The Tubex Library

Simon Rohou

April 23rd, 2019

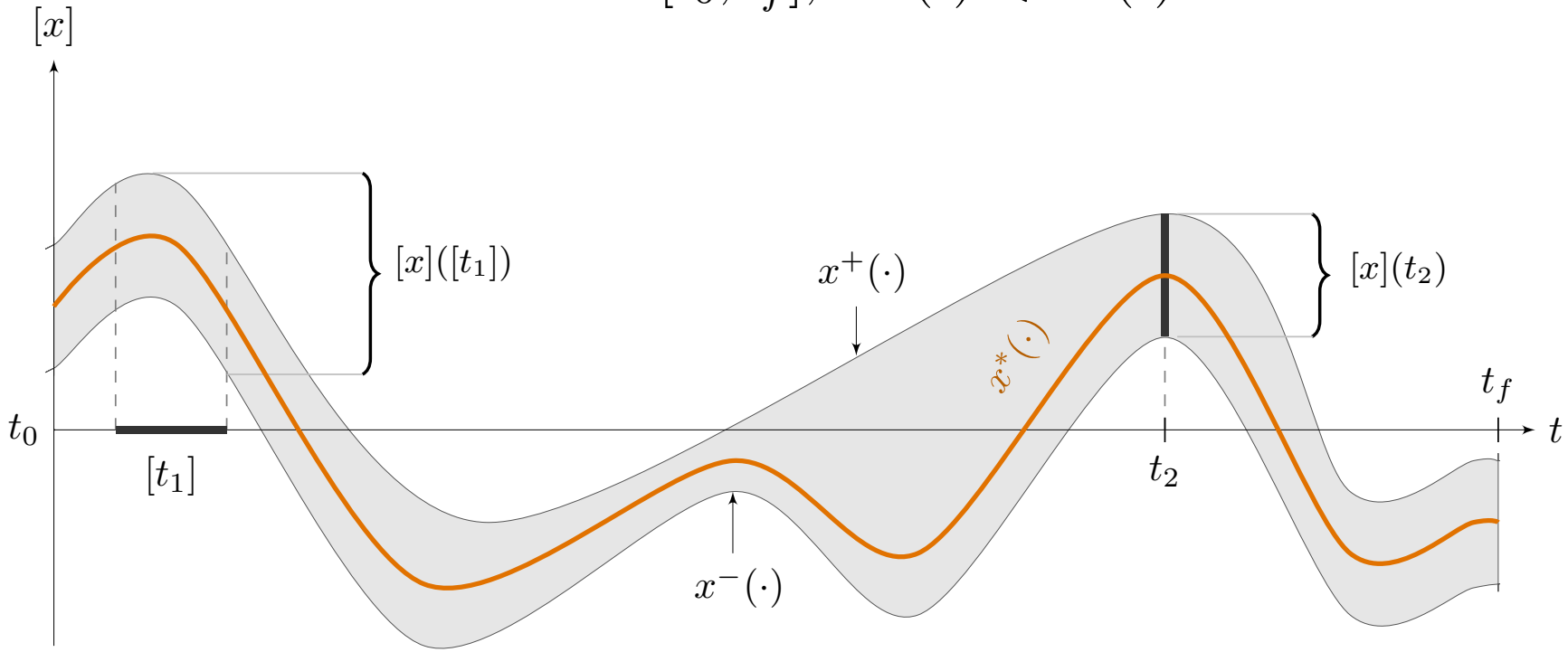
Section 1

Tubes

Tubes

Definition

Tube $[x](\cdot)$: interval of trajectories $[x^-(\cdot), x^+(\cdot)]$
such that $\forall t \in [t_0, t_f],\ x^-(t) \leqslant x^+(t)$

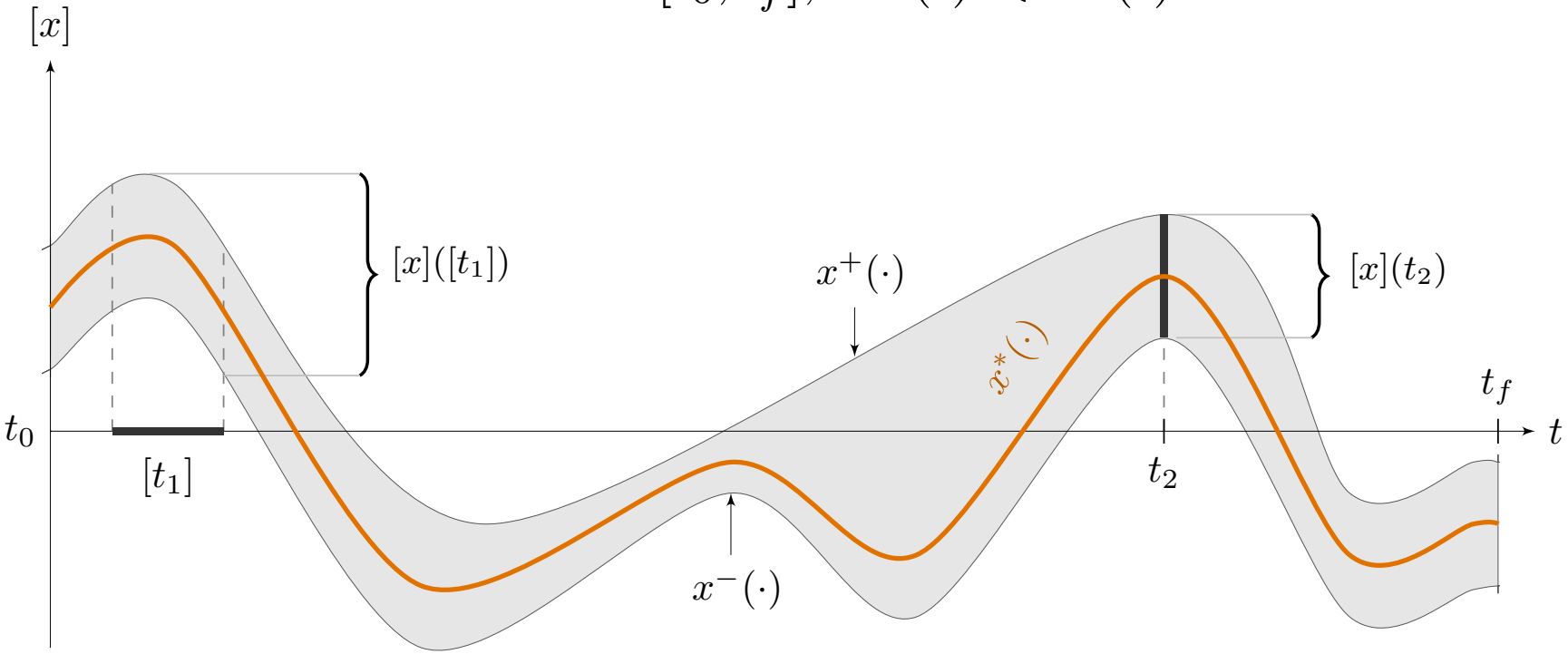


Tube $[x](\cdot)$ enclosing an uncertain trajectory $x^*(\cdot)$

Tubes

Definition

Tube $[x](\cdot)$: interval of trajectories $[x^-(\cdot), x^+(\cdot)]$
such that $\forall t \in [t_0, t_f],\ x^-(t) \leqslant x^+(t)$

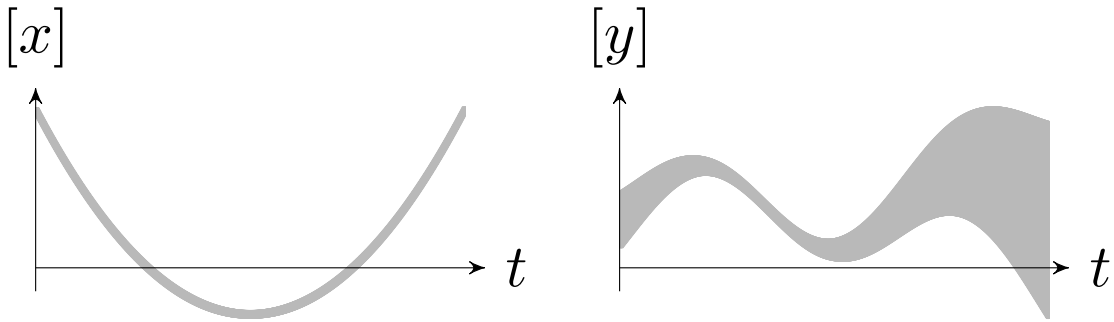


Tube $[x](\cdot)$ enclosing an uncertain trajectory $x^*(\cdot)$

Set-membership approach:
$x^*(\cdot) \in [x](\cdot)$, computations on bounds $\Rightarrow$ **guaranteed outputs**

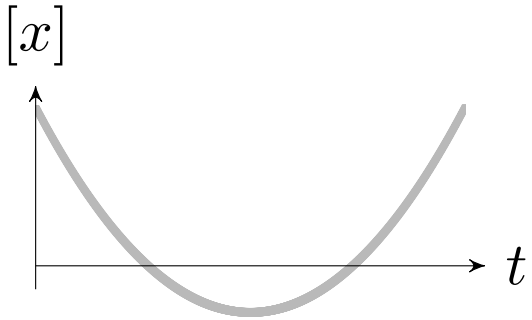
Tubes

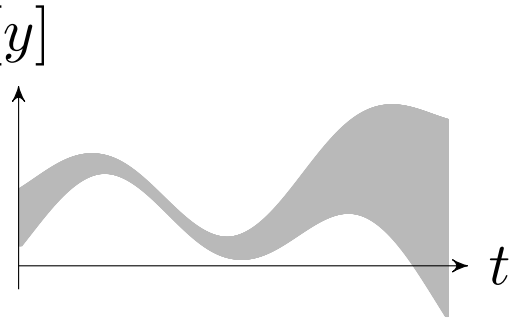
Tubes arithmetic

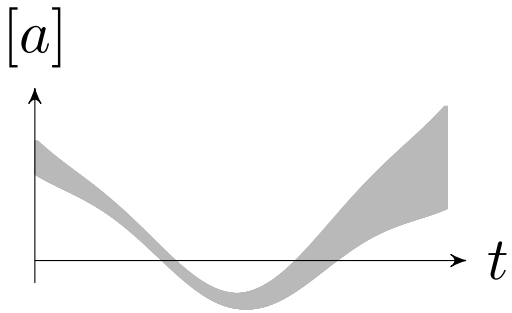


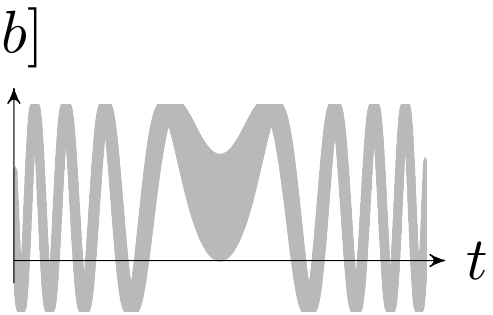
Tubes

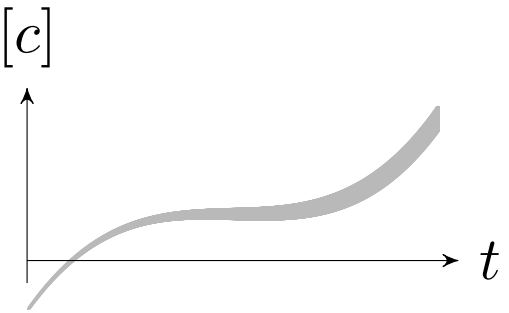
Tubes arithmetic











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

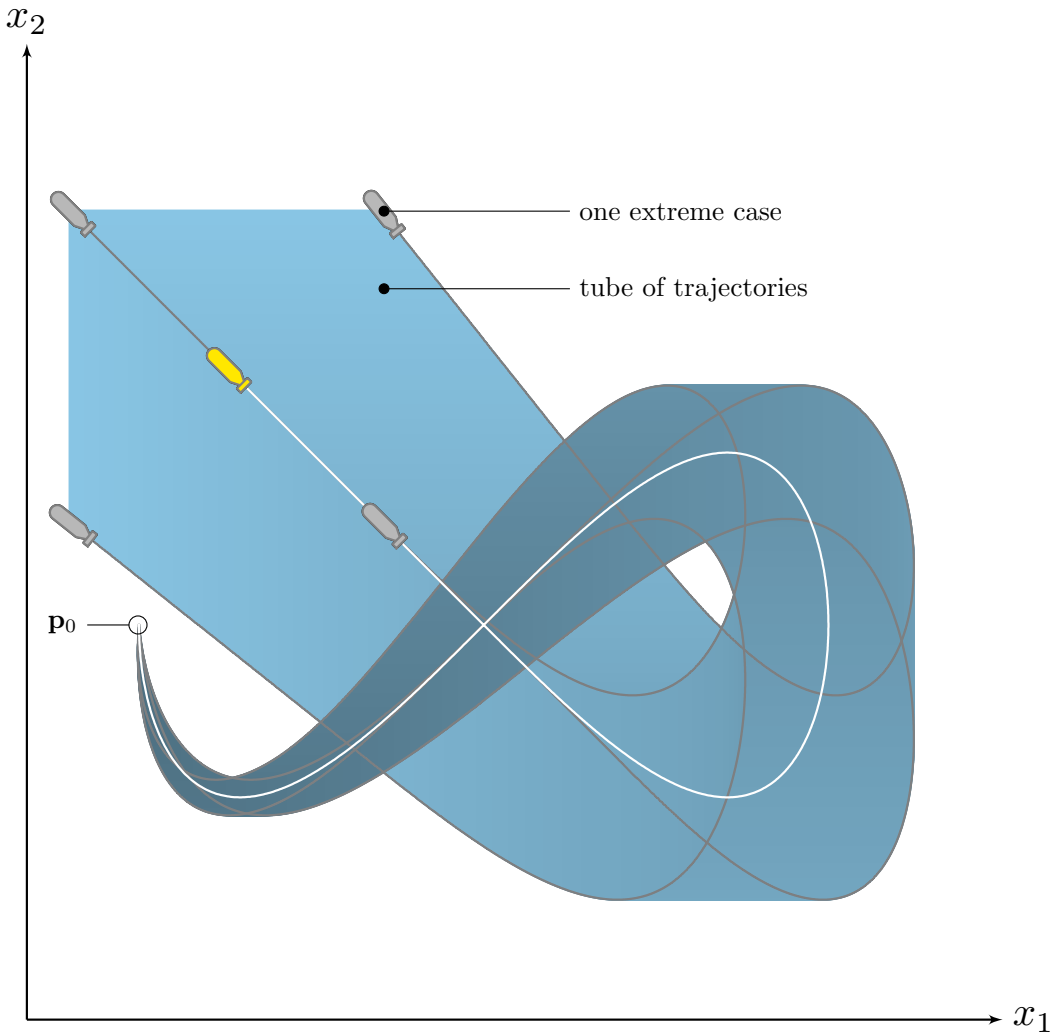
$$[b](\cdot) = \sin\big([x](\cdot)\big)$$

$$[c](\cdot) = \int_0 [x](\tau)d\tau$$

Tubes

Example of application: mobile robotics

Enclosure of an uncertain trajectory of a **mobile robot**:



Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available
- computations stand on IBEX (`Interval`, `Ctc`, ...)
  - same structure (contractors, functions, vector cases)
  - temporal objects (`Trajectory`, `tubex::Function`)

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available
- computations stand on IBEX (`Interval`, `Ctc`, ...)
  - same structure (contractors, functions, vector cases)
  - temporal objects (`Trajectory`, `tubex::Function`)
- **constraint propagation** with contractors:
  - non-linear differential equations
  - time uncertainties
  - delays
  - ...

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available
- computations stand on IBEX (`Interval`, `Ctc`, ...)
  - same structure (contractors, functions, vector cases)
  - temporal objects (`Trajectory`, `tubex::Function`)
- **constraint propagation** with contractors:
  - non-linear differential equations
  - time uncertainties
  - delays
  - ...
- several **robotics** applications and illustrations

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available
- computations stand on IBEX (`Interval`, `Ctc`, ...)
  - same structure (contractors, functions, vector cases)
  - temporal objects (`Trajectory`, `tubex::Function`)
- **constraint propagation** with contractors:
  - non-linear differential equations
  - time uncertainties
  - delays
  - ...
- several **robotics** applications and illustrations
- LGPL license

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available
- computations stand on IBEX (`Interval`, `Ctc`, ...)
  - same structure (contractors, functions, vector cases)
  - temporal objects (`Trajectory`, `tubex::Function`)
- **constraint propagation** with contractors:
  - non-linear differential equations
  - time uncertainties
  - delays
  - ...
- several **robotics** applications and illustrations
- LGPL license
- 3600+ unit tests

Tubes

Tubex in a nutshell

- extension of IBEX (static) for **dynamical systems**
- development since 2015
- **C++ library**, Python wrapping soon available
- computations stand on IBEX (`Interval`, `Ctc`, ...)
  - same structure (contractors, functions, vector cases)
  - temporal objects (`Trajectory`, `tubex::Function`)
- **constraint propagation** with contractors:
  - non-linear differential equations
  - time uncertainties
  - delays
  - ...
- several **robotics** applications and illustrations
- LGPL license
- 3600+ unit tests
- http://simon-rohou.fr/research/tubex-lib

Section 2

Tubes contractors

Tubes contractors

Definition

Contractors on boxes can be extended to sets of trajectories.

Definition

A contractor $\mathcal{C}_{\mathcal{L}}$ applied on a tube $[x](\cdot)$ aims at removing infeasible trajectories according to a given constraint $\mathcal{L}$ so that:

Tubes contractors

Definition

Contractors on boxes can be extended to sets of trajectories.

Definition

A contractor $\mathcal{C}_{\mathcal{L}}$ applied on a tube $[x](\cdot)$ aims at removing infeasible trajectories according to a given constraint $\mathcal{L}$ so that:

$$(i) \quad \forall t \in [t_0, t_f],\ \mathcal{C}_{\mathcal{L}}\big([x](t)\big) \subseteq [x](t) \qquad \text{(contraction)}$$

Tubes contractors

Definition

Contractors on boxes can be extended to sets of trajectories.

Definition

A contractor $\mathcal{C}_{\mathcal{L}}$ applied on a tube $[x](\cdot)$ aims at removing infeasible trajectories according to a given constraint $\mathcal{L}$ so that:

$$(i) \quad \forall t \in [t_0, t_f],\ \mathcal{C}_{\mathcal{L}}\big([x](t)\big) \subseteq [x](t) \qquad \text{(contraction)}$$

$$(ii) \quad \begin{pmatrix} \mathcal{L}\big(x(\cdot)\big) \\ x(\cdot) \in [x](\cdot) \end{pmatrix} \implies x(\cdot) \in \mathcal{C}_{\mathcal{L}}\big([x](\cdot)\big) \qquad \text{(consistency)}$$

Tubes contractors

Arithmetic contractors

Example:

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

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

Tubes contractors

Arithmetic 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 $c(\cdot) = \int_0^{\cdot} x(\tau) d\tau$:

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

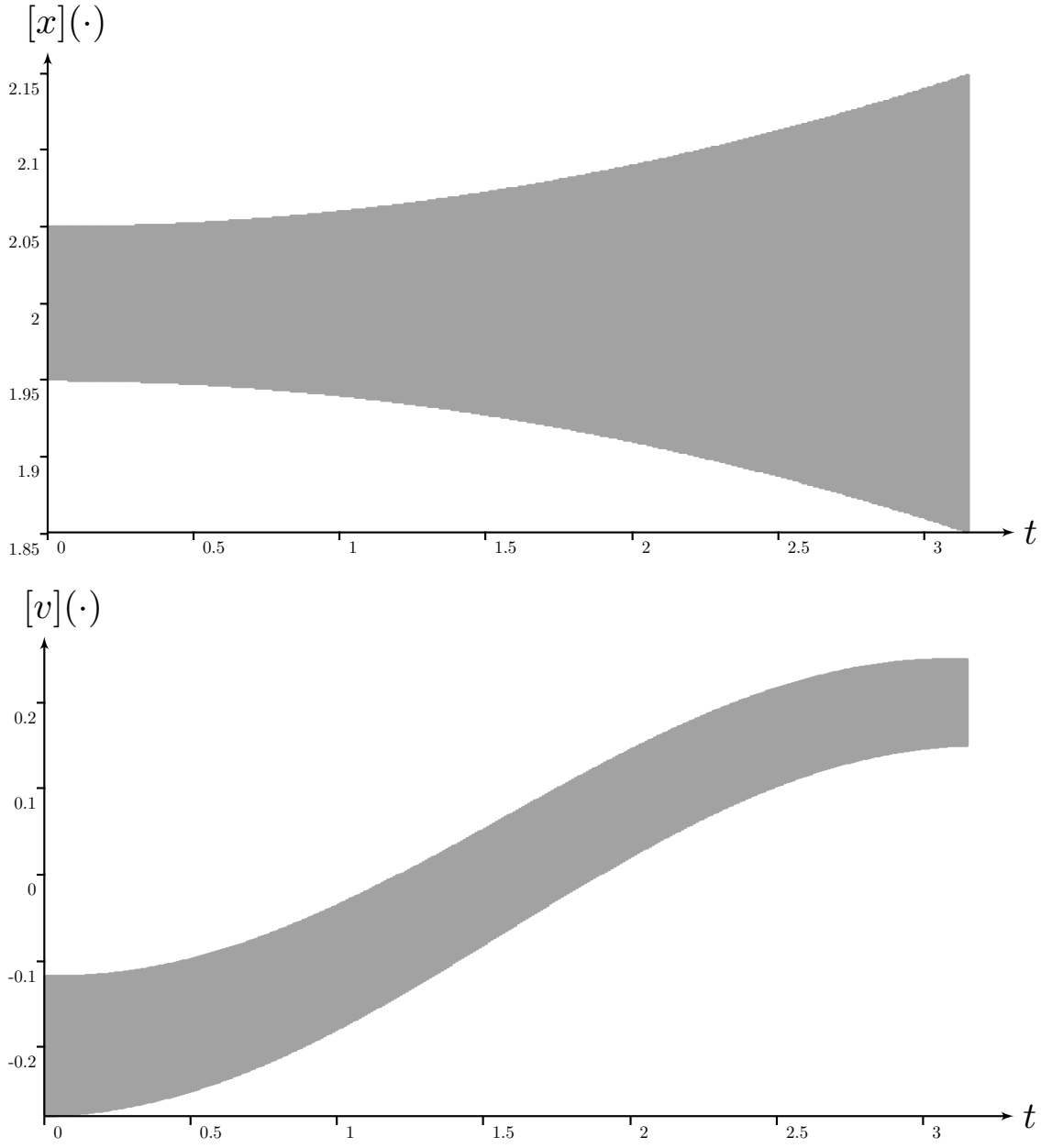
Tubes contractors

$$\mathcal{L}_{\frac{d}{dt}}\left(\mathbf{x}(\cdot), \mathbf{v}(\cdot)\right)$$

Differential constraint:

- $\dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot)$
- elementary constraint

Related contractor $\mathcal{C}_{\frac{d}{dt}}$:



Tubes contractors

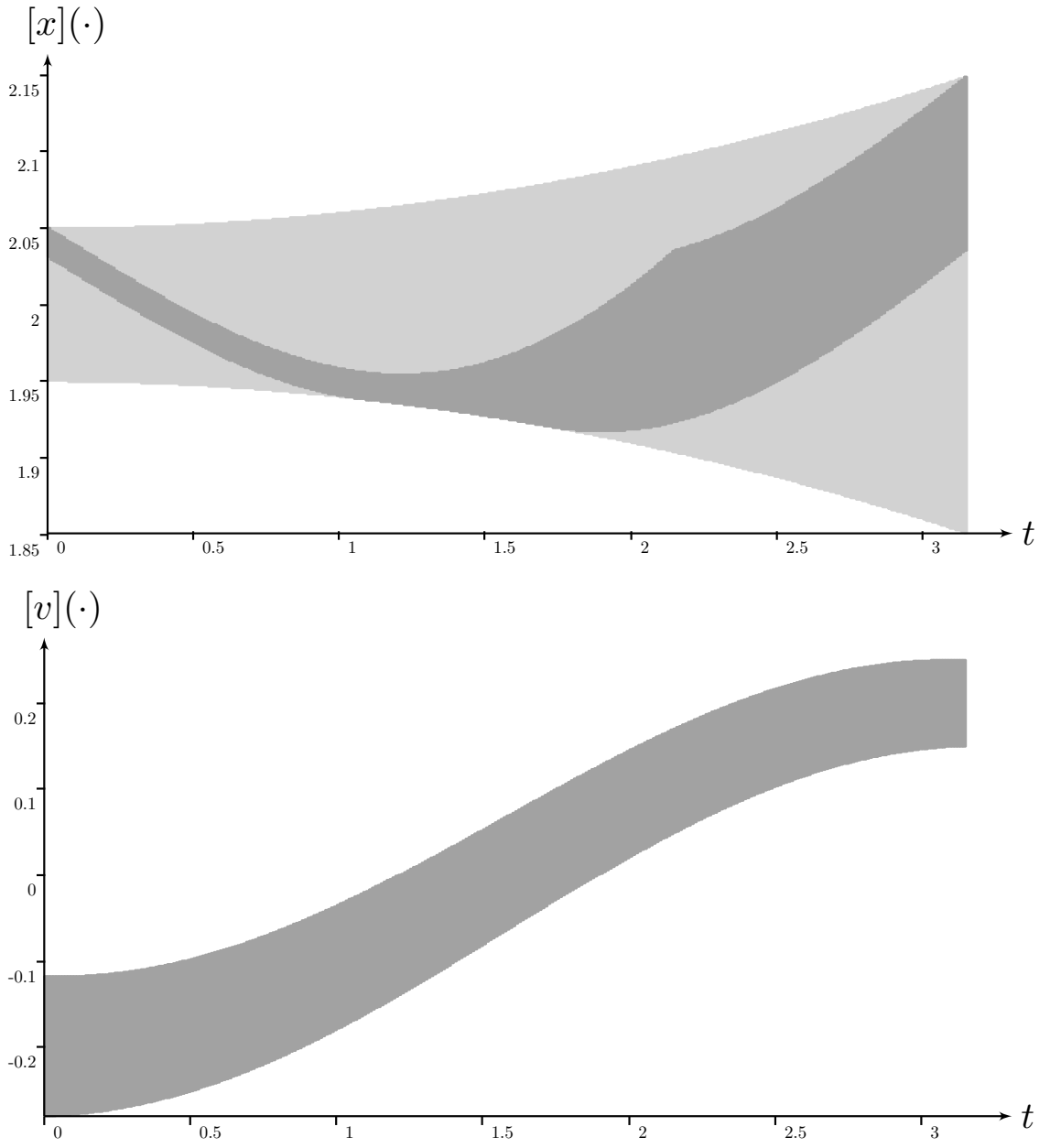
$$\mathcal{L}_{\frac{d}{dt}}\left(\mathbf{x}(\cdot),\mathbf{v}(\cdot)\right)$$

