Tube Programming
Applied to State Estimation

Simon Rohou[1], Luc Jaulin[1], Lyudmila Mihaylova[2],
Fabrice Le Bars[1], Sandor M. Veres[2]

[1] ENSTA Bretagne, Lab-STICC, UMR CNRS 6285, France
[2] University of Sheffield, Western Bank Sheffield S10 2TN, UK
simon.rohou@ensta-bretagne.org

Symposium on Scientific Computing,
Computer Arithmetics and Verified Numerics
September 2016

Section 1

Introduction

Introduction

Guaranteed integration

Problem:
We consider the problem of guaranteed integration of differential state equations defined as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{n}(t)$$

Where:

- $t \in \mathbb{R}$ is the time
- $\mathbf{x}(t)$ is the state vector
- $\mathbf{n}(t)$ is the noise vector, assumed to belong to a known box $[\mathbf{n}]$
- $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is the *evolution* function

Introduction

Guaranteed integration

Problem:
We consider the problem of guaranteed integration of differential state equations defined as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{n}(t)$$

Existing methods:

- initial box $[\mathbf{x}](0)$ known
- methods based on Euler, Runge-Kutta or Taylor integration
- validation using the Picard Theorem
- available libraries: Cosy, Vnode, DynIbex, Capd

Introduction

Guaranteed integration

Problem:
We consider the problem of guaranteed integration of differential state equations defined as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{n}(t)$$

Proposed method:

- **constraint-based** approach for guaranteed interval integration

Introduction

Constraint Network based on trajectories

A problem involving constraints is classically presented with a **Constraint Network**:

[Mackworth1977], [Chabert2009]

Variables:

Constraints:

Domains:

Introduction

Constraint Network based on trajectories

A problem involving constraints is classically presented with a **Constraint Network**:

[Mackworth1977], [Chabert2009]

$$\begin{cases} \textbf{Variables:} \\ \textbf{Constraints:} \\ \\ \\ \textbf{Domains:} \end{cases}$$

Our approach consists in using:

Introduction

Constraint Network based on trajectories

A problem involving constraints is classically presented with a **Constraint Network**:

[Mackworth1977], [Chabert2009]

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot) \\ \textbf{Constraints:} \\ \\ \\ \textbf{Domains:} \end{cases}$$

Our approach consists in using:

- **trajectories** as variables: $\mathbf{x}(\cdot)$

Introduction

Constraint Network based on trajectories

A problem involving constraints is classically presented with a **Constraint Network**:

[Mackworth1977], [Chabert2009]

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot) \\ \textbf{Constraints: } \\ \\ \\ \textbf{Domains: } [\mathbf{x}](\cdot) \end{cases}$$

Our approach consists in using:

- **trajectories** as variables: $\mathbf{x}(\cdot)$
- **tubes** as domains: $\mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)$

Introduction

Constraint Network based on trajectories

A problem involving constraints is classically presented with a **Constraint Network**:

[Mackworth1977], [Chabert2009]

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot) \\ \textbf{Constraints:} \\ \quad - \quad \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{n}(t) \\ \quad - \quad \dots \\ \textbf{Domains: } [\mathbf{x}](\cdot) \end{cases}$$

Our approach consists in using:

- **trajectories** as variables: $\mathbf{x}(\cdot)$
- **tubes** as domains: $\mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)$

Now we can consider constraints defined with **differential equations** or any other **non-linear operation** on trajectories.

Section 2

Tube Programming

Tube Programming

Tubes: definition

Tube $[x](\cdot)$**:** interval of functions $[x^-, x^+]$ such that $\forall t \in \mathbb{R},\ x^-(t) \leqslant x^+(t)$

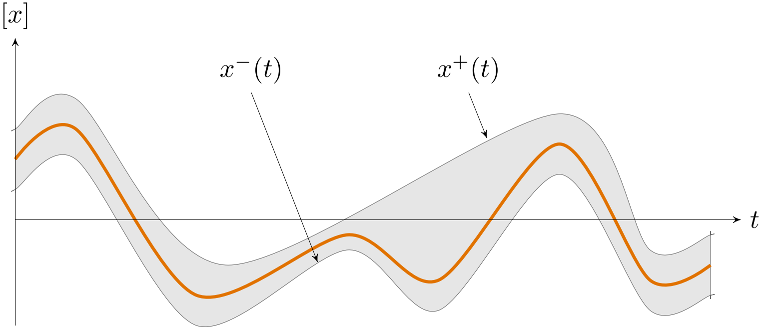


Figure: tube $[x](\cdot)$ enclosing an uncertain trajectory $x^*(\cdot)$

Tube Programming

Tubes: implementation

The computer representation of a tube encloses $[x^-(\cdot), x^+(\cdot)]$ inside an interval of step functions $[\underline{x^-}(\cdot), \overline{x^+}(\cdot)]$ such that:

$$\forall t \in \mathbb{R},\ \underline{x^-}(t) \leqslant x^-(t) \leqslant x^+(t) \leqslant \overline{x^+}(t)$$

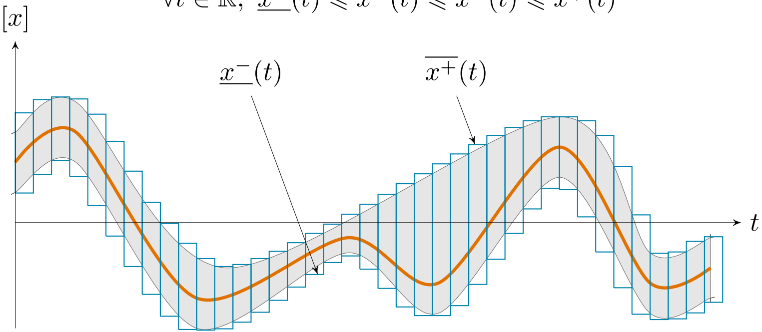


Figure: tube implementation with a set of boxes – this outer representation adds pessimism but enables guaranteed and simple computations

Tube Programming

Tubes: integral

Definition: the integral of a tube $[x](\cdot)=[x^-,x^+]$ is an interval:

$$\int_a^b [x](\tau)d\tau = \left\{\int_a^b x(\tau)d\tau \mid x \in [x]\right\} = \left[\int_a^b x^-(\tau)d\tau, \int_a^b x^+(\tau)d\tau\right]$$

[Aubry2013]

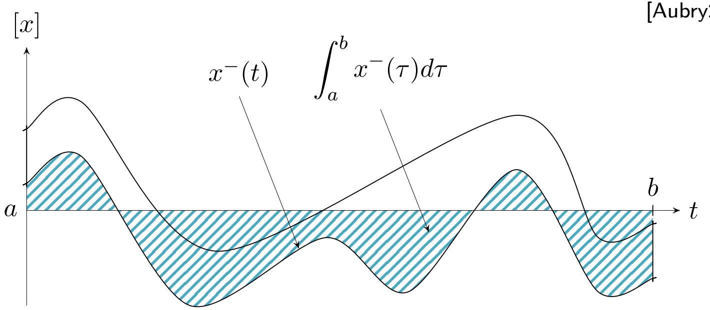


Figure: blue area: lower bound of the tube's integral

Tube Programming

Tubes: integral

Definition: the integral of a tube $[x](\cdot) = [x^-, x^+]$ is an interval:

$$\int_a^b [x](\tau)d\tau = \left\{ \int_a^b x(\tau)d\tau \mid x \in [x] \right\} = \left[\int_a^b x^-(\tau)d\tau, \int_a^b x^+(\tau)d\tau \right]$$

