

Bounded evolution of a set

Luca Geretti

University of Verona, Italy



Outline

Hybrid evolution of a set

Continuous step

1. From the starting set, given a time step h , construct the flow set
2. Apply to the whole $[0, h]$ time interval to get the reached set
3. Apply to the h time value to get the finishing set

Hybrid evolution of a set

Continuous step

1. From the starting set, given a time step h , construct the flow set
2. Apply to the whole $[0, h]$ time interval to get the reached set
3. Apply to the h time value to get the finishing set

Discrete step

1. From the flow set, identify the presence of crossings with guards
2. Compute the kinds of crossings
3. Compute the crossing times
4. Apply the constraints for the intersection
5. Apply the resets

Outline

How to construct a flow set

We need to find a set Φ for which the **Picard operator** applied on itself is a contraction:

$$\Phi_{k+1}(x, t) = x + \int_0^t f(\Phi_k(x, s)) ds \subseteq \Phi_k(x, t), \quad \forall x \in X_0$$

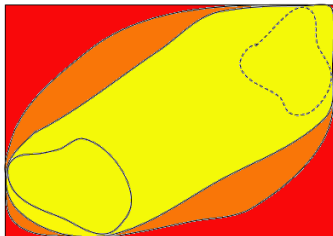
where f is the ODE function and X_0 is the starting set.

There are two approaches:

1. Get a bounding for Φ and refine Φ until satisfied with the result;
2. Construct a polynomial with desired order for Φ , then find a uniform error e for which the contraction is satisfied.

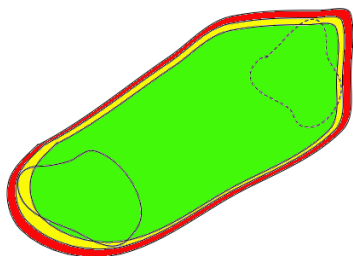
How to construct a flow set - Method 1

1. Use the **Euler Method** on the bounding box of X_0 , called X_0^{bb} : if the result is contained in X_0^{bb} , then it is a valid Φ_0 .
 2. Use the Picard operator, integrating using **automatic differentiation**, to refine Φ from Φ_0 as many times as desired.
- The temporal order increases, the uniform error decreases.



How to construct a flow set - Method 2

1. Construct a polynomial expansion for Φ called p , using automatic differentiation;
 2. Try a value of the uniform error e such that $p + e$ is a contraction:
 - ▶ If successful, repeat 2. using the uniform error obtained from the contraction (until satisfied with the result);
 - ▶ If not, try with a larger value of e .
- The temporal order is fixed, the uniform error decreases.



Pros and cons of each method

Method 1

- + We get flow bounds (including the step size) early;
- The temporal order must grow to reduce the error;
- Being low-order, the Euler Method might be incapable of verifying a step size for which a contraction is actually possible.

Pros and cons of each method

Method 1

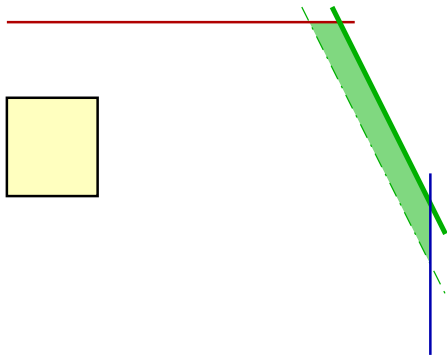
- + We get flow bounds (including the step size) early;
- The temporal order must grow to reduce the error;
- Being low-order, the Euler Method might be incapable of verifying a step size for which a contraction is actually possible.

Method 2

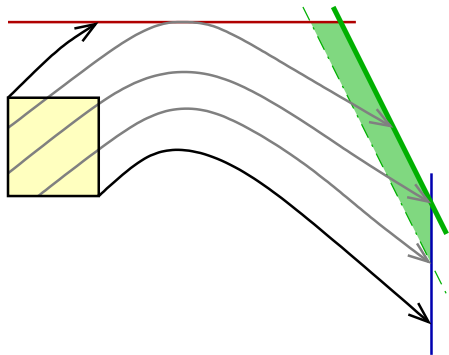
- + It decouples the temporal order from the error;
- It chooses the temporal order in advance, hence we can't adapt the order to account for local contractibility properties;
- It is more expensive to compute in general.