Differential constraint:

- $\dot{\mathbf{x}}(\cdot)=\mathbf{v}(\cdot)$
- elementary constraint

Related contractor $\mathcal{C}_{\frac{d}{dt}}$:

- $\mathcal{C}_{\frac{d}{dt}}\left([\mathbf{x}](\cdot),[\mathbf{v}](\cdot)\right)$



Tubes contractors

$$\mathcal{L}_{\frac{d}{dt}}\left(\mathbf{x}(\cdot),\mathbf{v}(\cdot)\right)$$

Differential constraint:

- $\dot{\mathbf{x}}(\cdot)=\mathbf{v}(\cdot)$
- elementary constraint

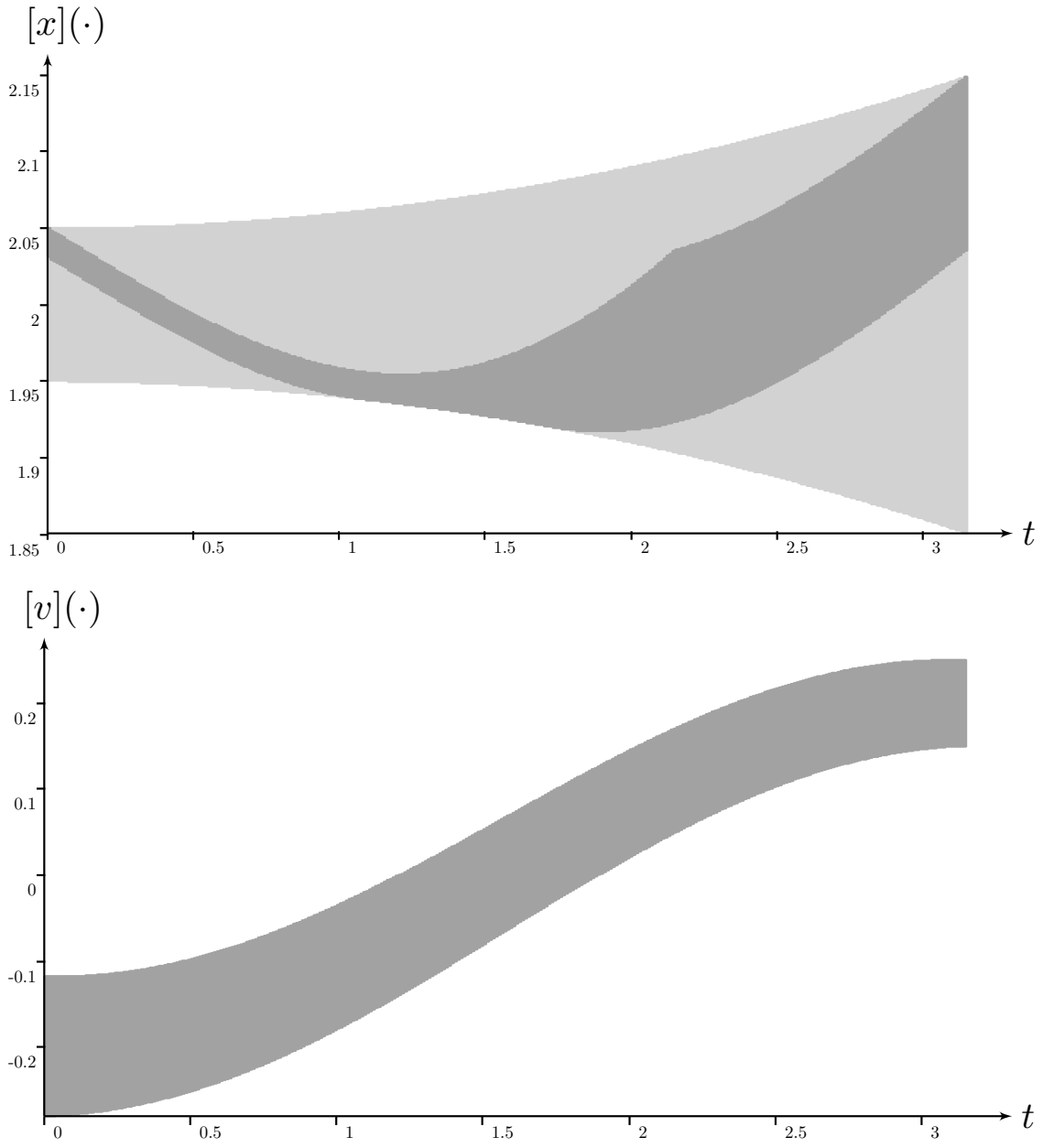
Related contractor $\mathcal{C}_{\frac{d}{dt}}$:

- $\mathcal{C}_{\frac{d}{dt}}\left([\mathbf{x}](\cdot),[\mathbf{v}](\cdot)\right)$

Guaranteed computation of robot trajectories

Rohou, Jaulin, Mihaylova, Le Bars, Veres

Robotics and Autonomous Systems, 2017



Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

Evaluation constraint $\begin{cases} \mathbf{z} = \mathbf{y}(t) \\ \\ \end{cases}$

Tubes contractors

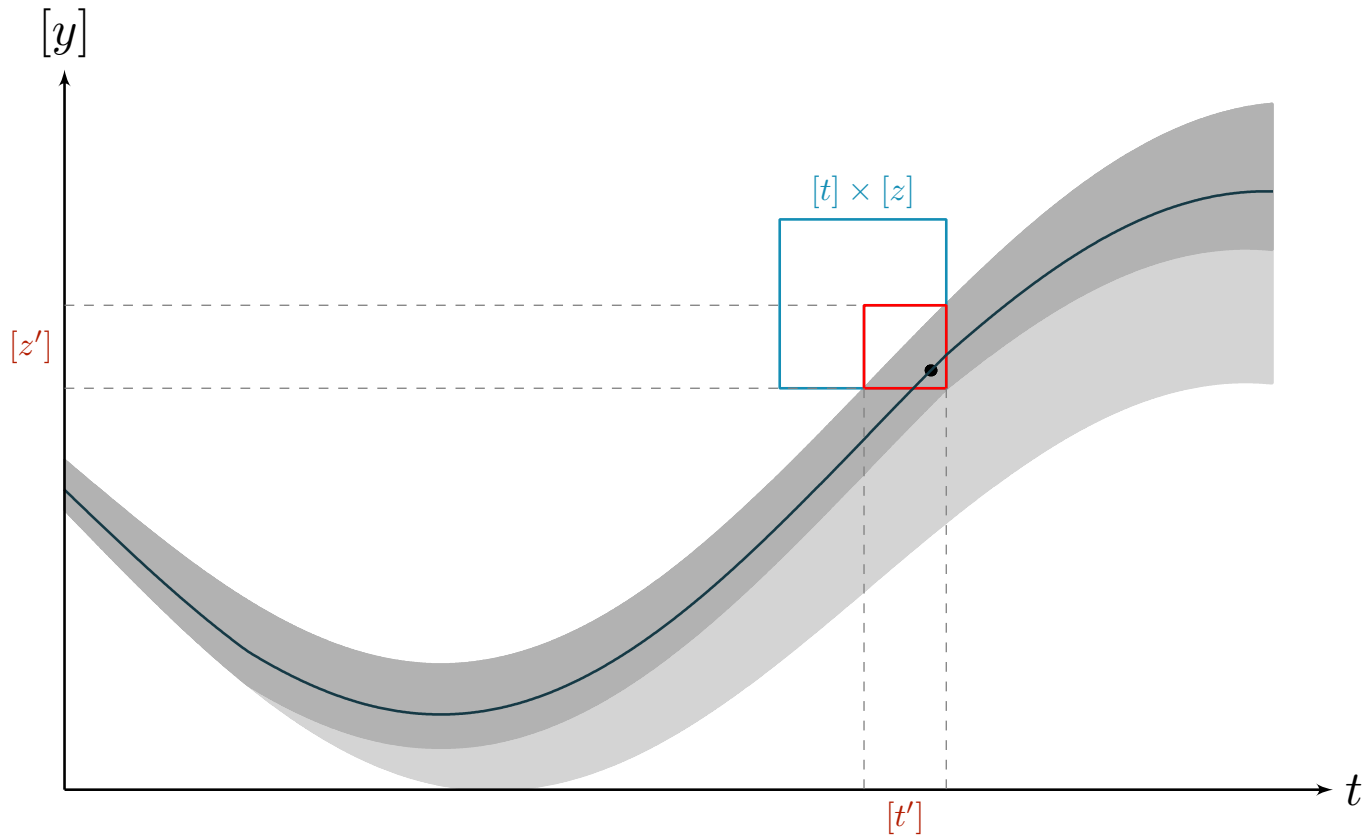
$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

$$\text{Evaluation constraint} \begin{cases} \mathbf{z} = \mathbf{y}(t) \\ \\ t \in [t], \mathbf{z} \in [\mathbf{z}], \mathbf{y}(\cdot) \in [\mathbf{y}](\cdot) \end{cases}$$

Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

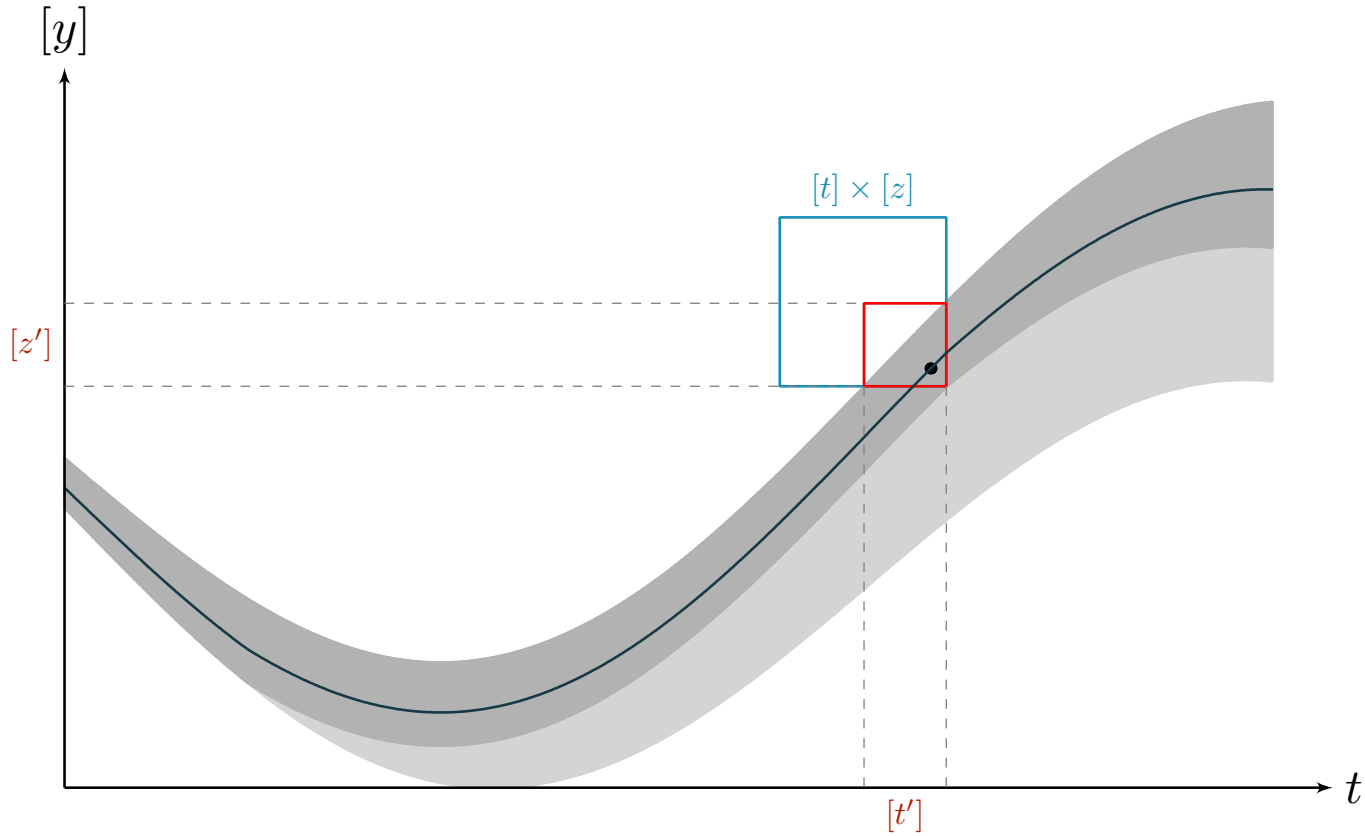
Evaluation constraint $\begin{cases} \mathbf{z} = \mathbf{y}(t) \\ \\ t \in [t], \mathbf{z} \in [\mathbf{z}], \mathbf{y}(\cdot) \in [\mathbf{y}](\cdot) \end{cases}$



Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

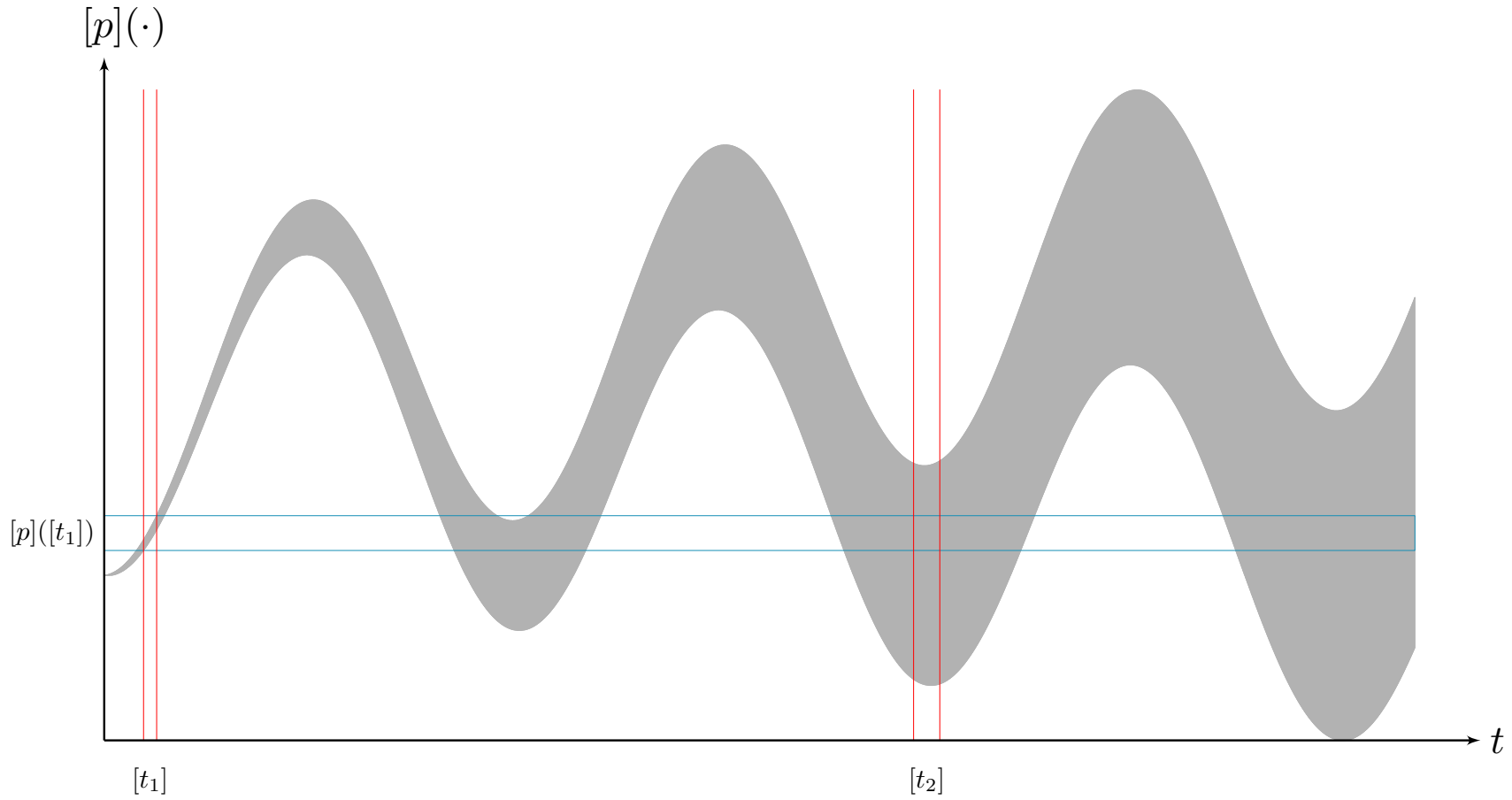
Evaluation constraint
$$\begin{cases} \mathbf{z} = \mathbf{y}(t) \\ \dot{\mathbf{y}}(\cdot) = \mathbf{w}(\cdot) \\ t \in [t], \mathbf{z} \in [\mathbf{z}], \mathbf{y}(\cdot) \in [\mathbf{y}](\cdot), \mathbf{w}(\cdot) \in [\mathbf{w}](\cdot) \end{cases}$$



Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

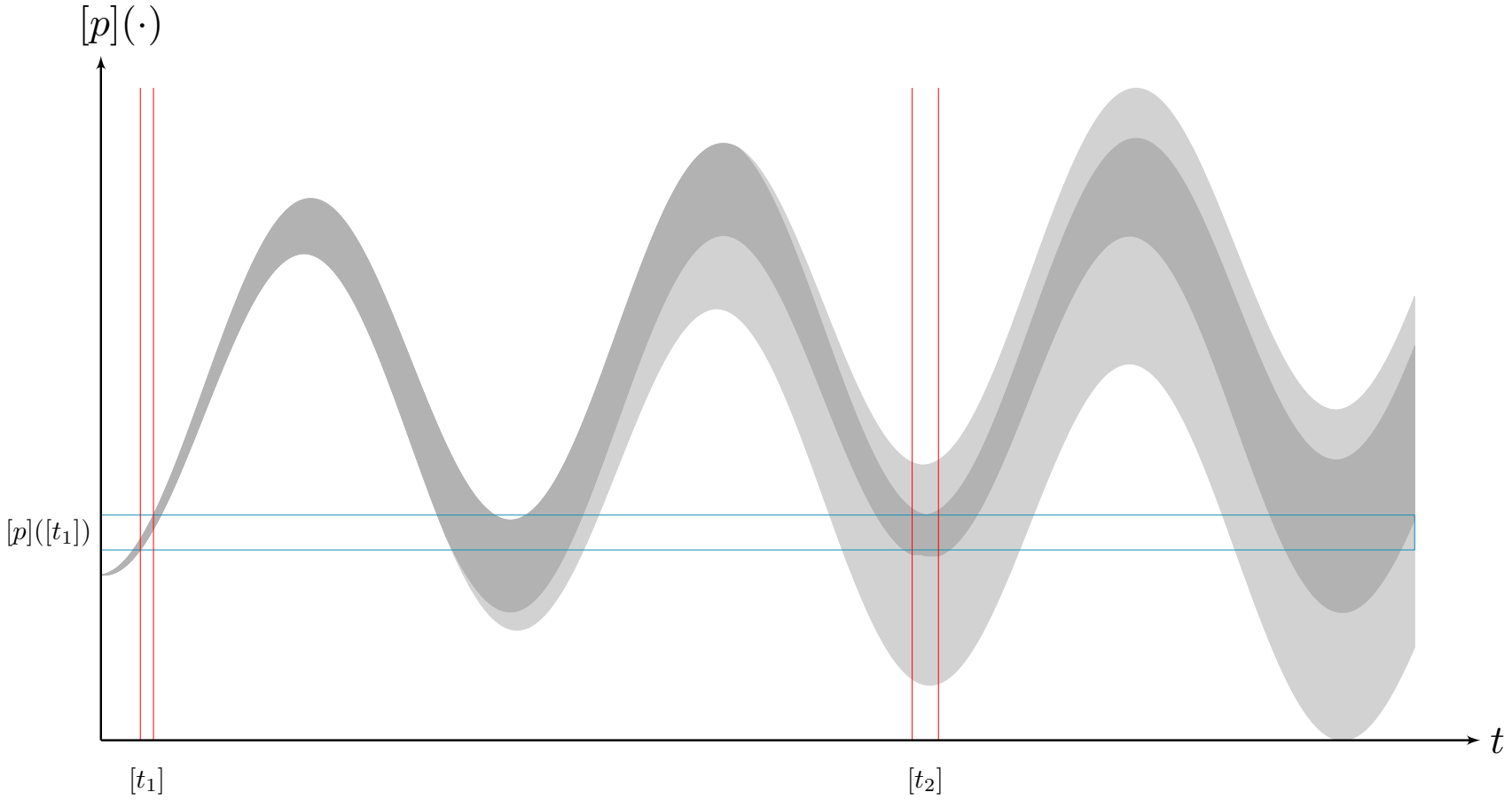
- $\mathbf{t}^* \in [t_1] \times [t_2], \quad \mathbf{p}^*(\cdot) \in [\mathbf{p}](\cdot)$
- constraint: $\mathbf{p}(t_1) = \mathbf{p}(t_2)$



Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

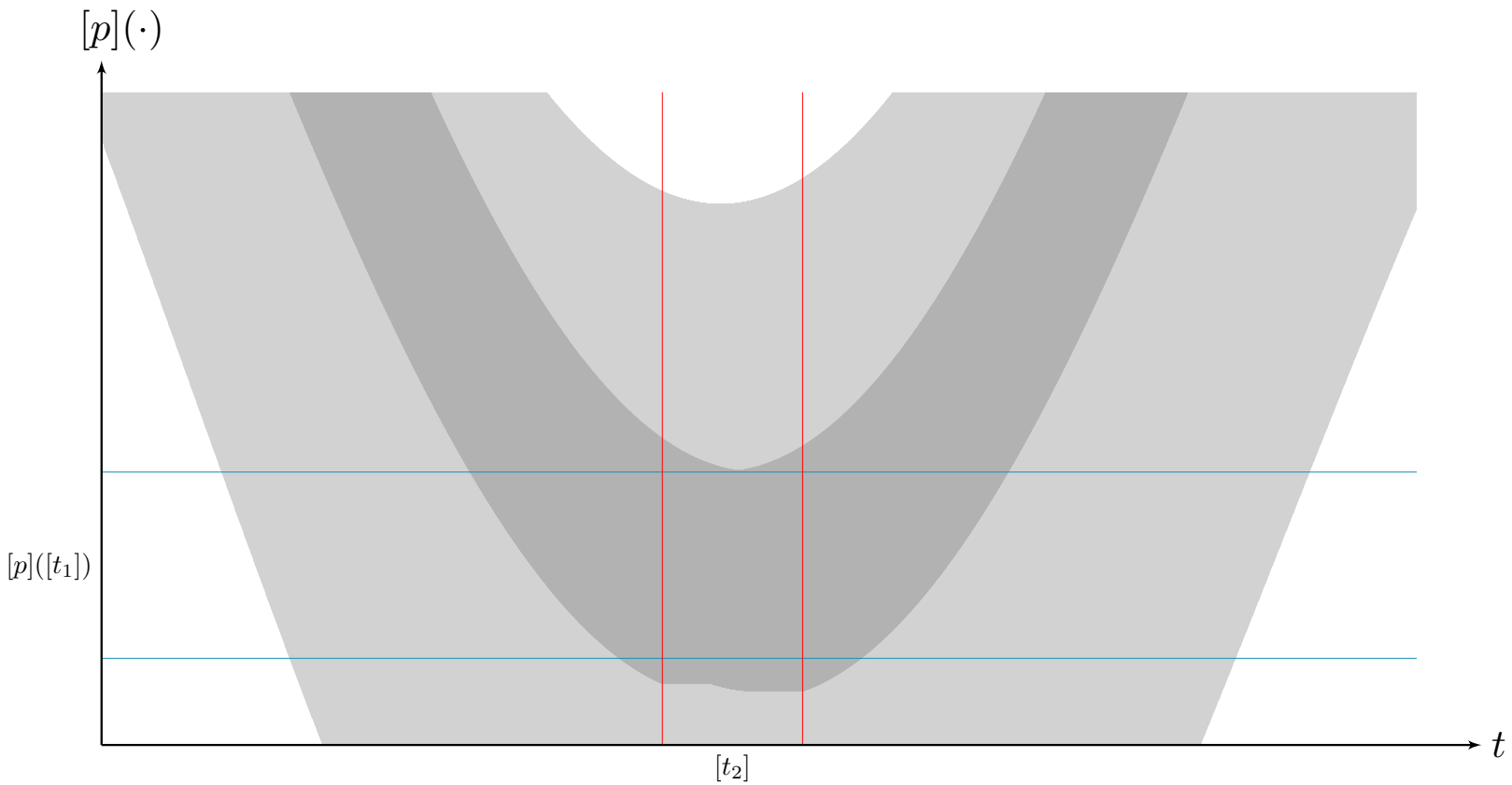
- $\mathbf{t}^* \in [t_1] \times [t_2], \quad \mathbf{p}^*(\cdot) \in [\mathbf{p}](\cdot)$
- constraint: $\mathbf{p}(t_1) = \mathbf{p}(t_2)$



Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

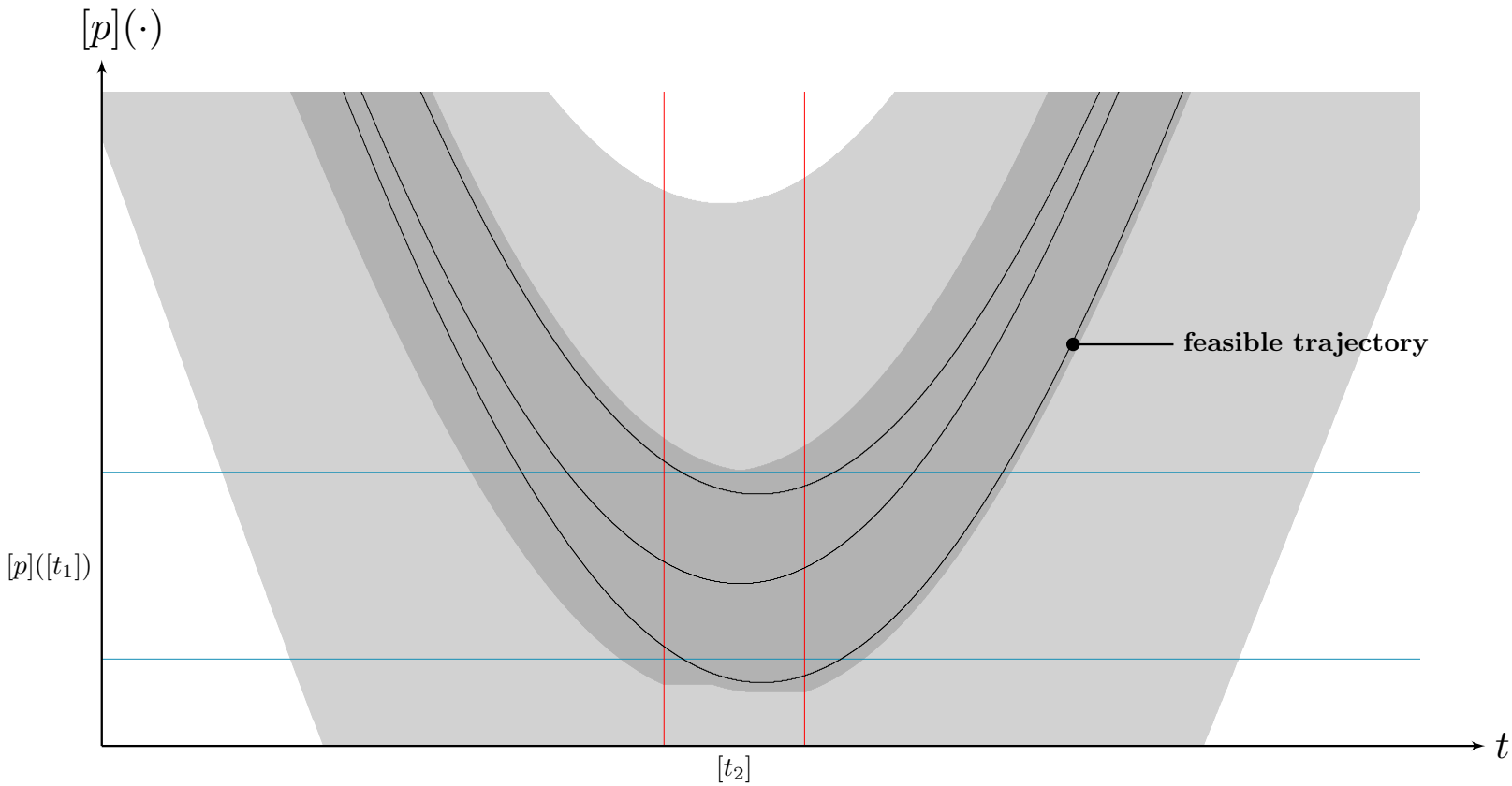
- $\mathbf{t}^* \in [t_1] \times [t_2], \quad \mathbf{p}^*(\cdot) \in [\mathbf{p}](\cdot)$
- constraint: $\mathbf{p}(t_1) = \mathbf{p}(t_2)$



Tubes contractors

$$\mathcal{L}_{\text{eval}}\big(t, \mathbf{z}, \mathbf{y}(\cdot), \mathbf{w}(\cdot)\big)$$

- $\mathbf{t}^* \in [t_1] \times [t_2], \quad \mathbf{p}^*(\cdot) \in [\mathbf{p}](\cdot)$
- constraint: $\mathbf{p}(t_1) = \mathbf{p}(t_2)$



Section 3

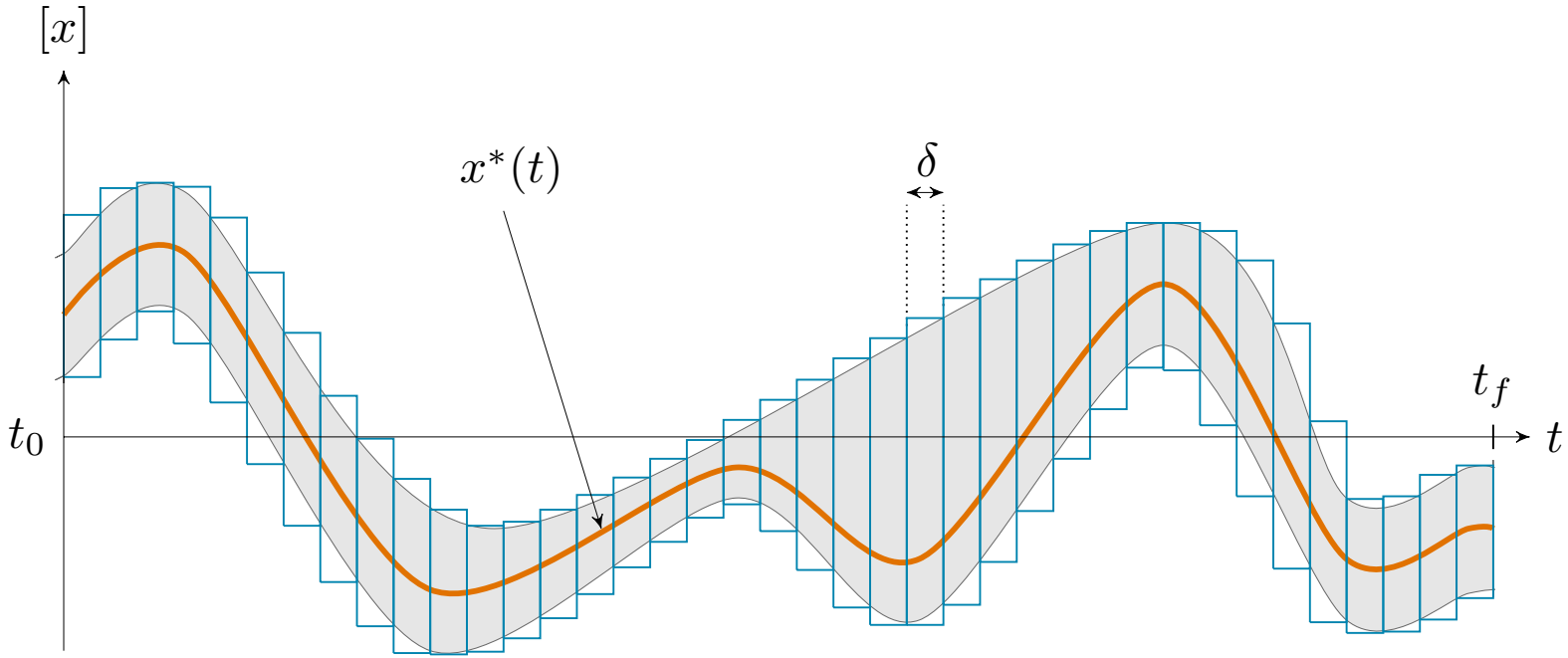
Implementation choices

Implementation choices

Slices representation

Reliable approximation of a tube $[\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$ as an interval of step functions $[\underline{\mathbf{x}^-}(\cdot), \overline{\mathbf{x}^+}(\cdot)]$ such that:

$$\forall t \in [t_0, t_f],\ \underline{\mathbf{x}^-}(t) \leqslant \mathbf{x}^-(t) \leqslant \mathbf{x}^+(t) \leqslant \overline{\mathbf{x}^+}(t) \tag{1}$$



Tube $[x](\cdot)$ enclosing an uncertain trajectory $x^*(\cdot)$

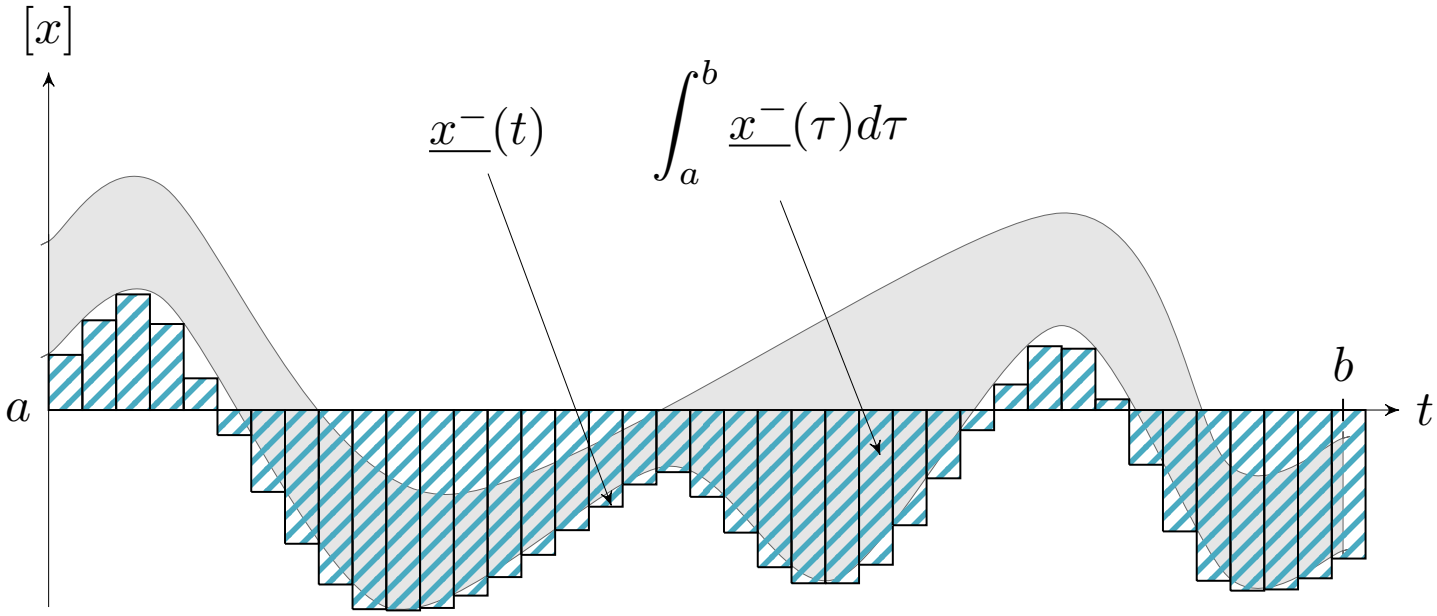
Implementation choices

Slices representation (integrals)

Outer approximation of an integral, computed as

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

[Aubry2013]



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

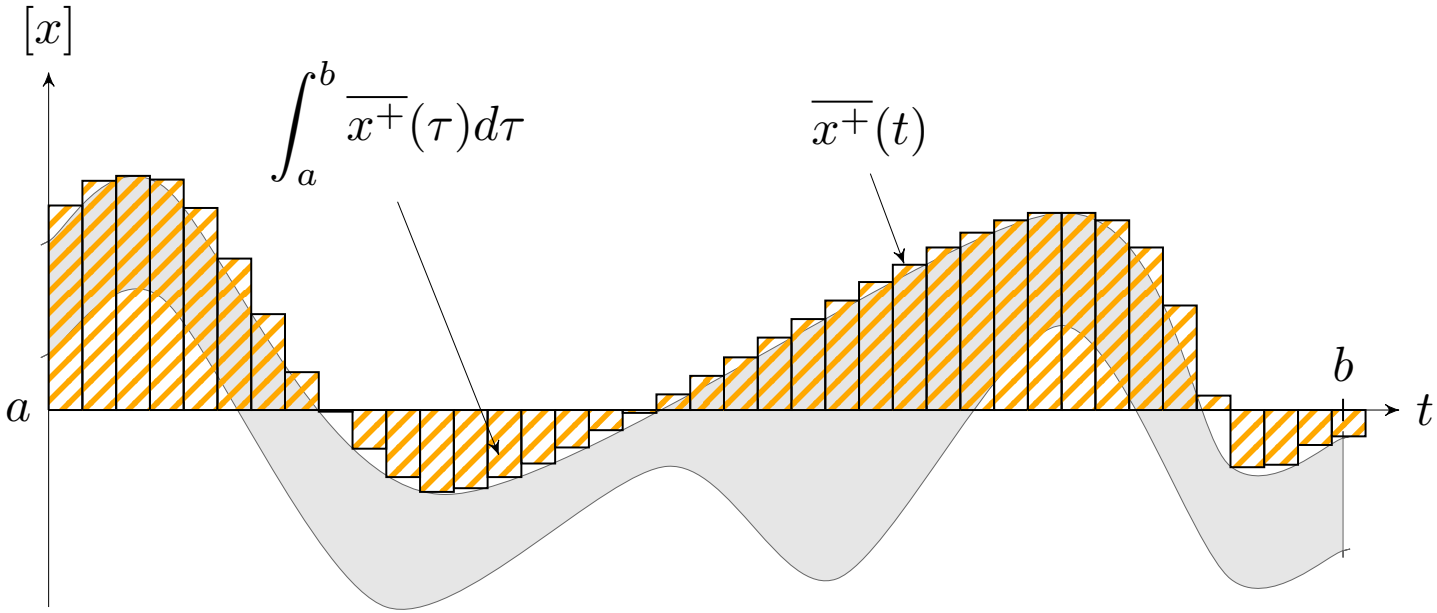
Implementation choices

Slices representation (integrals)

Outer approximation of an integral, computed as

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

[Aubry2013]



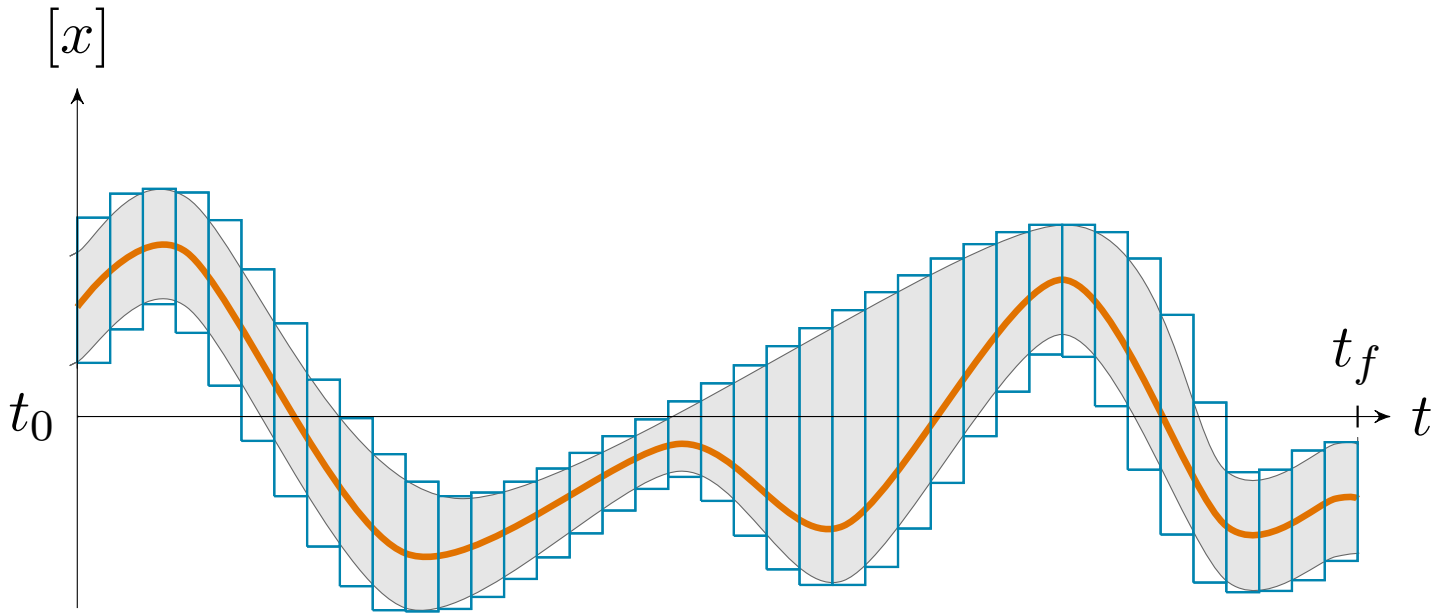
Red area: outer approximation of the upper bound of the tube's integral

Implementation choices

Optimized data representation

Data structure synthesizing the slices: **binary tree**

- fast access to the slices and quick evaluations, inversions, etc.