[Aubry2013]

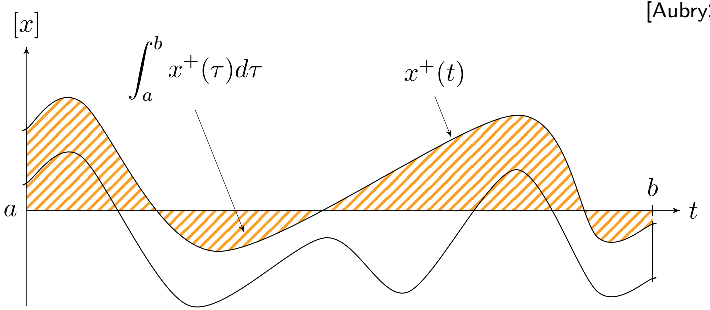


Figure: orange area: upper bound of the tube's integral

Tube Programming

Tubes: integral and its implementation

Implementation: an outer approximation of the integral is computed

$$\int_a^b [x](\tau)d\tau \subset \left[\int_a^b \underline{x^-}(\tau)d\tau, \int_a^b \overline{x^+}(\tau)d\tau\right]$$

[Aubry2013]

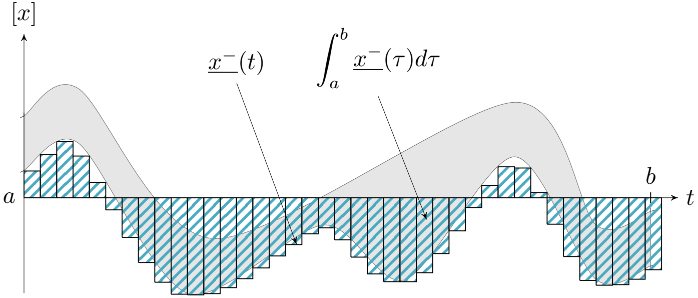


Figure: blue area: outer approximation of the lower bound of the tube's integral

Tube Programming

Tubes: integral and its implementation

Implementation: an outer approximation of the integral is computed

$$\int_a^b [x](\tau)d\tau \subset \left[\int_a^b \underline{x^-}(\tau)d\tau, \int_a^b \overline{x^+}(\tau)d\tau\right]$$

[Aubry2013]

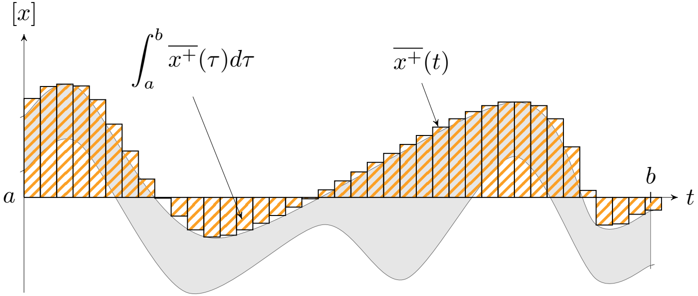


Figure: red area: outer approximation of the upper bound of the tube's integral

Tube Programming

Tubes: arithmetic and contractors

Example:

Tube arithmetic makes it possible to compute the following tubes:

$$\begin{aligned}
[a](\cdot) &= [x](\cdot) + [y](\cdot) \\
[b](\cdot) &= \sin\big([x](\cdot)\big) \\
[c](\cdot) &= \int_0 [x](\tau)d\tau
\end{aligned}$$

Tube Programming

Tubes: arithmetic and contractors

Example:
Tube arithmetic makes it possible to compute the following tubes:

$$\begin{aligned} [a](\cdot) &= [x](\cdot) + [y](\cdot) \\ [b](\cdot) &= \sin\big([x](\cdot)\big) \\ [c](\cdot) &= \int_0 [x](\tau)d\tau \end{aligned}$$

Contractors for tubes:
To each primitive constraint on trajectories, tubes are contracted without removing any feasible solution.

Tube Programming

Tubes: minimal and non-minimal contractors

Example:

The minimal contractor associated to the constraint $a(\cdot) = x(\cdot) + y(\cdot)$:

$$\begin{pmatrix} [a](\cdot) \\ [x](\cdot) \\ [y](\cdot) \end{pmatrix} \mapsto \begin{pmatrix} [a](\cdot) \cap ([x](\cdot) + [y](\cdot)) \\ [x](\cdot) \cap ([a](\cdot) - [y](\cdot)) \\ [y](\cdot) \cap ([a](\cdot) - [x](\cdot)) \end{pmatrix}$$

Tube Programming

Tubes: minimal and non-minimal contractors

Example:
The minimal contractor associated to the constraint $a(\cdot) = x(\cdot) + y(\cdot)$:

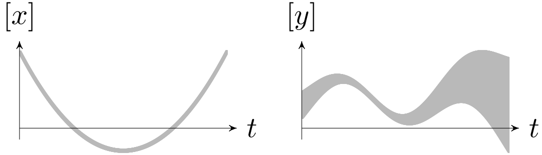
$$\begin{pmatrix} [a](\cdot) \\ [x](\cdot) \\ [y](\cdot) \end{pmatrix} \mapsto \begin{pmatrix} [a](\cdot) \cap ([x](\cdot) + [y](\cdot)) \\ [x](\cdot) \cap ([a](\cdot) - [y](\cdot)) \\ [y](\cdot) \cap ([a](\cdot) - [x](\cdot)) \end{pmatrix}$$

Example:
The non-minimal contractor associated to the constraint $c(\cdot) = \int_0 x(\tau)d\tau$:

$$\begin{pmatrix} [x](\cdot) \\ [c](\cdot) \end{pmatrix} \mapsto \begin{pmatrix} [x](\cdot) \\ [c](\cdot) \cap \int_0 [x](\tau)\, d\tau \end{pmatrix}$$