Outline

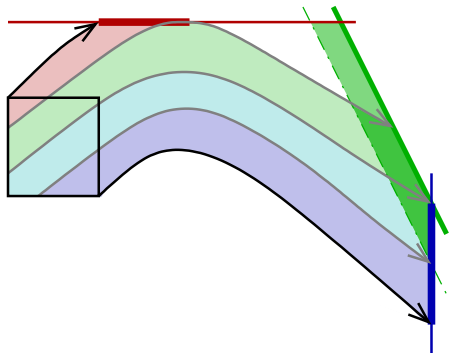
A graphical overview of discrete steps



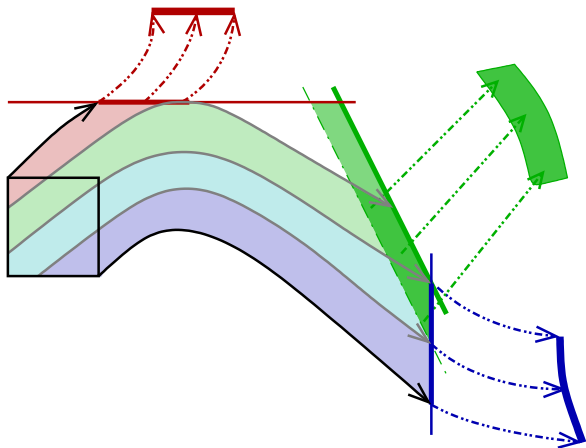
A graphical overview of discrete steps



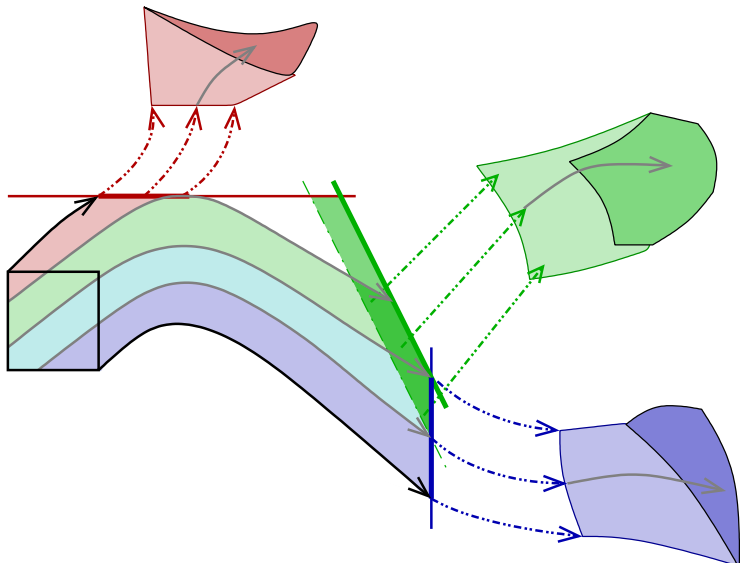
A graphical overview of discrete steps



A graphical overview of discrete steps



A graphical overview of discrete steps



Identify the presence of crossings

In order to identify crossings between a guard function $g : \mathbb{R}^m \rightarrow \mathbb{R}$ and points subject to a (nonlinear) vector field function $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$, the **Lie derivative** is fundamental:

$$L_f g = \nabla g \cdot f = \sum_{i=1}^m \frac{\partial g}{\partial x_i} f_i$$

Identify the presence of crossings

In order to identify crossings between a guard function $g : \mathbb{R}^m \rightarrow \mathbb{R}$ and points subject to a (nonlinear) vector field function $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$, the **Lie derivative** is fundamental:

$$L_f g = \nabla g \cdot f = \sum_{i=1}^m \frac{\partial g}{\partial x_i} f_i$$

When $L_f g$ is **evaluated** for the reached set s after one continuous step, we get an **interval** for which we can conclude the following for the event e associated to the guard:

- $L_f g(s) < 0$: inactive event (no crossing);

Identify the presence of crossings

In order to identify crossings between a guard function $g : \mathbb{R}^m \rightarrow \mathbb{R}$ and points subject to a (nonlinear) vector field function $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$, the **Lie derivative** is fundamental:

$$L_f g = \nabla g \cdot f = \sum_{i=1}^m \frac{\partial g}{\partial x_i} f_i$$

When $L_f g$ is **evaluated** for the reached set s after one continuous step, we get an **interval** for which we can conclude the following for the event e associated to the guard:

- $L_f g(s) < 0$: inactive event (no crossing);
- $L_f g(s) > 0$: initially active event (already completely crossed);

Identify the presence of crossings

In order to identify crossings between a guard function $g : \mathbb{R}^m \rightarrow \mathbb{R}$ and points subject to a (nonlinear) vector field function $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$, the **Lie derivative** is fundamental:

$$L_f g = \nabla g \cdot f = \sum_{i=1}^m \frac{\partial g}{\partial x_i} f_i$$

When $L_f g$ is **evaluated** for the reached set s after one continuous step, we get an **interval** for which we can conclude the following for the event e associated to the guard:

- $L_f g(s) < 0$: inactive event (no crossing);
- $L_f g(s) > 0$: initially active event (already completely crossed);
- $L_f g(s)$ crosses 0: partially active (there exists a crossing at a time $0 < \gamma < \delta$, with δ the time step size).

When the concavity is relevant

If no definite positivity/negativity is gathered from $L_f g(s)$, then we can resort to the second derivative: $L_f^2 g = L_f L_f g$.

