Improving validated computation of Viability Kernels

Benjamin Martin[1] Olivier Mullier[2]

[1]LIX, École Polytechnique
[2]U2IS, ENSTA Paristech

13 April 2018
International Conference on Hybrid Systems: Computation and Control
Porto, Portugal

Research financially supported by the DGA MRIS

Controlled dynamical system and viability kernel

We consider the following controlled nonlinear system:

$$(S) \begin{cases} \dot{x} = f(x, u) \\ x(0) = x_0 \end{cases}$$

with $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $x(t) \in \mathbb{R}^n$ the states, for all $t \geqslant 0$.

Controlled dynamical system and viability kernel

We consider the following controlled nonlinear system:

$$(S) \begin{cases} \dot{x} = f(x, u) \\ x(0) = x_0 \end{cases}$$

with $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $x(t) \in \mathbb{R}^n$ the states, for all $t \geqslant 0$.

Viability Kernel of K

Given a set of states K, its viability kernel $\text{Viab}_S(K)$ is the set of all **initial conditions** for which there always **exists a trajectory** remaining inside K for an indefinite amount of time.

Controlled dynamical system and viability kernel

We consider the following controlled nonlinear system:

$$(S) \begin{cases} \dot{x} = f(x, u) \\ x(0) = x_0 \end{cases}$$

with $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $x(t) \in \mathbb{R}^n$ the states, for all $t \geqslant 0$.

Viability Kernel of K

$$\text{Viab}_S(K) := \{x_0 \in K | (\exists u(t) \in \mathcal{U})(\forall t \in [0, +\infty])(\varphi(x_0, u(t), t) \in K)\},$$

$\varphi(x_0, u(t), t)$ is the value of $x(t)$ starting from x_0 with control input $u(.)$.

Computing the viability kernel

Motivation

Computation of validated inner-approximation of viability kernels of non-linear continuous time (control) systems for safety analysis.

Computing the viability kernel

Motivation

Computation of validated inner-approximation of viability kernels of non-linear continuous time (control) systems for safety analysis.

Some approaches from the literature:

- discretization-based [Saint-Pierre, 1994, Deffuant et al., 2007, Girard, 2012, Reissig et al., 2017].
- set-oriented or Lyapunov-like methods [Kaynama et al., 2012, She and Xue, 2013, Korda et al., 2013].
- interval based framework [Monnet et al., 2016].

Computing the viability kernel

Motivation

Computation of validated inner-approximation of viability kernels of non-linear continuous time (control) systems for safety analysis.

Some approaches from the literature:

- discretization-based [Saint-Pierre, 1994, Deffuant et al., 2007, Girard, 2012, Reissig et al., 2017].
- set-oriented or Lyapunov-like methods [Kaynama et al., 2012, She and Xue, 2013, Korda et al., 2013].
- interval based framework [Monnet et al., 2016].

Sketch of the approach [Monnet et al., 2016]

A two phase method:

Sketch of the approach [Monnet et al., 2016]

A two phase method:

1. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

Sketch of the approach [Monnet et al., 2016]

A two phase method:

I. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

II. improving the inner-approximation by computing iteratively its capture basin (implemented by validated numerical integration).

Sketch of the approach [Monnet et al., 2016]

A two phase method:

I. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

II. improving the inner-approximation by computing iteratively its capture basin (implemented by validated numerical integration).

Capture basin

Given a target set T, the capture basin of T within K for a time horizon t_{end}, $\text{Capt}_S^{t_{end}}(K, T)$, is the **the set of initial conditions** for which **there exists a trajectory** reaching T before t_{end} and **without exiting** K.

Sketch of the approach [Monnet et al., 2016]

A two phase method:

I. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

II. improving the inner-approximation by computing iteratively its capture basin (implemented by validated numerical integration).

Capture basin

$$\mathrm{Capt}_S^{t_{end}}(K,T) := \left\{ x_0 \in K \;\middle|\; \begin{array}{r} (\exists \tilde{t} \in [0, t_{\mathrm{end}}])(\exists \tilde{u} \in \mathcal{U})(\varphi(x_0, \tilde{u}(\tilde{t}), \tilde{t}) \in T \\ \wedge (\forall t \in [0, \tilde{t}])(\varphi(x_0, \tilde{u}(t), t) \in K)) \end{array} \right\}$$

Sketch of the approach [Monnet et al., 2016]

A two phase method:

I. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

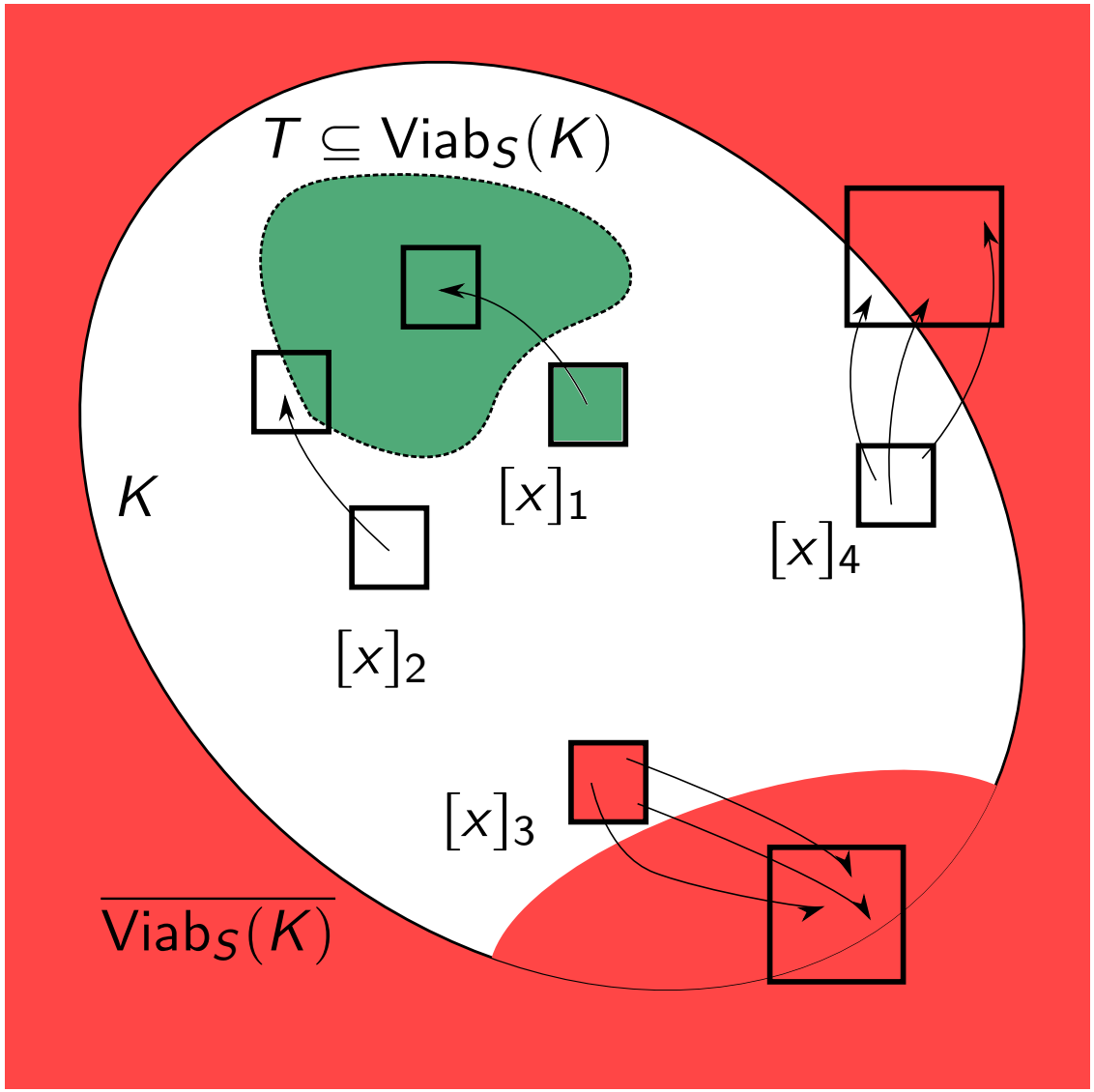
II. improving the inner-approximation by computing iteratively its capture basin (implemented by validated numerical integration).

Capture basin

$$\mathrm{Capt}_S^{t_{end}}(K,T) := \left\{ x_0 \in K \;\middle|\; \begin{array}{r} (\exists \tilde{t} \in [0, t_{\mathrm{end}}])(\exists \tilde{u} \in \mathcal{U})(\varphi(x_0, \tilde{u}(\tilde{t}), \tilde{t}) \in T \\ \wedge (\forall t \in [0, \tilde{t}])(\varphi(x_0, \tilde{u}(t), t) \in K)) \end{array} \right\}$$

$$T \subseteq \mathrm{Viab}_S(K) \implies \mathrm{Capt}_S^{t_{end}}(K,T) \subseteq \mathrm{Viab}_S(K),\ \forall t_{\mathrm{end}} \geqslant 0$$

Illustration



Interval Analysis

Notations:

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

$\mathbb{IR}$: set of real intervals. Boxes: interval vectors.

Interval Analysis

Notations:

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

$\mathbb{IR}$: set of real intervals. Boxes: interval vectors.

Main tools:

Interval extensions: over-approximation of functions over interval inputs.

Contractors: contract a box removing inconsistent values (*w.r.t* some properties).

Interval Newton: prove existence of solutions to a system of equations.

Validated integration: over-approximation of trajectories.

[Neumaier, 1991, Jaulin et al., 2001]

Goal

Two phase algorithm [Monnet et al., 2016] is expensive.

$\Longrightarrow$ towards improving it:

Goal

Two phase algorithm [Monnet et al., 2016] is expensive.

$\Longrightarrow$ towards improving it:

- enhancing the improvement phase by more efficient validated numerical integration (done by Olivier Mullier, not detailed in this talk)

Goal

Two phase algorithm [Monnet et al., 2016] is expensive.