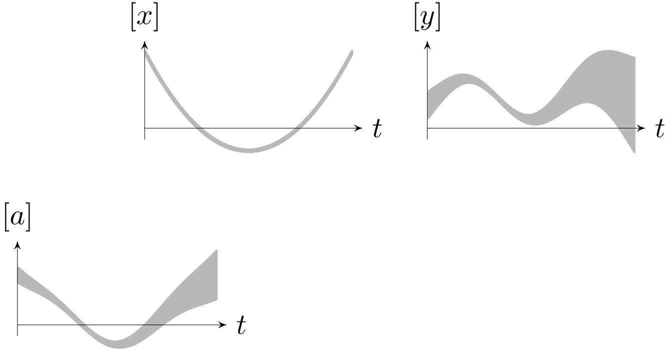
Tube Programming

Tubes programming: example



Tube Programming

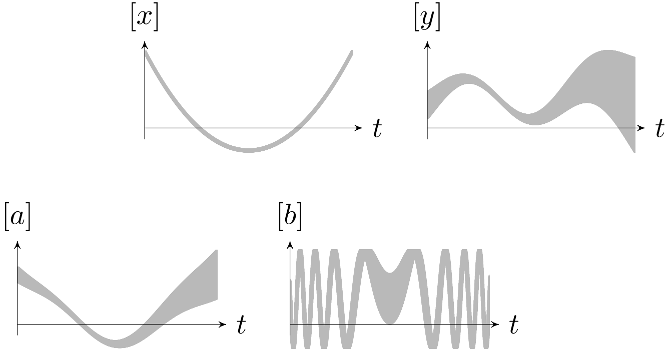
Tubes programming: example



$a(\cdot) = x(\cdot) + y(\cdot)$

Tube Programming

Tubes programming: example

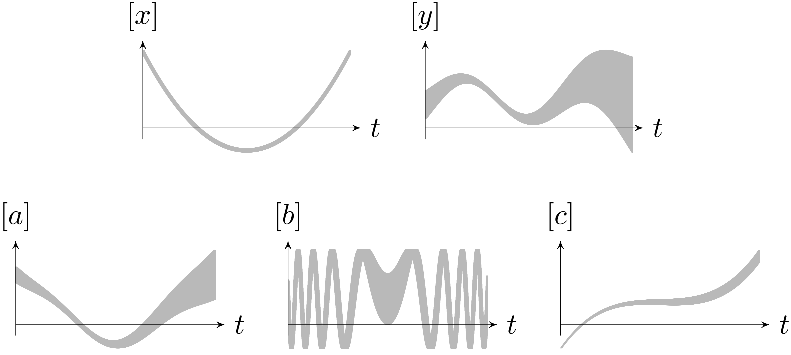


$$a(\cdot) = x(\cdot) + y(\cdot)$$

$$b(\cdot) = \sin\big(x(\cdot)\big)$$

Tube Programming

Tubes programming: example



$$a(\cdot) = x(\cdot) + y(\cdot) \qquad b(\cdot) = \sin\big(x(\cdot)\big) \qquad c(\cdot) = \int_0 x(\tau)d\tau$$

Section 3

Interval Integration

Interval Integration

Initial value problem

Our approach consists in describing an interval integration as a **constraint network** applied on trajectories:

- one constraint is given by:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{n}(t)$$

- another one is given by the initial condition, assumed known:

$$\mathbf{x}(0) = \mathbf{x}_0$$

Uncertainties are handled within intervals: $\mathbf{x}_0 \in [\mathbf{x}_0]$, $\mathbf{n}(t) \in [\mathbf{n}](t)$.
Contractors will remove unfeasible trajectories.

Interval Integration

Example: propagation method

Let us consider the following **initial value problem**:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

This problem is unstable.

Interval Integration

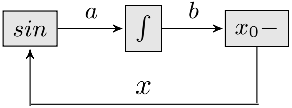
Example: propagation method

Let us consider the following **initial value problem**:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

This problem is unstable.
Decomposition into **primitive constraints**:

$$\begin{cases} a(\cdot) & = & \sin\big(x(\cdot)\big) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



All trajectories $x(\cdot)$, $a(\cdot)$, $b(\cdot)$ belong to tubes.
Presence of one loop $\longrightarrow$ iterative resolution until a **fix-point**.

Interval Integration

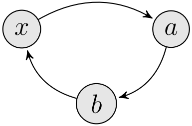
Forward propagation method

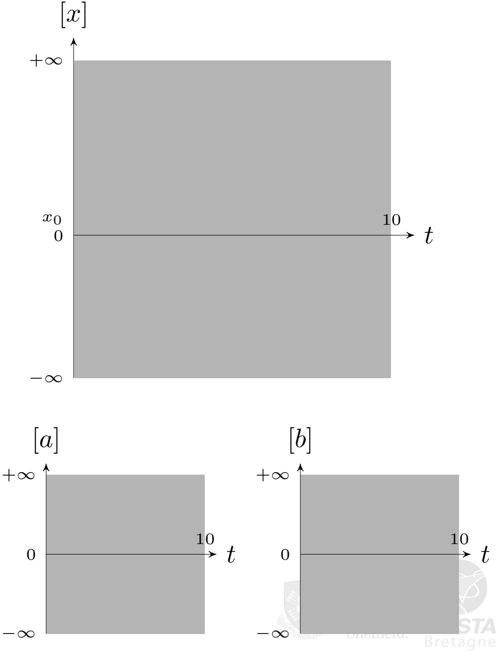
Problem:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$





Initial step

Interval Integration

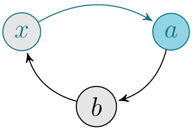
Forward propagation method

Problem:

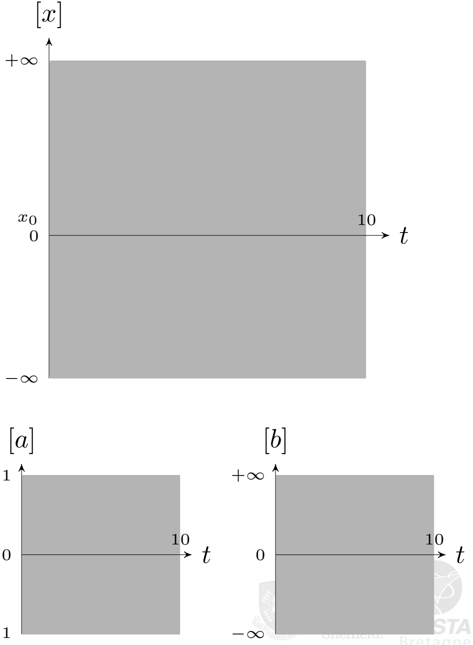
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #1



Interval Integration

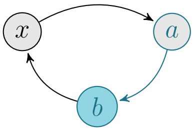
Forward propagation method

Problem:

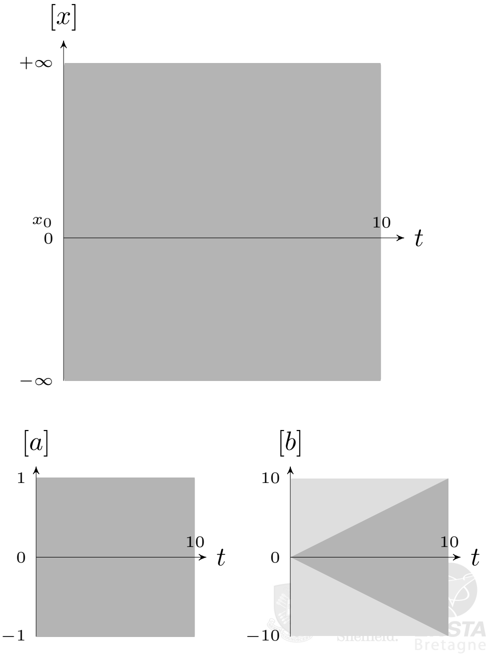
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #2



Interval Integration

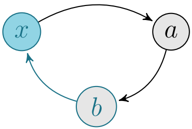
Forward propagation method

Problem:

