heyoka: Modern Numerical Integration and its uses in Astrodynamics and Spaceflight Mechanics

Francesco Biscani

September 14, 2022

European Space Operations Centre (ESOC)

What is heyoka anyway?

heyoka is a software package for the numerical integration of ODEs (ordinary differential equations) via Taylor's method

https://github.com/bluescarni/heyoka
https://github.com/bluescarni/heyoka.py

Heyoka

Use in ACT projects

- G&CNETs + Backward Generation of Optimal Examples
- Poincaré maps classification
- Dark matter inversion
- EclipseNET
- Kelvins space debris competition

Why should I care?! I can use ...

Main motivations

Only one actively-maintained implementation of Taylor's method

Superior performance (speed & accuracy)

Unique features enabling novel applications

Taylor's method

Initial value problem:

$$\begin{cases} x'(t) & = f(x(t)) \\ x(t_0) & = x_0 \end{cases}$$

Taylor's method

Initial value problem:

$$\begin{cases} x'(t) &= f(x(t)) \\ x(t_0) &= x_0 \end{cases}$$

Construct the Taylor series of the solution around $t = t_0$:

$$x(t_1) = x(t_0) + x'(t_0)(t_1 - t_0) + \frac{1}{2}x''(t_0)(t_1 - t_0)^2 + \dots$$

Taylor's method

Initial value problem:

$$\begin{cases} x'(t) &= f(x(t)) \\ x(t_0) &= x_0 \end{cases}$$

Construct the Taylor series of the solution around $t = t_0$:

$$x(t_1) = x(t_0) + x'(t_0)(t_1 - t_0) + \frac{1}{2}x''(t_0)(t_1 - t_0)^2 + \ldots$$

The integration error can be constrained in a mathematically-rigorous way

Taylor's method

Initial value problem:

$$\begin{cases} x'(t) & = f(x(t)) \\ x(t_0) & = x_0 \end{cases}$$

Construct the Taylor series of the solution around $t = t_0$:

$$x(t_1) = x(t_0) + x'(t_0)(t_1 - t_0) + \frac{1}{2}x''(t_0)(t_1 - t_0)^2 + \ldots$$

The integration error can be constrained in a mathematically-rigorous way

Free dense output with guaranteed precision

Technical difficulties

How to compute the Taylor coefficients?

Technical difficulties

How to compute the Taylor coefficients?

Basic recursion:

$$\begin{aligned} x'(t) &= f(x(t)) \\ x''(t) &= \frac{\partial f(x(t))}{\partial x} x'(t) \\ &\dots \end{aligned}$$

Technical difficulties

How to compute the Taylor coefficients?

Basic recursion:

$$\begin{aligned} x'(t) &= f(x(t)) \\ x''(t) &= \frac{\partial f(x(t))}{\partial x} x'(t) \\ &\dots \end{aligned}$$

Symbolic differentiation: cumbersome, error-prone, exponential complexity

Technical difficulties

How to compute the Taylor coefficients?

Basic recursion:

$$
\begin{aligned}
x'(t) &= f(x(t)) \\
x''(t) &= \frac{\partial f(x(t))}{\partial x} x'(t) \\
&\dots
\end{aligned}
$$

Symbolic differentiation: cumbersome, error-prone, exponential complexity

Automatic differentiation (AD) to the rescue:

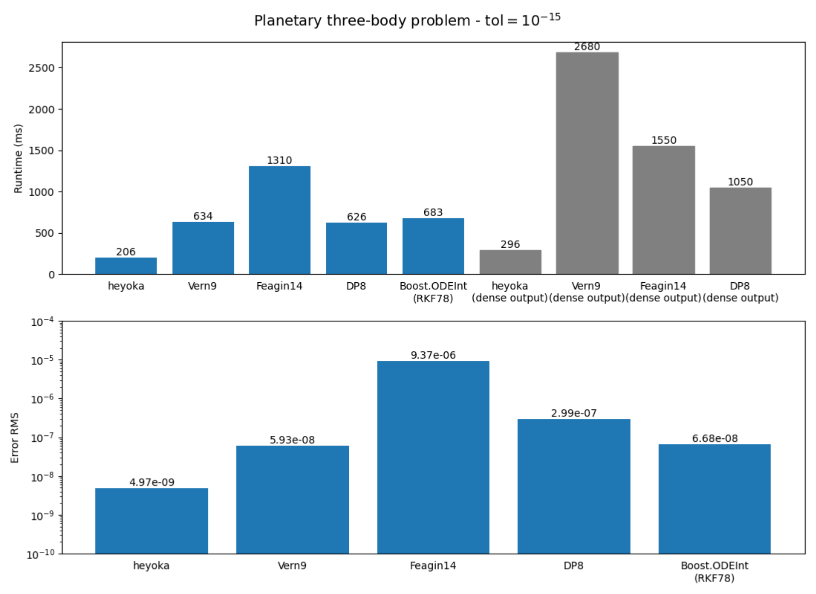
1. decompose $f(x(t))$ into a graph of elementary subexpressions
2. apply AD rules on the subexpressions

