi_j)^{[5]}(\xi) + (y^*_{j+1} - y_{j+1}) - E_{j+1} \\
\mathbf{[e_{j+1}]} &\in (y - \varphi_j)^{[5]}(\mathbf{[R]}) + Jac(\psi_j, \mathbf{y_j} + \mathbf{[e_j]}).\mathbf{[e_j]} - \mathbf{[E_{j+1}]}
\end{aligned}$$



$(y - \varphi_j)^{[5]} dt^5(\mathbf{[R]})$



$Jac(\psi_j, \mathbf{y_j} + \mathbf{[e_j]}).\mathbf{[e_j]}$



$\mathbf{[E_{j+1}]}$

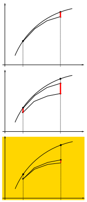
To sum up.

$\boxed{\textit{Goal:}\text{ give a rigourous bound on } y(t_j) - \mathbf{y_j}}$

$$\begin{aligned} y(t_{j+1}) &= \mathbf{y_{j+1}} + (y(t_{j+1}) - y^*_{j+1}) + (y^*_{j+1} - \mathbf{y_{j+1}})) \\ &= \mathbf{y_{j+1}} + (y - \varphi_j)^{[5]}(\xi) + (y^*_{j+1} - y_{j+1}) - E_{j+1} \\ \mathbf{[e_{j+1}]} &\in (y - \varphi_j)^{[5]}(\mathbf{[R]}) + Jac(\psi_j, \mathbf{y_j} + \mathbf{[e_j]}).\mathbf{[e_j]} - \mathbf{[E_{j+1}]} \end{aligned}$$

- Implementation issues:
  - Löhner's QR-factorization method for reducing the wrapping effect
  - Overapproximation of $\mathbf{[E_j]}$: we use the *global error arithmetics*

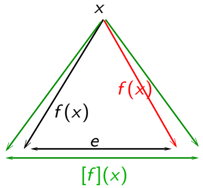
Computing the error : round-off error.



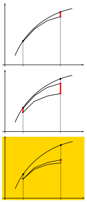
$$a = f_a + e_a\overrightarrow{\varepsilon_e} \quad and \quad b = f_b + e_b\overrightarrow{\varepsilon_e}$$

$$\begin{aligned}
a + b &= \uparrow_\sim (f_a + f_b) + (e_a + e_b + \downarrow_\sim (f_a + f_b))\overrightarrow{\varepsilon_e} \\
a - b &= \uparrow_\sim (f_a - f_b) + (e_a - e_b + \downarrow_\sim (f_a - f_b))\overrightarrow{\varepsilon_e} \\
a \times b &= \uparrow_\sim (f_a \times f_b) + (e_a f_b + e_b f_a + e_a e_b + \downarrow_\sim (f_a \times f_b))\overrightarrow{\varepsilon_e}
\end{aligned}$$

- Let the user know both the result (f) and its distance to the real result.



Computing the error : round-off error.



Suppose that we are working on a 4 digits machine. We have two global error numbers, $a = 621.3 + 0.05\overrightarrow{\varepsilon_e}$ and $b = 1.287 + 0.0005\overrightarrow{\varepsilon_e}$, that we want to multiply.

	621.3	$+$	$0.05\overrightarrow{\varepsilon_e}$	a
$\times$	1.287	$+$	$0.0005\overrightarrow{\varepsilon_e}$	b
$=$	799.6131			Real result
		$+$	$0.06435\overrightarrow{\varepsilon_e}$	Error due to a
		$+$	$0.31065\overrightarrow{\varepsilon_e}$	Error due to b
		$+$	$0.000025\overrightarrow{\varepsilon_e}$	Second order term
$=$	799.6ε			Floating point result
				$=\uparrow_\sim (f_a \times f_b)$
		$+$	$0.375025\overrightarrow{\varepsilon_e}$	
		$+$	$0.0131\overrightarrow{\varepsilon_e}$	$\downarrow_\sim (f_a \times f_b)$
$=$	799.6ε	$+$	$0.388[1,2]\overrightarrow{\varepsilon_e}$	

Outline

Numerical Results.

- The method has been implemented in a library GRKlib:
  - use formal derivation techniques for computing the derivatives.
  - propagates separately method and round off errors.
  - can be used with both double and multiprecision arithmetics.
  - only implements order 4 Runge-Kutta formula.
- Tried it on various problems:
  -

Numerical Results.

- The method has been implemented in a library GRKlib:
    - use formal derivation techniques for computing the derivatives.
    - propagates separately method and round off errors.
    - can be used with both double and multiprecision arithmetics.
    - only implements order 4 Runge-Kutta formula.
- Tried it on various problems:
    - Linear problem
      Simple rotation:

$$\dot{Y} = \begin{pmatrix} 0 & -0.707107 & -0.5 \\ 0.707107 & 0 & 0.5 \\ 0.5 & 0 & -0.5 \end{pmatrix} Y$$

$t =$	100	500	1000
ϵ	4.10^{-4}	2.10^{-3}	4.10^{-3}

Numerical Results.

- The method has been implemented in a library GRKlib:
  - use formal derivation techniques for computing the derivatives.
  - propagates separately method and round off errors.
  - can be used with both double and multiprecision arithmetics.
  - only implements order 4 Runge-Kutta formula.
- Tried it on various problems:
  - Linear problem
    Simple contraction:

$$\dot{Y} = \begin{pmatrix} -0.4375 & 0.0625 & -0.265165 \\ 0.0625 & -0.4375 & -0.265165 \\ -0.265165 & -0.265165 & -0.375 \end{pmatrix} Y$$

$t =$	100	500	1000
ϵ	3.10^{-5}	3.10^{-5}	$3,3.10^{-5}$

Numerical Results.

- The method has been implemented in a library GRKlib:
  - use formal derivation techniques for computing the derivatives.
  - propagates separately method and round off errors.
  - can be used with both double and multiprecision arithmetics.
  - only implements order 4 Runge-Kutta formula.
- Tried it on various problems:
  - Non linear problem
    Lorenz equations:

$$\begin{cases} \dot{y_1} = 10(y_2 - y_1) \\ \dot{y_2} = y_1(28 - y_3) - y_2 \\ \dot{y_3} = y_1 * y_2 - \frac{8}{3}y_3 \end{cases}$$

$t =$	5	10	15
ϵ	2.10^{-8}	4.10^{-5}	6.10^{-4}

Conclusion.

In this talk, we:

- showed how to make a validated integration method out of a Runge-Kutta integration scheme.
- informally compared the formulae for the error with the ones from Taylor series method.

Our implementation shows that we can achieve good precision results, although only order 4 method is used.

Conclusion.

In this talk, we:

- showed how to make a validated integration method out of a Runge-Kutta integration scheme.
- informally compared the formulae for the error with the ones from Taylor series method.

Our implementation shows that we can achieve good precision results, although only order 4 method is used.

- Advantage of the method:
  - based on well known numerical method (Runge-Kutta), which can be finely tuned for every problem.
  - it allows effective step size control, with ideas coming from control theory.

Conclusion.

In this talk, we:

- showed how to make a validated integration method out of a Runge-Kutta integration scheme.
- informally compared the formulae for the error with the ones from Taylor series method.

Our implementation shows that we can achieve good precision results, although only order 4 method is used.

- Advantage of the method:
  - based on well known numerical method (Runge-Kutta), which can be finely tuned for every problem.
  - it allows effective step size control, with ideas coming from control theory.
- Further work:
  - add other integration schemes to our library (order 5/6 RK methods).
  - use better domains for the representation of the error to reduce wrapping effect (Taylor models).