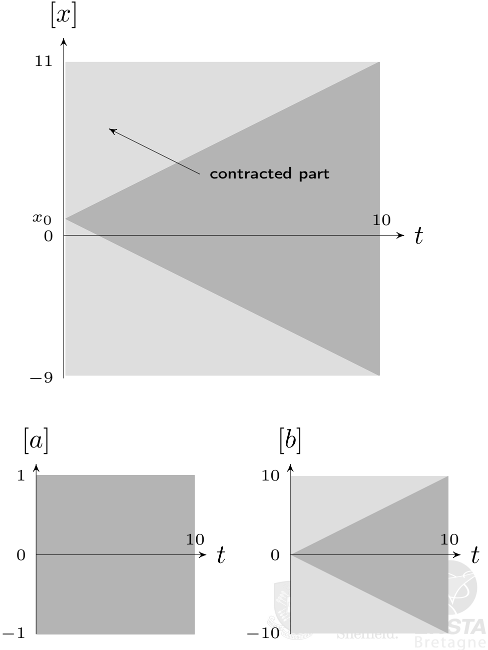
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #3



Interval Integration

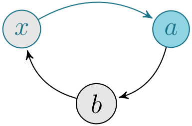
Forward propagation method

Problem:

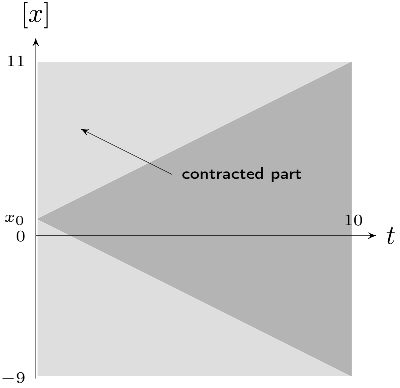
$$\left\{\begin{array}{rcl} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{array}\right.$$

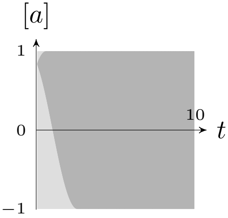
Decomposition:

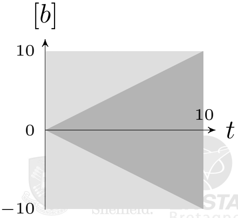
$$\left\{\begin{array}{rcl} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{array}\right.$$



Step #4







Interval Integration

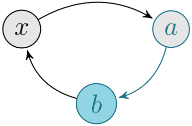
Forward propagation method

Problem:

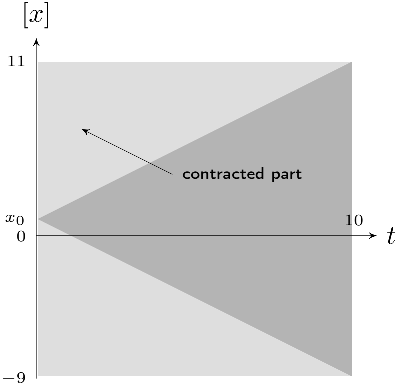
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

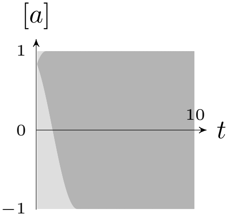
Decomposition:

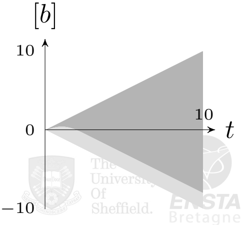
$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #5







Interval Integration

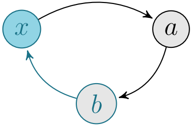
Forward propagation method

Problem:

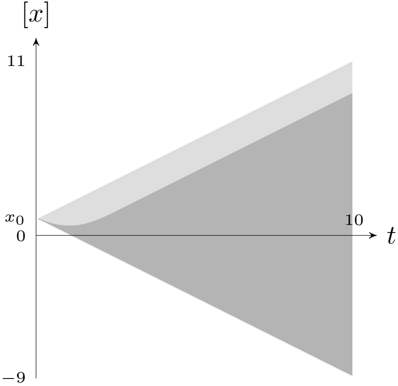
$$\begin{cases} \dot{x} &= -\sin(x) \\ x_0 &= 1 \end{cases}$$

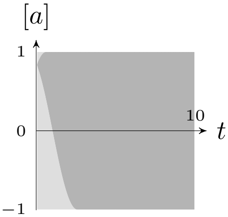
Decomposition:

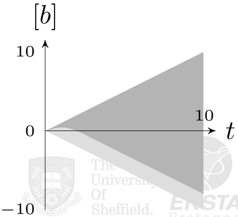
$$\begin{cases} a(\cdot) &= \sin\left(x(\cdot)\right) \\ b(\cdot) &= \int_0 a(\tau)d\tau \\ x(\cdot) &= x_0 - b(\cdot) \end{cases}$$



Step #6







Interval Integration

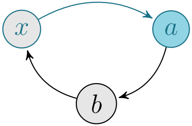
Forward propagation method

Problem:

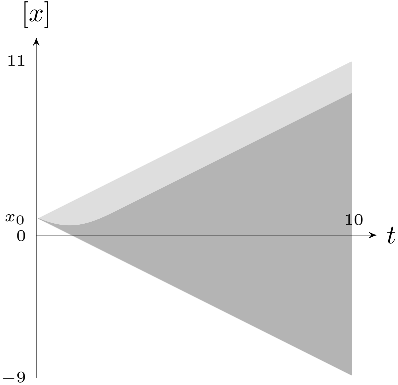
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

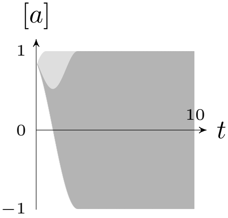
Decomposition:

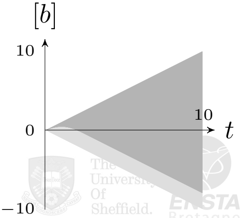
$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #7







Interval Integration

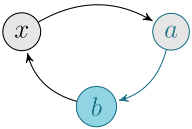
Forward propagation method

Problem:

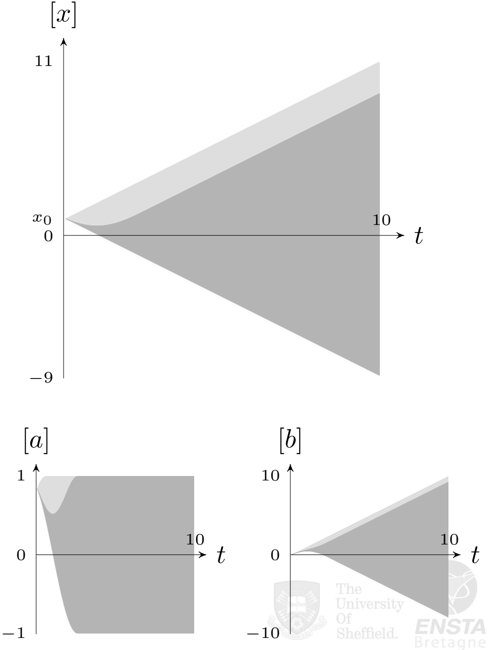
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #8



Interval Integration

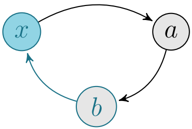
Forward propagation method

Problem:

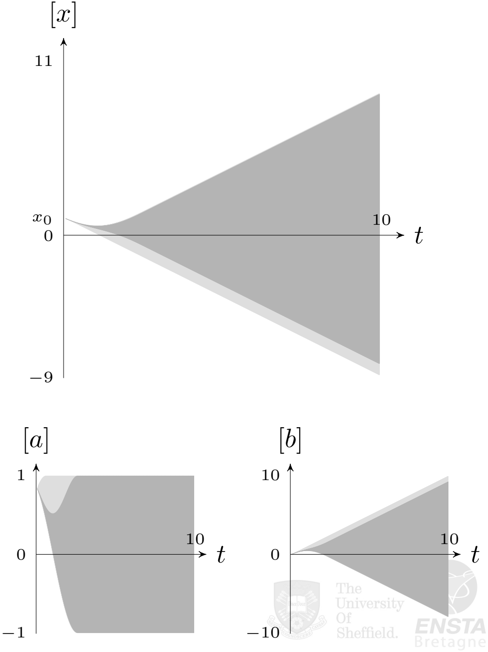
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #9



Interval Integration

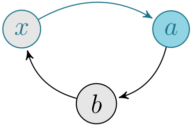
Forward propagation method

Problem:

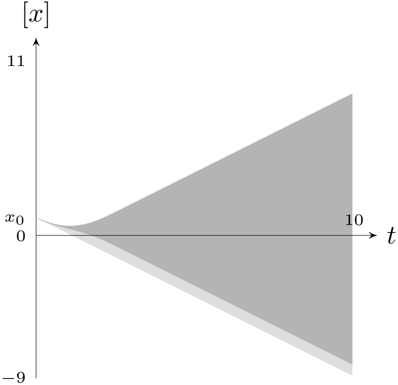
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

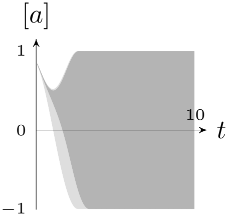
Decomposition:

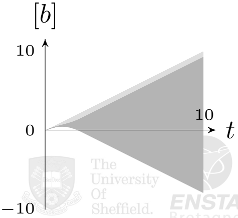
$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #10







Interval Integration

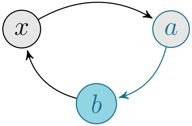
Forward propagation method

Problem:

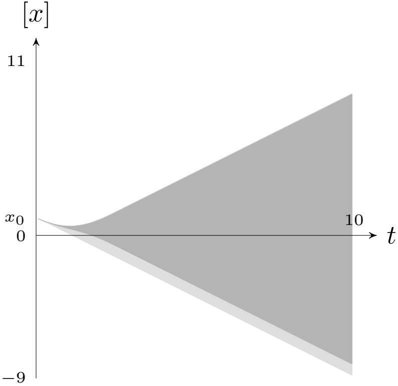
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

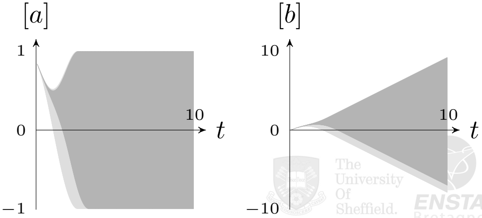
Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #11





Interval Integration

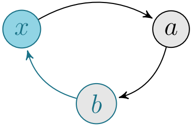
Forward propagation method

Problem:

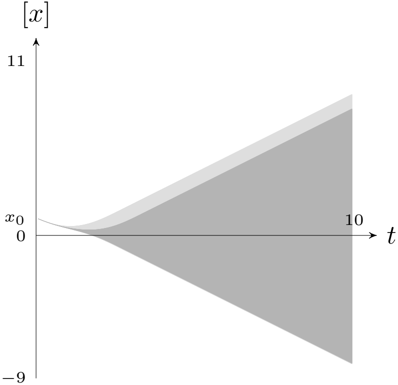
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

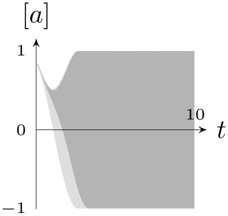
Decomposition:

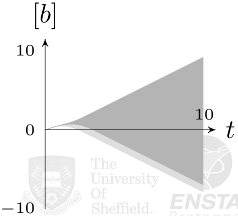
$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #12







Interval Integration

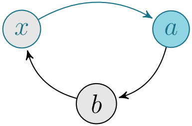
Forward propagation method

Problem:

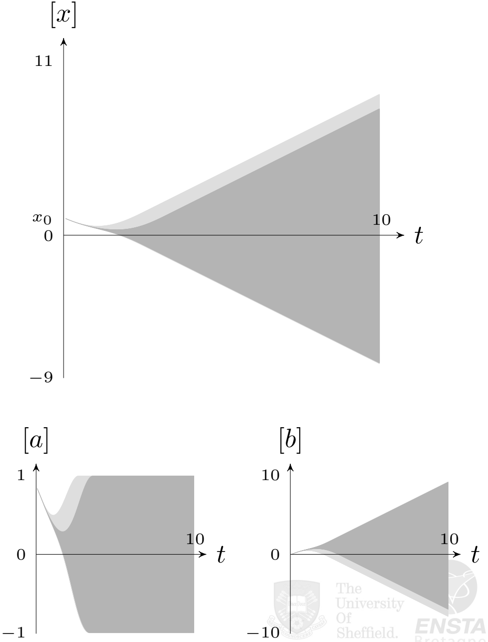
$$\begin{cases} \dot{x} &= -\sin(x) \\ x_0 &= 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) &= \sin\left(x(\cdot)\right) \\ b(\cdot) &= \int_0 a(\tau)d\tau \\ x(\cdot) &= x_0 - b(\cdot) \end{cases}$$



Step #13



Interval Integration

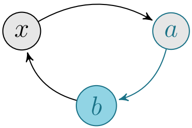
Forward propagation method

Problem:

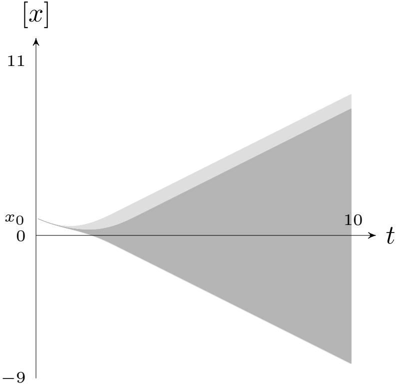
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

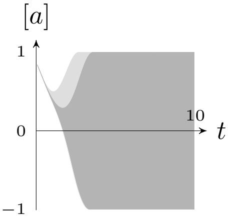
Decomposition:

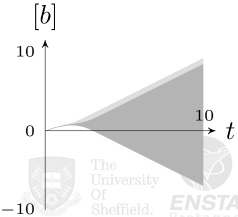
$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #14







Interval Integration

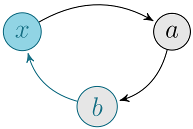
Forward propagation method

Problem:

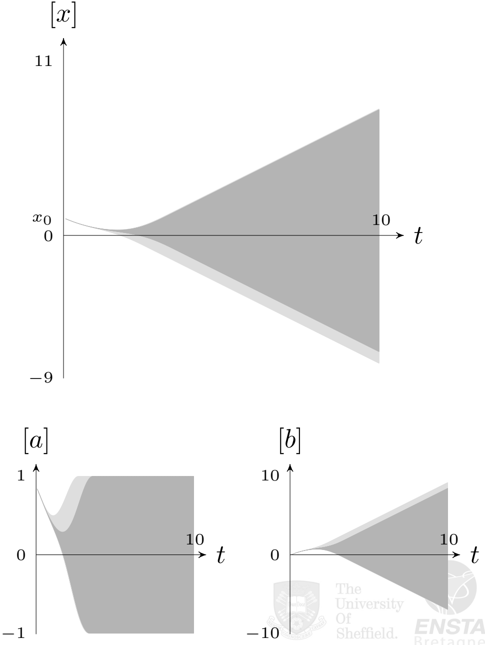
$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \end{cases}$$



Step #15



Interval Integration

Forward propagation method

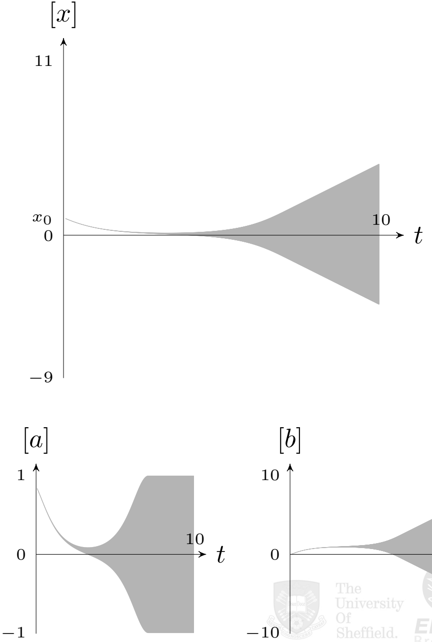
Problem:

$$\begin{cases} \dot{x} &= -\sin(x) \\ x_0 &= 1 \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) &= \sin\left(x(\cdot)\right) \\ b(\cdot) &= \int_0 a(\tau)d\tau \\ x(\cdot) &= x_0 - b(\cdot) \end{cases}$$

Final step (#42)
A fixpoint has been reached,
no more contractions



The University Of Sheffield.
ENSTA Bretagne

Interval Integration

Example: comparing with CAPD

Comparing with the CAPD library: [CAPD]

- the obtained **envelop is poor** compared with the results produced with the CAPD library, giving an accurate interval:

$$\begin{aligned} x(10) \in [x_f] &= [4.96\ldots \times 10^{-5}, 4.96\ldots \times 10^{-5}] \\ width([x_f]) &= 1.72 \times 10^{-15} \end{aligned}$$

- CAPD is **typically devoted to** this type of problem where the information given on the initial state has to propagate forward

Section 4

Simulation of a Mobile Robot

Simulation of a Mobile Robot

Dubin's car

For mobile robots, the differential equation often has a structure of a **causal kinematic chain** $\longrightarrow$ no iterative method needed.

Simulation of a Mobile Robot

Dubin's car

For mobile robots, the differential equation often has a structure of a **causal kinematic chain** $\longrightarrow$ no iterative method needed.

Considering a Dubin's car $\mathcal{R}$ which state is $\mathbf{x} = (x, y, \theta)^\intercal$.

$$\mathcal{R}\left\{\begin{array}{lcl} \dot{x}(t) & = & v \cdot \cos(\theta) \\ \dot{y}(t) & = & v \cdot \sin(\theta) \\ \dot{\theta}(t) & = & u(t) \end{array}\right.$$

Simulation of a Mobile Robot

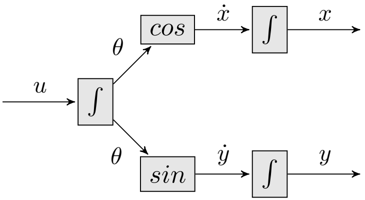
Dubin's car

For mobile robots, the differential equation often has a structure of a **causal kinematic chain** $\longrightarrow$ no iterative method needed.

Considering a Dubin's car $\mathcal{R}$ which state is $\mathbf{x} = (x, y, \theta)^\intercal$.

$$\mathcal{R}\left\{\begin{array}{rcl} \dot{x}(t) & = & v \cdot \cos(\theta) \\ \dot{y}(t) & = & v \cdot \sin(\theta) \\ \dot{\theta}(t) & = & u(t) \end{array}\right.$$

We obtain the following causal chain:



Simulation of a Mobile Robot

Dubin's car in forward

In order to compare with the CAPD library, let us specify:

- car's speed: $v = 10m.s^{-1}$
- $\mathbf{x}_0 \in [-1, 1] \times [-1, 1] \times [\frac{-6\pi}{5} - 0.002, \frac{-6\pi}{5} + 0.002]$
- $u(t) \in -\cos\left(\frac{t+33}{5}\right) + [-0.02, 0.02]$

Simulation of a Mobile Robot

Dubin's car in forward

In order to compare with the CAPD library, let us specify:

- car's speed: $v = 10m.s^{-1}$
- $\mathbf{x}_0 \in [-1,1] \times [-1,1] \times [\frac{-6\pi}{5} - 0.002, \frac{-6\pi}{5} + 0.002]$
- $u(t) \in -\cos\left(\frac{t+33}{5}\right) + [-0.02, 0.02]$

Initial tube $[u](\cdot)$ is obtained with the analytical expression.
Other tubes $[\theta](\cdot)$, $[\dot{x}](\cdot)$, $[x](\cdot)$, $\ldots$, are initialized to $[-\infty, +\infty]$.

Simulation of a Mobile Robot

Dubin's car in forward

In order to compare with the CAPD library, let us specify:

- car's speed: $v = 10m.s^{-1}$
- $\mathbf{x}_0 \in [-1,1] \times [-1,1] \times [\frac{-6\pi}{5} - 0.002, \frac{-6\pi}{5} + 0.002]$
- $u(t) \in -\cos\left(\dfrac{t+33}{5}\right) + [-0.02, 0.02]$

Initial tube $[u](\cdot)$ is obtained with the analytical expression.
Other tubes $[\theta](\cdot)$, $[\dot{x}](\cdot)$, $[x](\cdot)$, $\dots$, are initialized to $[-\infty, +\infty]$.

N.B.: with tubes, both analytical expression and **real data** can be considered, which is useful for robotic purposes.

Simulation of a Mobile Robot

Dubin's car in forward

The following figure shows that our method is more accurate than CAPD on this example:

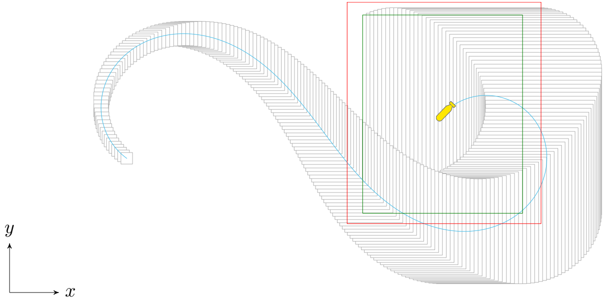


Figure: Interval simulation of the robot following Dubin's car equations. Blue line: true poses of the robot. Gray boxes: tubes $[x](\cdot) \times [y](\cdot)$ projected on the world frame. Green box: final box obtained for $t = 14$. Red box: result computed with CAPD.