However, the implementation is technically very challenging

heyoka – The sales pitch

- Novel implementation of Taylor's method based on a just-in-time (JIT) compilation approach
- Batch mode to fully utilise modern vector instruction sets
- Support for coarse-grained and fine-grained automatic parallelisation
- Support for extended-precision arithmetic
- Optimally accurate
- Support for reliable and accurate event-detection

Obligatory benchmark slide



Event detection

Detect specific conditions in the state of a system

Event detection

Detect specific conditions in the state of a system

Examples: collisions, spacecraft eclipse, Poincaré maps, …

Event detection

Detect specific conditions in the state of a system

Examples: collisions, spacecraft eclipse, Poincaré maps, ...

Events defined by an event equation: $g(x) = 0$

Event detection

Detect specific conditions in the state of a system

Examples: collisions, spacecraft eclipse, Poincaré maps, ...

Events defined by an event equation: $g(\boldsymbol{x}) = 0$

E.g., collision of two spheres:

$$(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2 - 4R^2 = 0$$

Event detection

Detect specific conditions in the state of a system

Examples: collisions, spacecraft eclipse, Poincaré maps, ...

Events defined by an event equation: $g(\mathbf{x}) = 0$

E.g., collision of two spheres:

$$(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2 - 4R^2 = 0$$

Existing approaches leave much to be desired ...

There's gotta be a better way...

Event detection in Taylor's method

Taylor series of the event function (via AD):

$$g(t) = g(t_0) + g'(t_0)(t - t_0) + \frac{1}{2} g''(t_0)(t - t_0)^2 + \ldots$$

Event detection in Taylor's method

Taylor series of the event function (via AD):

$$g(t) = g(t_0) + g'(t_0)(t - t_0) + \frac{1}{2} g''(t_0)(t - t_0)^2 + \dots$$

Continuous, high-fidelity approximation of the event function as a time polynomial

Event detection in Taylor's method

Taylor series of the event function (via AD):

$$g\left(t\right)=g\left(t_0\right)+g'\left(t_0\right)\left(t-t_0\right)+\frac{1}{2}g''\left(t_0\right)\left(t-t_0\right)^2+\ldots$$

Continuous, high-fidelity approximation of the event function as a time polynomial

Rigorous polynomial root-finding algorithms to locate an event's tigger time

Event detection in Taylor's method

Taylor series of the event function (via AD):

$$g(t) = g(t_0) + g'(t_0)(t - t_0) + \frac{1}{2}g''(t_0)(t - t_0)^2 + \dots$$

Continuous, high-fidelity approximation of the event function as a time polynomial

Rigorous polynomial root-finding algorithms to locate an event's tigger time

Use dense output to propagate the state of the system up to the trigger time

Sneak peek - application to space debris

Goal: detection of collisions and/or close encounters

Sneak peek - application to space debris

Goal: detection of collisions and/or close encounters

Main challenge: small bodies, moving fast

Sneak peek - application to space debris

Goal: detection of collisions and/or close encounters

Main challenge: small bodies, moving fast

Preliminary results:

- Guaranteed collision detection for cm-sized objects moving at km/s speeds
- No constraints on the integration timestep
- $\sim$ 4 hours for $\sim$ 20k objects, 20 years (64 cores)

Sneak peek - application to space debris

Goal: detection of collisions and/or close encounters

Main challenge: small bodies, moving fast

Preliminary results:

- Guaranteed collision detection for cm-sized objects moving at km/s speeds
- No constraints on the integration timestep
- ~ 4 hours for ~ 20k objects, 20 years (64 cores)

Still a work-in-progress...

Questions?