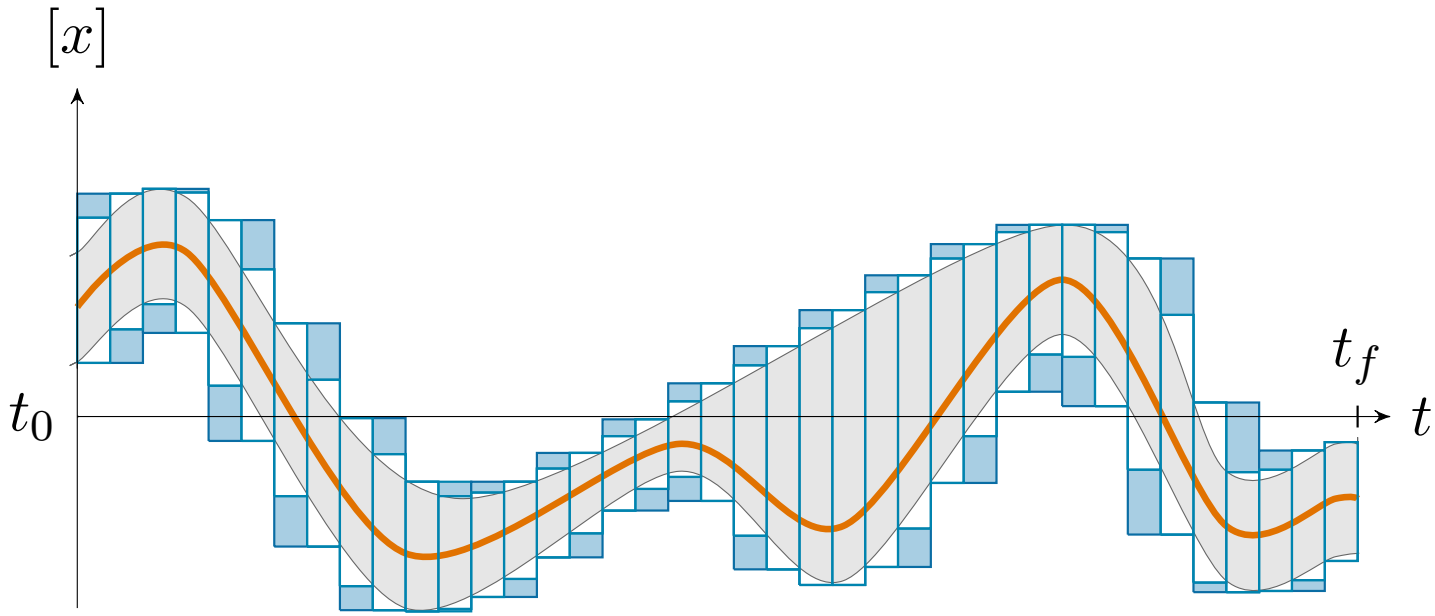
All slices are synthesized into a binary tree

Implementation choices

Optimized data representation

Data structure synthesizing the slices: **binary tree**

- fast access to the slices and quick evaluations, inversions, etc.



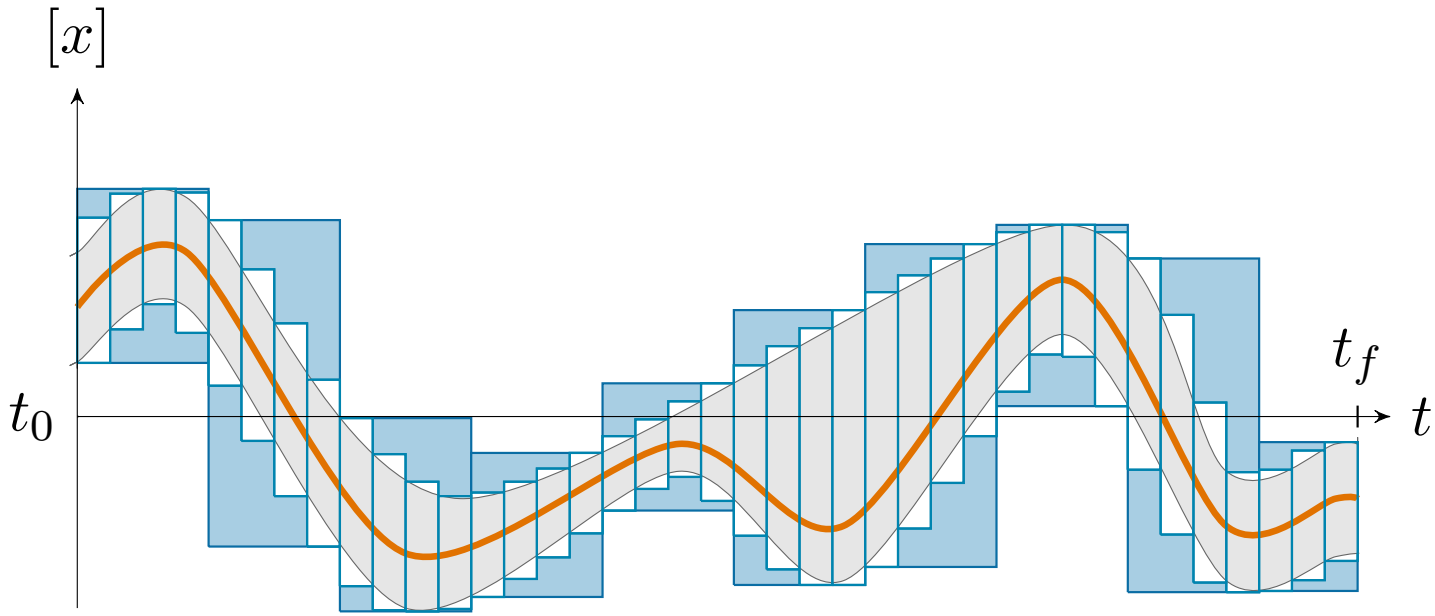
All slices are synthesized into a binary tree

Implementation choices

Optimized data representation

Data structure synthesizing the slices: **binary tree**

- fast access to the slices and quick evaluations, inversions, etc.



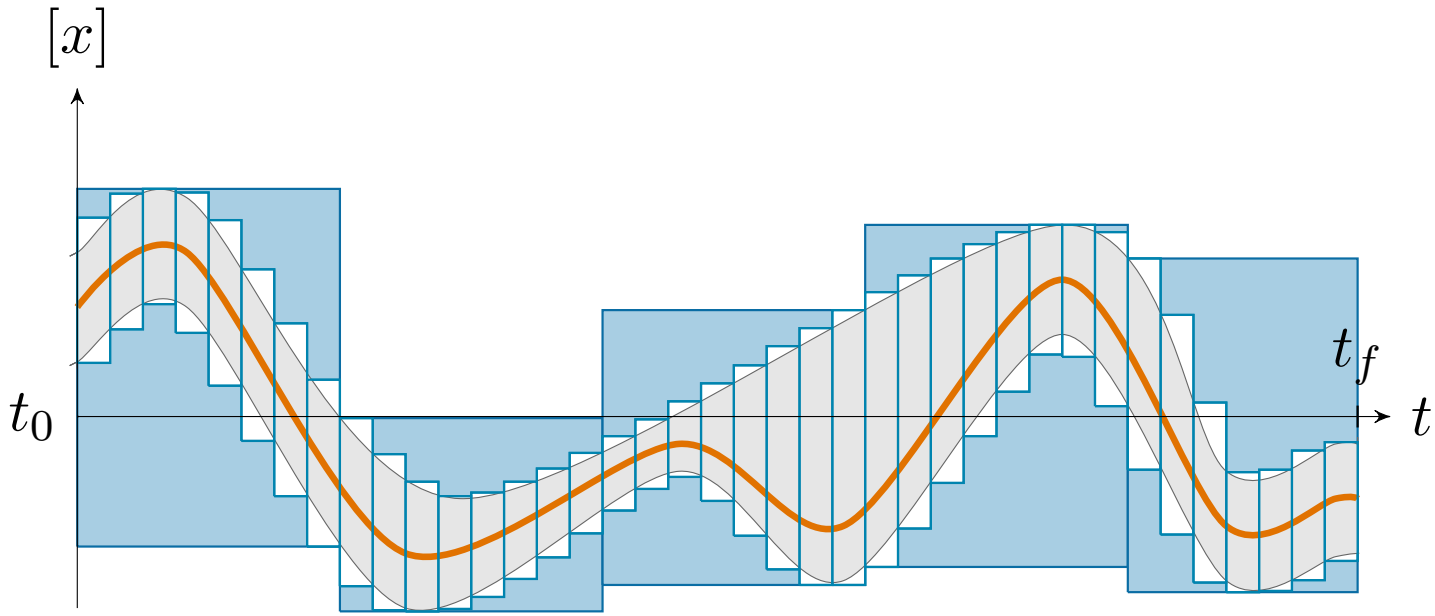
All slices are synthesized into a binary tree

Implementation choices

Optimized data representation

Data structure synthesizing the slices: **binary tree**

- fast access to the slices and quick evaluations, inversions, etc.



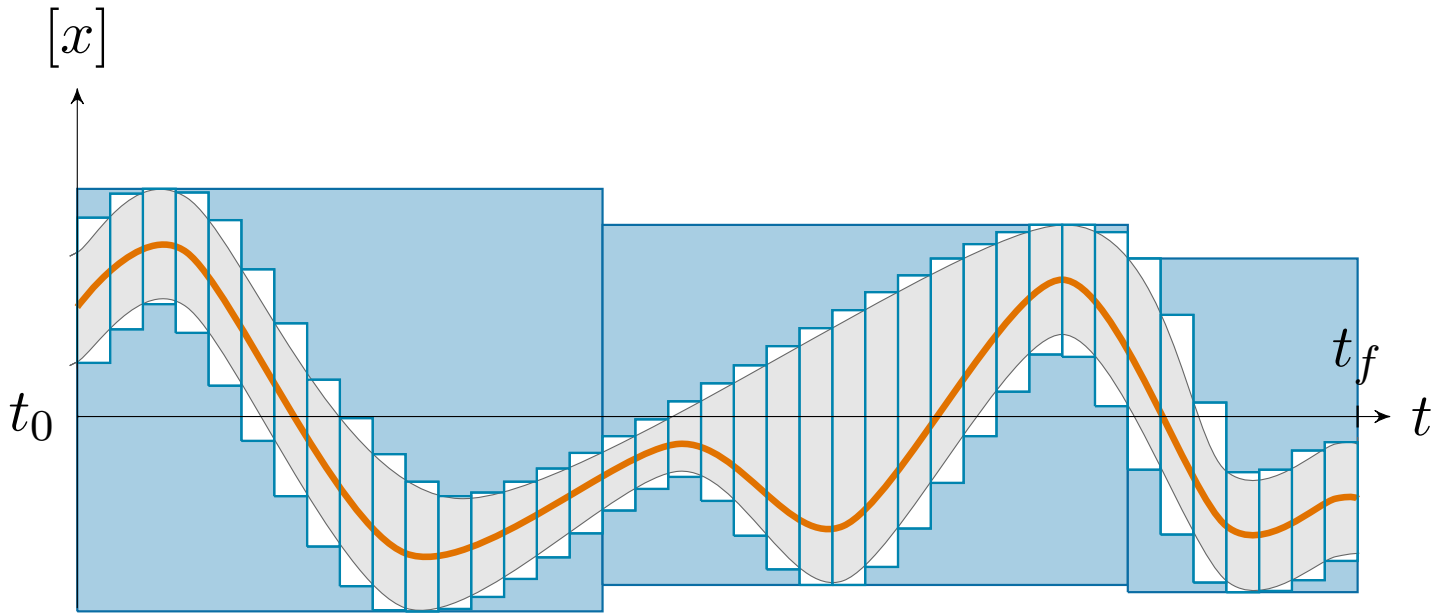
All slices are synthesized into a binary tree

Implementation choices

Optimized data representation

Data structure synthesizing the slices: **binary tree**

- fast access to the slices and quick evaluations, inversions, etc.