Simulation of a Mobile Robot

Dubin's car in forward/backward

Same simulation considering constraints on both initial and final states:

$$\begin{array}{rcl} \mathbf{x}(0) & \in & [-1,1] \times [-1,1] \\ \mathbf{x}(14) & \in & [53.9,55.9] \times [6.9,8.9] \times [-2.36,-2.32] \end{array}$$

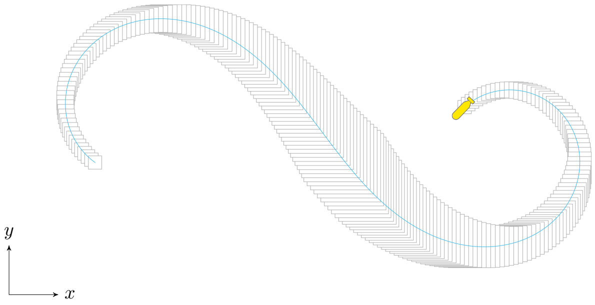


Figure: With a forward/backward contraction, uncertainties are maximal in the middle of the mission. No possible comparison with CAPD.

Simulation of a Mobile Robot

Dubin's car with measurements: state estimation

Same simulation adding **measurements from beacons**.

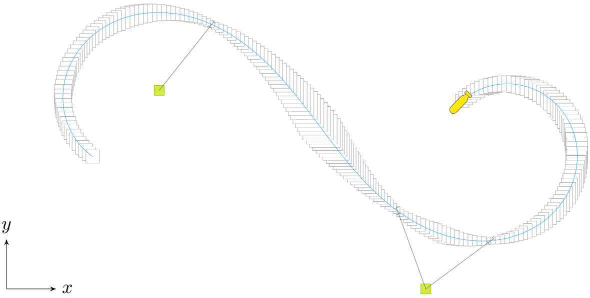


Figure: With a forward/backward contraction, uncertainties are maximal in the middle of the mission. No possible comparison with CAPD.

Simulation of a Mobile Robot

Dubin's car with measurements: state estimation

Same simulation adding **measurements from beacons**, without any knowledge on initial and final conditions:

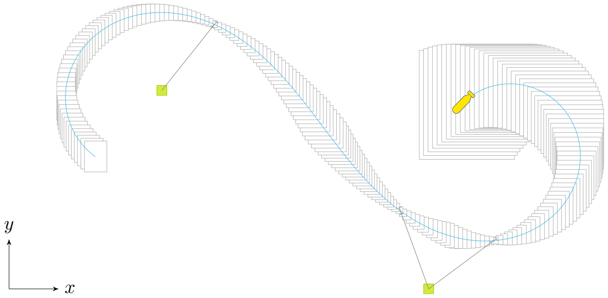


Figure: With a forward/backward contraction, uncertainties are maximal in the middle of the mission. No possible comparison with CAPD.

Conclusion

Tube programming provides a simple and efficient constraint-based method for the guaranteed interval computation of mobile robots trajectories.

Conclusion

Tube programming provides a simple and efficient constraint-based method for the guaranteed interval computation of mobile robots trajectories.

Advantages:

- approach more general than other existing methods
- simple problem description by defining constraints
- competitive method for robotics applications
- easy to deal with datasets

Conclusion

Tube programming provides a simple and efficient constraint-based method for the guaranteed interval computation of mobile robots trajectories.

Advantages:

- approach more general than other existing methods
- simple problem description by defining constraints
- competitive method for robotics applications
- easy to deal with datasets

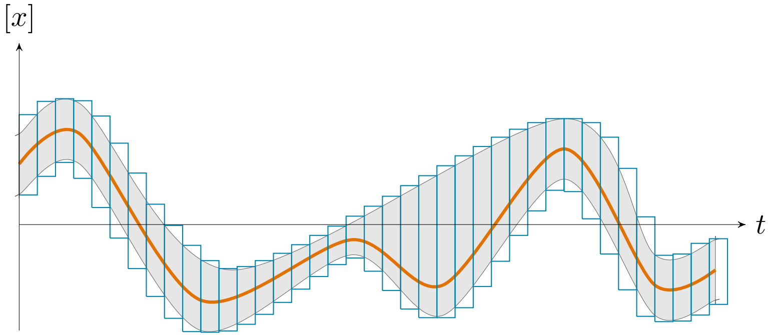
Drawbacks:

- pessimism added by implementation
- computation time?

Conclusion

An open-source `C++/python` tube library will be soon available.

`simon.rohou@gmail.com`



Support:

DGA
Direction Générale de l'Armement

Tools:

IBEX library
used for interval arithmetic, contractor programming

VIBES
used for rendering

References:

[Mackworth1977] A. K. Mackworth,
Consistency in networks of relations,
Artificial Intelligence, 8(1):99–118, 1977.

[Chabert2009] G. Chabert, L. Jaulin,
Contractor Programming,
Artificial Intelligence, 173:1079–1100, 2009.

[LeBars2012] F. Le Bars, J. Sliwka, O. Reynet, L. Jaulin,
State estimation with fleeting data,
Automatica, 48(2):381–387, 2012.

[Aubry2013] C. Aubry, R. Desmare, L. Jaulin,
Loop detection of mobile robots using interval analysis,
Automatica, 2013.

[CAPD] Capd, Computer Assisted Proofs in Dynamics group, a C++ package for rigorous numerics,
`http://capd.ii.uj.edu.pl`

Section 7

Appendix

Appendix

Application on real data

Figure: DAURADE AUV managed by DGA Techniques Navales Brest and the Service hydrographique et océanographique de la Marine, during an experiment in the Rade de Brest, October 2015.

Appendix

Application on real data

Classical kinematic model of an underwater robot:

$$\begin{cases} \dot{\mathbf{p}} & = & \mathbf{R}(\psi,\theta,\varphi) \cdot \mathbf{v}_r \\ \dot{\mathbf{v}}_r & = & \mathbf{a}_r - \boldsymbol{\omega}_r \wedge \mathbf{v}_r \end{cases}$$

Where:

- $\mathcal{R}_0$ is the absolute inertial coordinate system
- $\mathcal{R}_1$ is robot's own coordinate system
- (ψ,θ,φ) and $\mathbf{R}(\psi,\theta,\varphi)$ are Euler angles and matrix
- $\mathbf{p} = (p_x, p_y, p_z)$ gives the center of the robot in $\mathcal{R}_0$
- $\mathbf{v}_r$ is the speed vector in $\mathcal{R}_1$
- $\mathbf{a}_r$ is the acceleration vector in $\mathcal{R}_1$
- $\boldsymbol{\omega}_r = (\omega_x, \omega_y, \omega_z)$ is the rotation vector of the robot relative to $\mathcal{R}_0$ expressed in $\mathcal{R}_1$

Appendix

Application on real data

Classical kinematic model of an underwater robot:

$$\begin{cases} \dot{\mathbf{p}} & = & \mathbf{R}(\psi,\theta,\varphi) \;\cdot\; \mathbf{v}_r \\ \dot{\mathbf{v}}_r & = & \mathbf{a}_r - \boldsymbol{\omega}_r \wedge \mathbf{v}_r \end{cases}$$