- $L_f^2 g(s) > 0$: convex, which means that the event is either:
 - ▶ initially active;
 - ▶ never active;
 - ▶ not initially active but becomes active due to a transverse crossing;
 - ▶ the initial set is on the boundary of the guard set, possibly with the flow tangent to this set.

When the concavity is relevant

If no definite positivity/negativity is gathered from $L_f g(s)$, then we can resort to the second derivative: $L_f^2 g = L_f L_f g$.

- $L_f^2 g(s) > 0$: convex, which means that the event is either:
 - ▶ initially active;
 - ▶ never active;
 - ▶ not initially active but becomes active due to a transverse crossing;
 - ▶ the initial set is on the boundary of the guard set, possibly with the flow tangent to this set.
- $L_f^2 g(s) < 0$: concave, which means that the event is either:
 - ▶ initially active;
 - ▶ not initially active, but becomes active later;
 - ▶ never active, and the maximum of the guard along the flow lines is zero;
 - ▶ the state touches the guard at a point of tangency.

When the concavity is relevant

If no definite positivity/negativity is gathered from $L_f g(s)$, then we can resort to the second derivative: $L_f^2 g = L_f L_f g$.

- $L_f^2 g(s) > 0$: convex, which means that the event is either:
 - ▶ initially active;
 - ▶ never active;
 - ▶ not initially active but becomes active due to a transverse crossing;
 - ▶ the initial set is on the boundary of the guard set, possibly with the flow tangent to this set.
- $L_f^2 g(s) < 0$: concave, which means that the event is either:
 - ▶ initially active;
 - ▶ not initially active, but becomes active later;
 - ▶ never active, and the maximum of the guard along the flow lines is zero;
 - ▶ the state touches the guard at a point of tangency.
- $L_f^2 g(s)$ crosses 0: degenerate, can't say anything.

Compute the crossing time

- A crossing time is necessary to associate an accurate evolution time to the crossing set

Compute the crossing time

- A crossing time is necessary to associate an accurate evolution time to the crossing set
- It is also useful to discriminate in which case we lie when concavity is relevant

Compute the crossing time

- A crossing time is necessary to associate an accurate evolution time to the crossing set
- It is also useful to discriminate in which case we lie when concavity is relevant
- It is computed as $g \circ \phi(D)$, with ϕ the flow function and D the flow+time domain

Compute the crossing time

- A crossing time is necessary to associate an accurate evolution time to the crossing set
- It is also useful to discriminate in which case we lie when concavity is relevant
- It is computed as $g \circ \phi(D)$, with ϕ the flow function and D the flow+time domain

Hence in Ariadne the evolution time is a **function**, possibly dependent on spacial parameters, in order to accommodate the fact that points that cross may do it at different times.

Compute the crossing time

- A crossing time is necessary to associate an accurate evolution time to the crossing set
- It is also useful to discriminate in which case we lie when concavity is relevant
- It is computed as $g \circ \phi(D)$, with ϕ the flow function and D the flow+time domain

Hence in Ariadne the evolution time is a **function**, possibly dependent on spacial parameters, in order to accommodate the fact that points that cross may do it at different times.

- Other tools might settle with an **upper bound** on the evolution time or a **simple interval**, which however do not allow to split sets along time anymore or to introduce constraints that depend on time.

- The crossing kind and time direct us to the operations required to obtain the evolved set for each event.

Summarizing

- The crossing kind and time direct us to the operations required to obtain the evolved set for each event.
- Unfortunately, based on the actual shape of the crossing and the set, computing the crossing time **may fail** numerically, which requires to introduce **constraints** in the set to account for the crossing, which delegates resolution to a later time
 - ▶ It is possible that constraints along evolution can be resolved better, numerically.

- Collins, P.; Bresolin, D.; Geretti, L.; Villa, T. "Computing the evolution of hybrid systems using rigorous function calculus", 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'12), June 2012, pg. 284-290

Outline

Continuous evolution on the van der Pol oscillator

- Found at `examples/continuous/vanderpol.cpp`
 1. Run with verbosity 2
 2. `set_animated(true)` and use the Gnuplot backend
 3. Run with verbosity 3, commenting simulation and graphics
 4. Run with verbosity 4 up to 7
 5. Change the integrator's maximum error per step
 6. Change the evolver's step size
 7. Enlarge the initial set
 8. Change the integrator with `GradedTaylorSeriesIntegrator`

Hybrid evolution on the watertank system

■ Found at

`tutorials/hybrid_evolution/hybrid_evolution_tutorial.cpp`

1. Run with verbosity 3, commenting `simulate_evolution` and `compute_reachability` and disabling graphics with the `-d none -g none` command line flags
2. Change `delta` to zero to remove nondeterminism
3. Increase the maximum step size
4. Run with verbosity 4 and the flags above