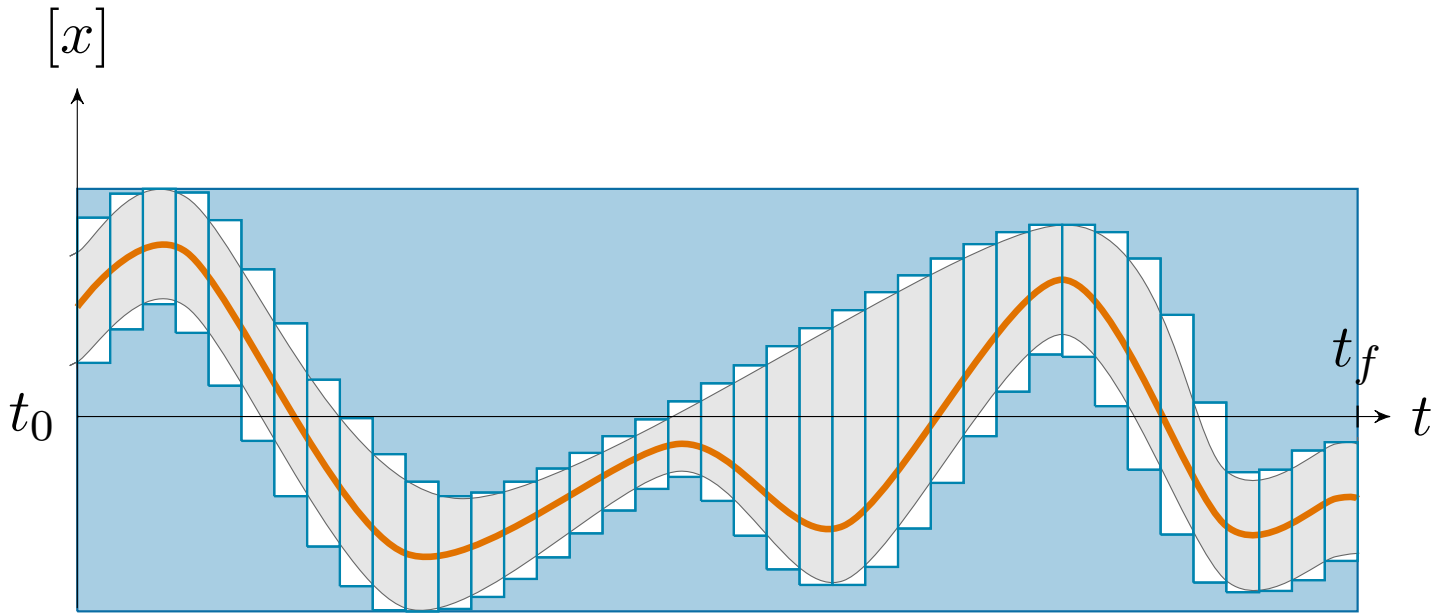
All slices are synthesized into a binary tree

Implementation choices

Optimized data representation

Data structure synthesizing the slices: **binary tree**

- fast access to the slices and quick evaluations, inversions, etc.

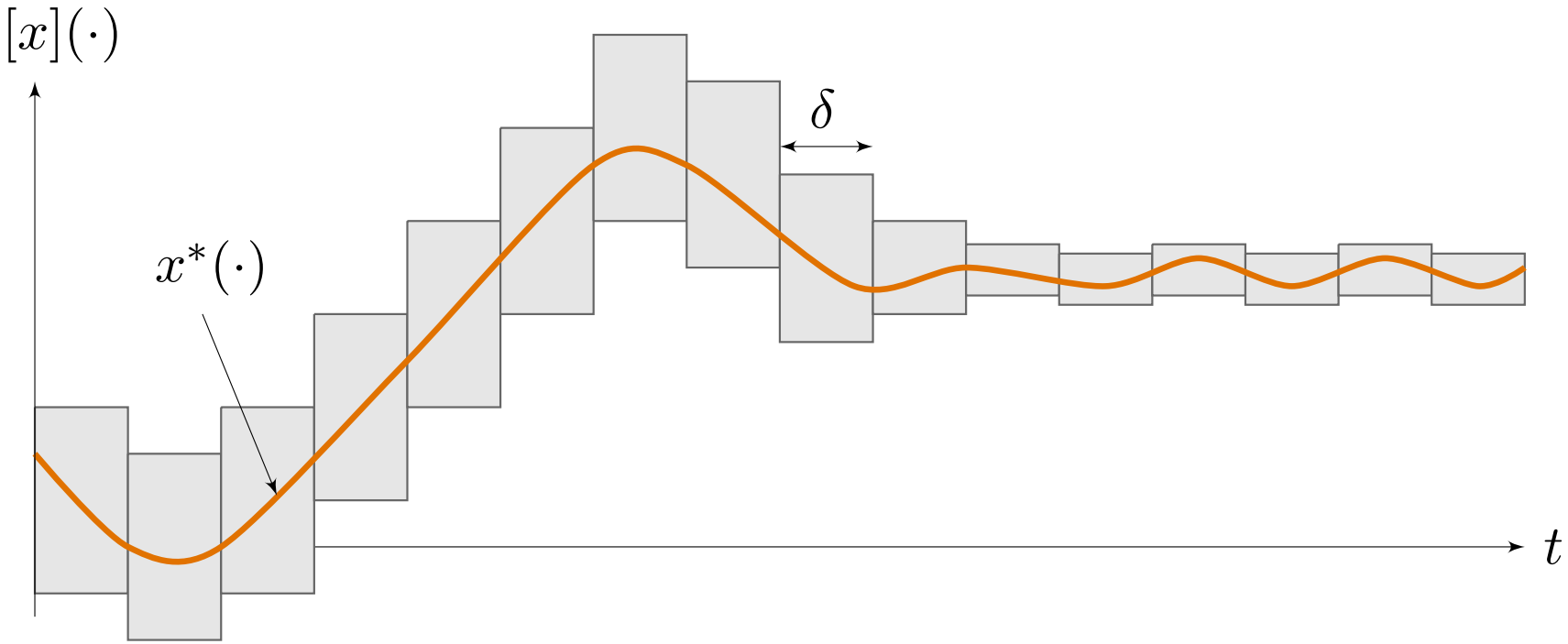


All slices are synthesized into a binary tree

Implementation choices

Non constant slicing representation

New feature of **Tubex v2.0**:

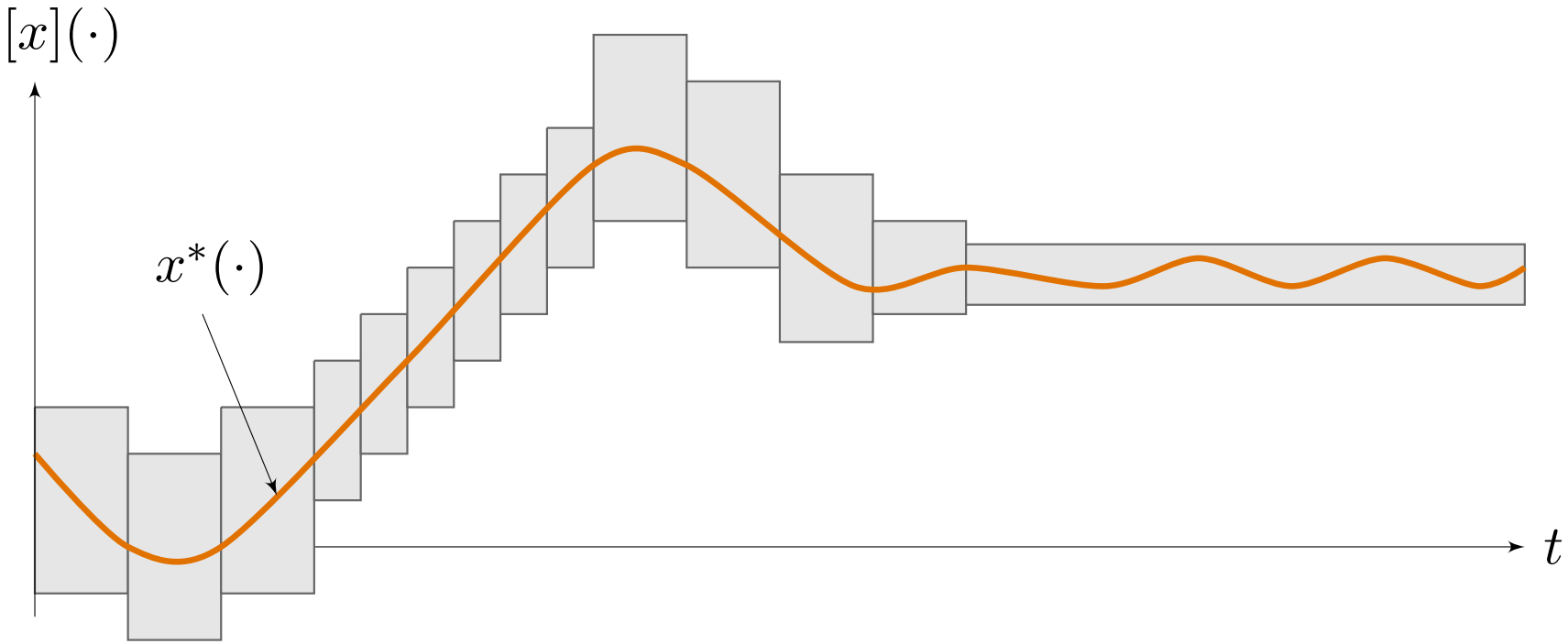


Slices of different temporal width

Implementation choices

Non constant slicing representation

New feature of **Tubex v2.0**:

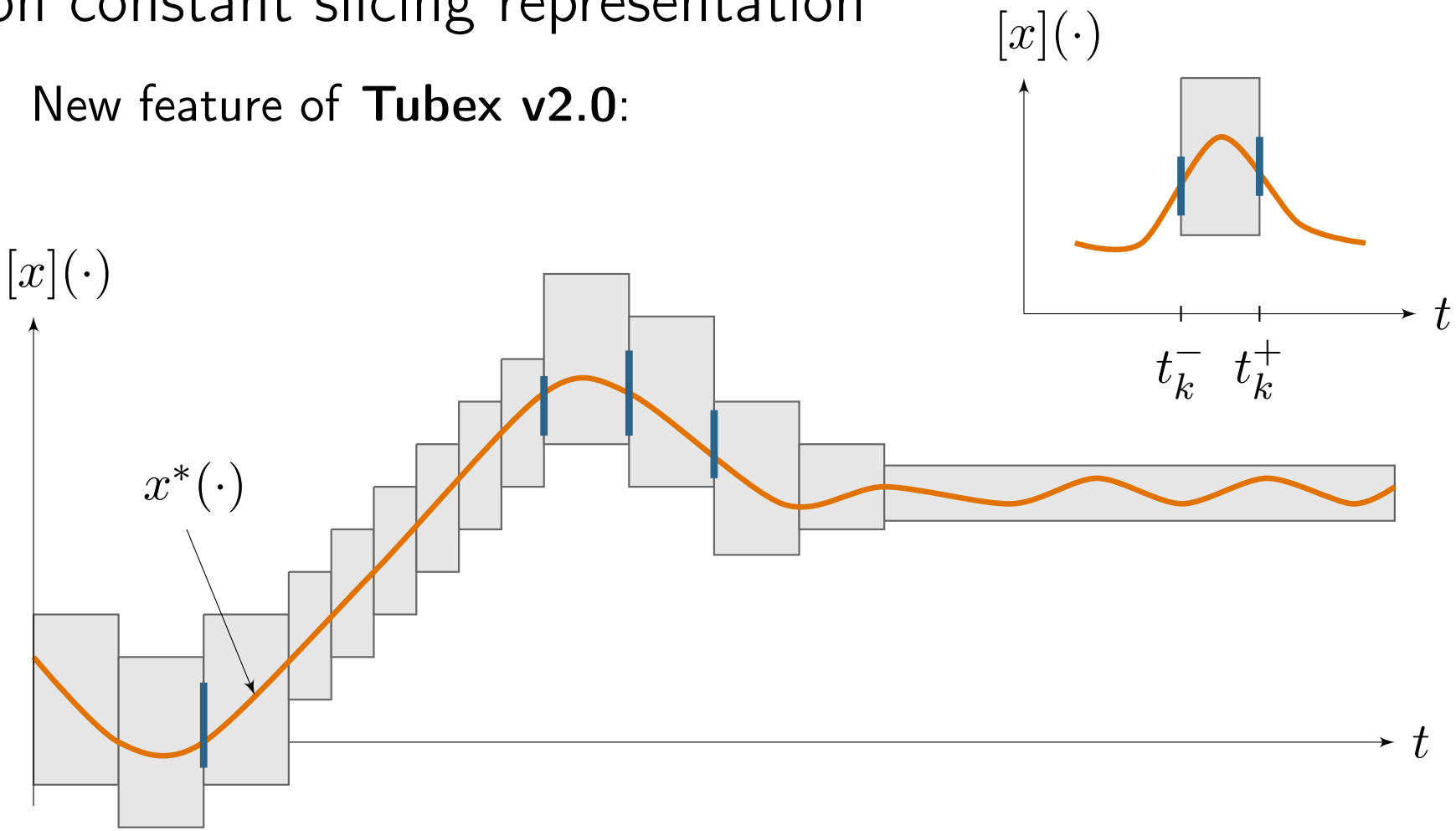


Slices of different temporal width

Implementation choices

Non constant slicing representation

New feature of **Tubex v2.0**:

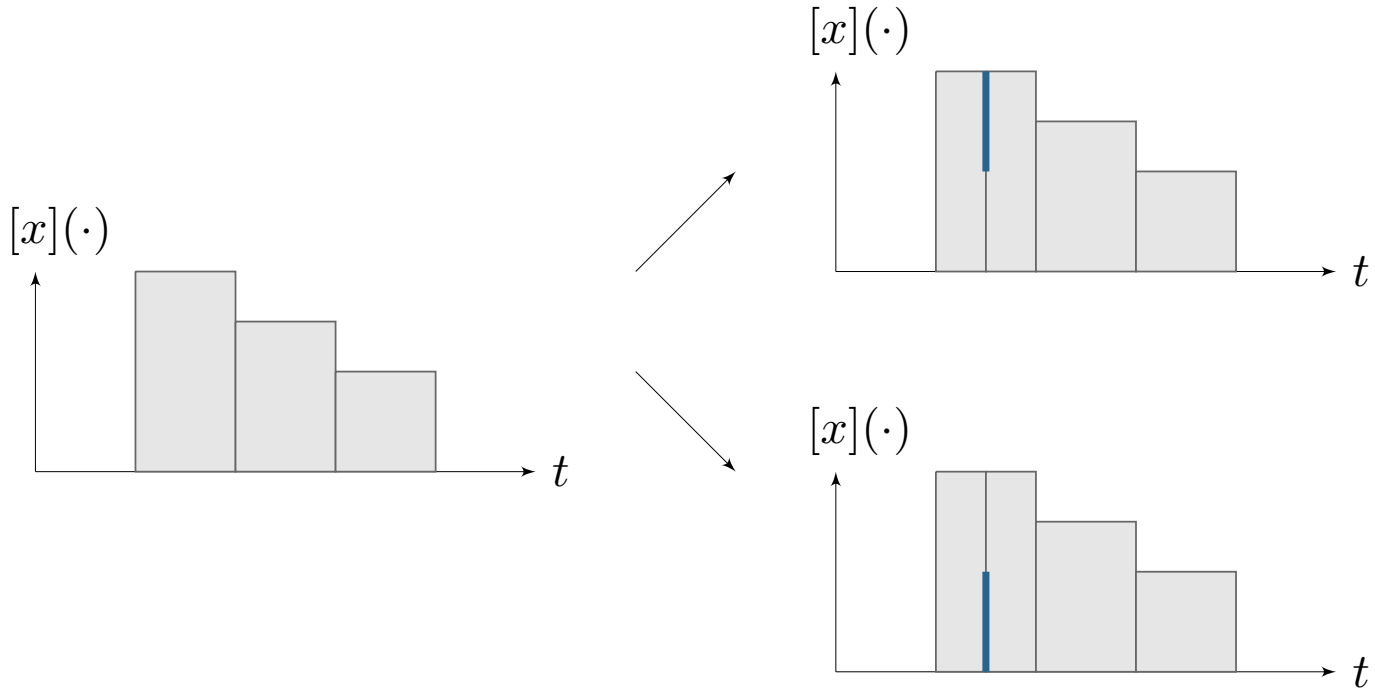


Slices of different temporal width, with gates

Implementation choices

Bisection of a tube

Illustration:



Bisection $\Longrightarrow$ one more slice and gate

Implementation choices

Complementary tools

Some other features of Tubex:

- **graphical** tools (based on VIBes viewer)
- **serialization** methods (binary storage)
- **robotics** applications (data loaders, loops computations, etc.)

Towards a **solver**...

- Contredo ANR project
- sampling and bisection of tubes: exploration of solutions
- solving BVP, DAE, etc. systems

Section 4

Example

Example

Using Tubex

```
1   /* ========== INITIALIZATION ========== */
2
3   Interval domain(0,10);
4   TubeVector x(domain, 6);
5   double timestep = 0.01;
6
7   x[0] = Tube(domain, timestep, // tube [x](.)
8   tubex::Function("(t-5)^2 + [-0.5,0.5]"));
9   x[1] = Tube(domain, timestep, // tube [y](.)
10  tubex::Function("-4*cos(t-5) + [-0.5,0.5] + 0.1*(t-3.3)
        ^2*[-2,2]"));
11
12  /* ========== ARITHMETIC ========== */
13
14  x[2] = x[0] + x[1];    // tube [a](.)
15  x[3] = sin(x[0]);      // tube [b](.)
16  x[4] = x[0].primitive(); // tube [c](.)
17  x[5] = abs(x[1]);      // tube [d](.)
```

Example

Using Tubex

```
1    /* ============ GRAPHICS ============ */
2
3    vibes::beginDrawing();
4    VIBesFigTubeVector fig_x("x", 0, 4);
5    fig_x.set_properties(100, 100, 600, 300);
6    fig_x.add_tubevector(&x, "x");
7    fig_x.add_trajectoryvector(&traj_x, "x*", "blue");
8    fig_x.show();
9    vibes::endDrawing();
```

Example

Using Tubex

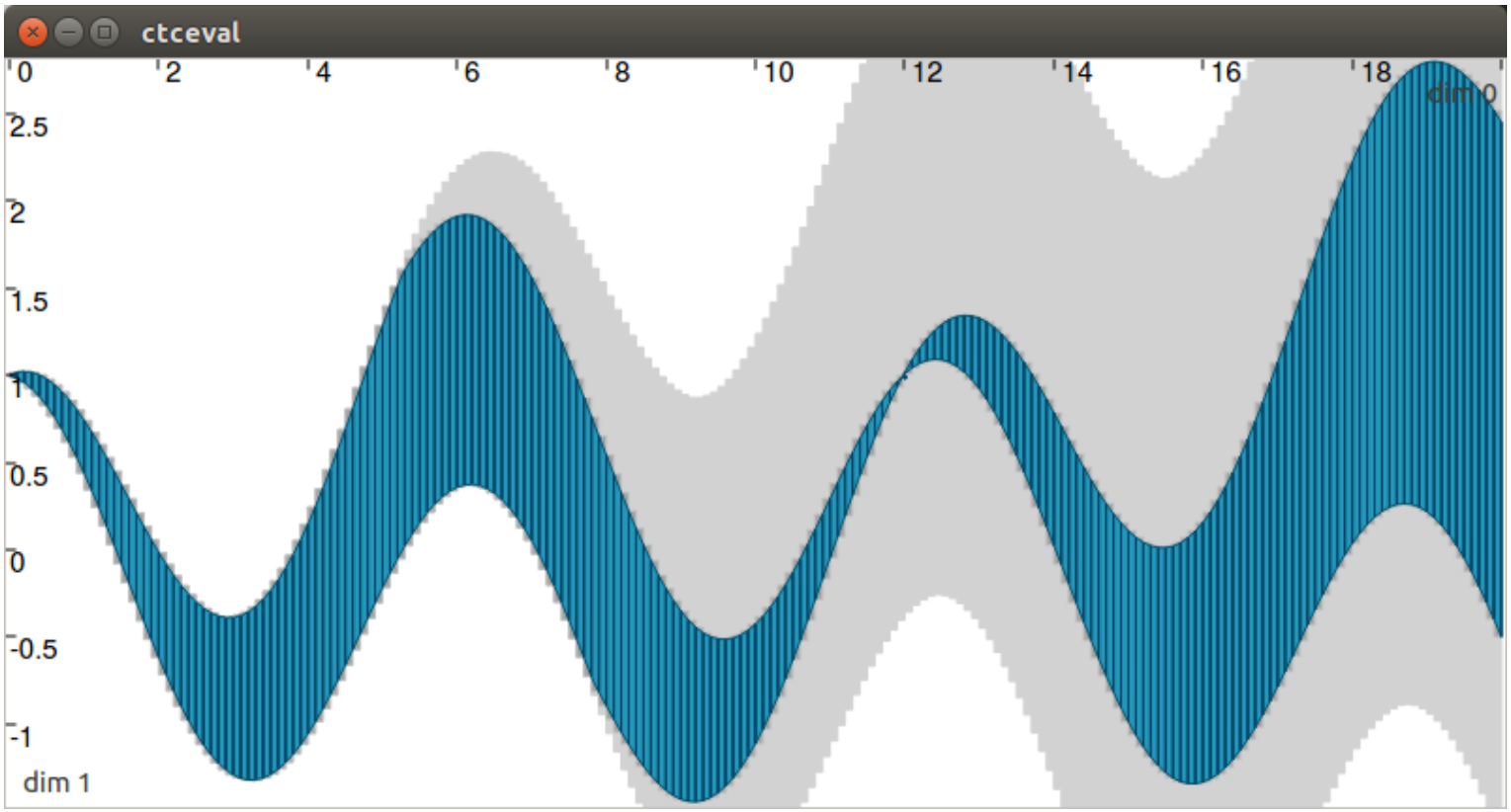
```
1   tubex::CtcFwdBwd ctc_fwdbwd(
2     tubex::Function("x", "y", "a", "p", "q",
3       "(x + y - a ;
4         atan(y) - p ;
5         2*sin(0.5*a) + sqrt(exp(p^2)) - q)"));
6
7   ctc_fwdbwd.contract(x);
```

- possibility to use any IBEX contractor object on tubes

Example

Using Tubex

```
1  CtcEval ctc_eval;
2  Interval t(12.); // evaluation domain
3  Interval z(1.); // evaluation value
```

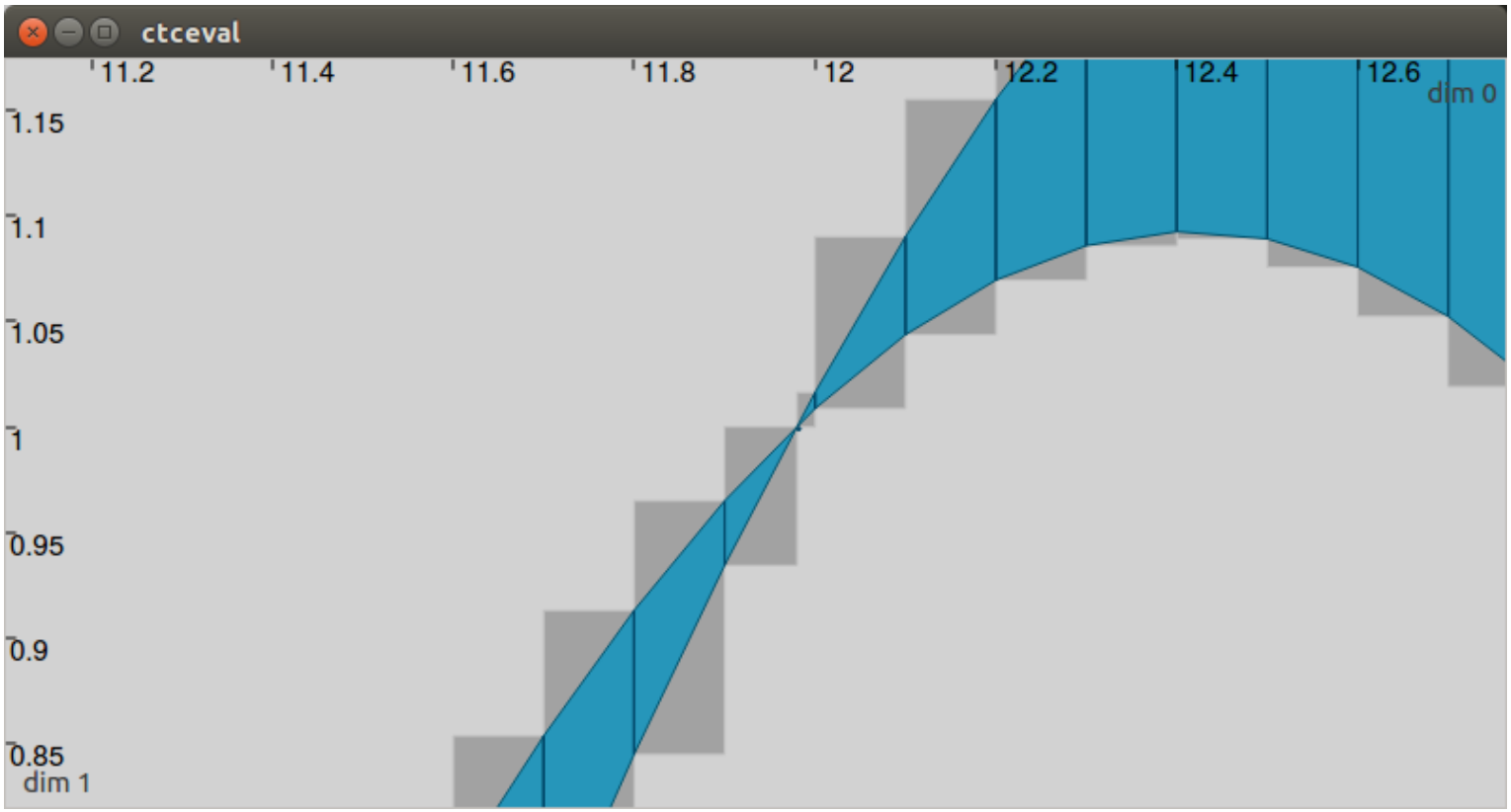


Tube evaluation

Example

Using Tubex

```
1  CtcEval ctc_eval;
2  Interval t(12.); // evaluation domain
3  Interval z(1.); // evaluation value
```



Tube evaluation

Section 5

Conclusion

Conclusion

Hot todo-list:

- python wrapping (pyIbex-based)
- user documentation $\Longrightarrow$ **v2.0 release**

Todo-list:

- additional primitive tube contractors
- robotics illustrations (topological degree)

Other ideas:

- reliable and fast polygon library
- connection with already-existing IVP solvers?
- extend IBEX's parser?