Sensed values:
Speed vector $\mathbf{v}_r$, acceleration vector $\mathbf{a}_r$, Euler angles (ψ,θ,φ)

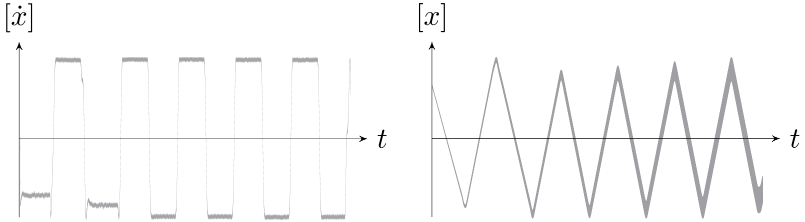
Corresponding tubes:

- feeded with sensor data:
  $[\mathbf{v}_r](\cdot)$, $[\mathbf{a}_r](\cdot)$, $[\psi](\cdot)$, $[\theta](\cdot)$, $[\varphi](\cdot)$
- to be computed:
  $[\mathbf{p}](\cdot)$

Appendix

Application on real data

Presenting Daurade's tubes $[\dot{x}](\cdot)$ (velocity along x in $\mathcal{R}_0$) and $[x](\cdot)$ (position along x in $\mathcal{R}_0$):



- $[\dot{x}](\cdot)$ – tube's thickness coming from measurements remains constant
- $[x](\cdot)$ – tube becomes thicker depicting cumulative errors due to the dead-reckoning method and $[\dot{x}](\cdot)$ non-zero thickness

Appendix

Application on real data

Mission map after one hour:

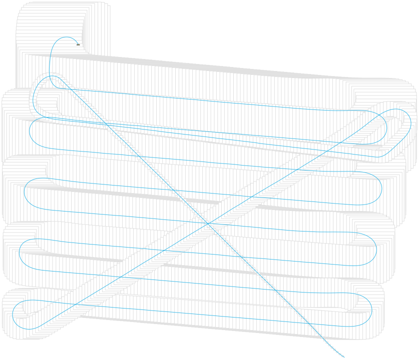


Figure: Blue line: Daurade's true trajectory, given by an ultra-short baseline (USBL), a system made of underwater acoustic sensors for positioning purposes. Gray boxes: tubes $[x](\cdot) \times [y](\cdot)$ projected on the world frame. Thanks to the DVL sensor, the drift is limited.

Appendix

Strangulation method

Some solutions $\mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)$ can be removed with a **strangulation (bottleneck) method**.

By testing a **fictive fleeting constraint**, for instance:

$$\mathbf{x}(t_a) < \mathbf{y}_a \ , \ \mathbf{y}_a \in [\mathbf{x}](t_a)$$

the aforementioned constraints can then apply again.

Appendix

Strangulation method

Some solutions $\mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)$ can be removed with a **strangulation (bottleneck) method**.

By testing a **fictive fleeting constraint**, for instance:

$$\mathbf{x}(t_a) < \mathbf{y}_a \ , \ \mathbf{y}_a \in [\mathbf{x}](t_a)$$

the aforementioned constraints can then apply again.

- if all solutions are removed: $[\mathbf{x}](\cdot) = \varnothing$, then the opposite constraint $\mathbf{x}(t_a) \geqslant \mathbf{y}_a$ is added to the network

Appendix

Strangulation method

Some solutions $\mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)$ can be removed with a **strangulation (bottleneck) method**.

By testing a **fictive fleeting constraint**, for instance:

$$\mathbf{x}(t_a) < \mathbf{y}_a \ , \ \mathbf{y}_a \in [\mathbf{x}](t_a)$$

the aforementioned constraints can then apply again.

- if all solutions are removed: $[\mathbf{x}](\cdot) = \varnothing$, then the opposite constraint $\mathbf{x}(t_a) \geqslant \mathbf{y}_a$ is added to the network
- otherwise another strangulation can be tested

Appendix

Strangulation method

Coming back to the example:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

Appendix

Strangulation method

Coming back to the example:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

- The following constraint is tested: $x(10) > \epsilon$
  Its propagation returns an empty set of trajectories $[x](\cdot) = \varnothing$.
  The constraint $x(10) \leqslant \epsilon$ is added to the network.

Appendix

Strangulation method

Coming back to the example:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \end{cases}$$

- The following constraint is tested: $x(10) > \epsilon$
  Its propagation returns an empty set of trajectories $[x](\cdot) = \varnothing$.
  The constraint $x(10) \leqslant \epsilon$ is added to the network.

- The following constraint is tested: $x(10) < -\epsilon$
  Its propagation returns an empty set of trajectories $[x](\cdot) = \varnothing$.
  The constraint $-\epsilon \leqslant x(10) \leqslant \epsilon$ is finally kept.

Appendix

Strangulation method

Coming back to the example:

$$\begin{cases} \dot{x} &= -\sin(x) \\ x_0 &= 1 \end{cases}$$

- The following constraint is tested: $x(10) > \epsilon$
  Its propagation returns an empty set of trajectories $[x](\cdot) = \varnothing$.
  The constraint $x(10) \leqslant \epsilon$ is added to the network.
- The following constraint is tested: $x(10) < -\epsilon$
  Its propagation returns an empty set of trajectories $[x](\cdot) = \varnothing$.
  The constraint $-\epsilon \leqslant x(10) \leqslant \epsilon$ is finally kept.

This process can be automated with a **bisection algorithm**.

Appendix

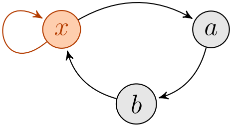
Strangulation method

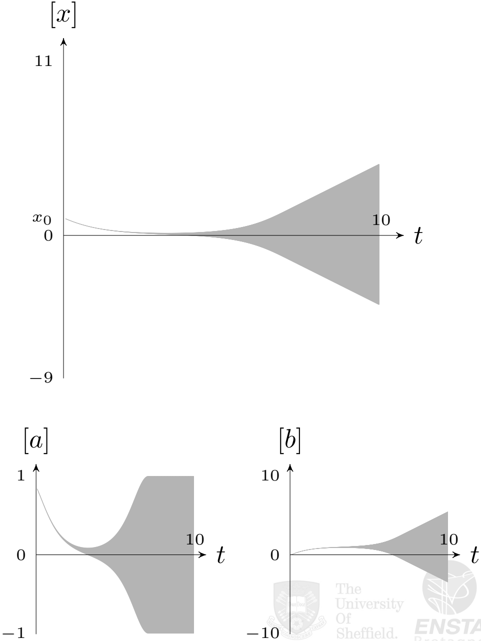
Problem:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \\ x_f & \in & [-\epsilon, \epsilon] \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau)d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \\ x(10) & = & x_f \end{cases}$$





Appendix

Strangulation method

Problem:

$$\begin{cases} \dot{x} & = & -\sin(x) \\ x_0 & = & 1 \\ x_f & \in & [-\epsilon, \epsilon] \end{cases}$$

Decomposition:

$$\begin{cases} a(\cdot) & = & \sin\left(x(\cdot)\right) \\ b(\cdot) & = & \int_0 a(\tau) d\tau \\ x(\cdot) & = & x_0 - b(\cdot) \\ x(10) & = & x_f \end{cases}$$