$\Longrightarrow$ towards improving it:

- enhancing the improvement phase by more efficient validated numerical integration (done by Olivier Mullier, not detailed in this talk)
- enhancing the first phase by generating large inner-approximations (this talk and paper)

Goal

Two phase algorithm [Monnet et al., 2016] is expensive.

$\Longrightarrow$ towards improving it:

- enhancing the improvement phase by more efficient validated numerical integration (done by Olivier Mullier, not detailed in this talk)
- enhancing the first phase by generating large inner-approximations (this talk and paper)

Idea

Taking more time for computing larger initial inner-approximation may reduce significantly the cost of the improvement phase.

First phase: original method

Find a function $h : \mathbb{R}^n \to \mathbb{R}$ of the states *s.t*

$$(\forall x \in K),\ h(x) = 0 \implies \exists u \in \mathcal{U},\ \dot{h}(x,u) < 0, \tag{1}$$

with $\dot{h}(x,u) := \langle \nabla_x h(x), f(x,u) \rangle$.

First phase: original method

Find a function $h : \mathbb{R}^n \to \mathbb{R}$ of the states *s.t*

$$(\forall x \in K),\ h(x) = 0 \implies \exists u \in \mathcal{U},\ \dot{h}(x,u) < 0, \tag{1}$$

with $\dot{h}(x,u) := \langle \nabla_x h(x), f(x,u) \rangle$.

Property

Let h be a function satisfying (1), define $\mathcal{H} := \{x \in K |\ h(x) \leqslant 0\}$. Then if $\mathcal{H} \subset \mathrm{int} K$, then $\mathcal{H} \subset \mathrm{Viab}_S(K)$.

First phase: original method

Find a function $h : \mathbb{R}^n \to \mathbb{R}$ of the states *s.t*

$$(\forall x \in K),\ h(x) = 0 \implies \exists u \in \mathcal{U},\ \dot{h}(x,u) < 0, \tag{1}$$

with $\dot{h}(x,u) := \langle \nabla_x h(x), f(x,u) \rangle$.

Property

Let h be a function satisfying (1), define $\mathcal{H} := \{x \in K |\ h(x) \leqslant 0\}$. Then if $\mathcal{H} \subset \mathrm{int} K$, then $\mathcal{H} \subset \mathrm{Viab}_S(K)$.

Implementation in [Monnet et al., 2016]: construction of Lyapunov-like function from the linearized system around equilibrium points.

Analysis of the first phase

Original implementation

Fast but may fail to produce large inner-approximations.

Analysis of the first phase

Original implementation

Fast but may fail to produce large inner-approximations.

Towards a generalization

Given a family of parametric template functions $h_p : \mathbb{R}^n \to \mathbb{R}$, find $p \in \mathcal{P} \subset \mathbb{R}^q$ such that:

$$(\forall x \in K)\ (h_p(x) = 0 \implies \exists u \in \mathcal{U},\ \dot{h}_p(x,u) < 0) \quad (2)$$

$$\wedge\ (x \in \partial K \implies h_p(x) > 0) \quad (3)$$

($+$ find $p \in \mathcal{P}$ maximizing $\mathrm{vol}\ \mathcal{H}_p$)

Analysis of the first phase

Original implementation

Fast but may fail to produce large inner-approximations.

Towards a generalization

Given a family of parametric template functions $h_p : \mathbb{R}^n \to \mathbb{R}$, find $p \in \mathcal{P} \subset \mathbb{R}^q$ such that:

$$(\forall x \in K)\ (h_p(x) \neq 0 \quad \vee \quad \exists u \in \mathcal{U},\ \dot{h}_p(x,u) < 0) \tag{2}$$

$$\wedge\ (x \notin \partial K \quad \vee \quad h_p(x) > 0) \tag{3}$$

($+$ find $p \in \mathcal{P}$ maximizing $\mathrm{vol}\ \mathcal{H}_p$)

Analysis of the first phase

Original implementation

Fast but may fail to produce large inner-approximations.

Towards a generalization

Given a family of parametric template functions $h_p : \mathbb{R}^n \to \mathbb{R}$, find $p \in \mathcal{P} \subset \mathbb{R}^q$ such that:

$$(\forall x \in K)\ (h_p(x) \neq 0 \quad \vee \quad \exists u \in \mathcal{U},\ \dot{h}_p(x,u) < 0) \tag{2}$$
$$\wedge\ (x \notin \partial K \quad \vee \quad h_p(x) > 0) \tag{3}$$

($+$ find $p \in \mathcal{P}$ maximizing $\mathrm{vol}\ \mathcal{H}_p$)

Quantified Constraint Satifisaction Problem (QCSP)

How to solve a QCSP

CSC-FPS Algorithm [Djaballah et al., 2017]

Validated interval algorithm for computing barrier functions (uncontrolled case). Close to our problem.

How to solve a QCSP

CSC-FPS Algorithm [Djaballah et al., 2017]

Validated interval algorithm for computing barrier functions (uncontrolled case). Close to our problem.

Computable Sufficient Condition (CSC) - Feasible Point Searcher (FPS):

- FPS (branching algorithm on p): explore the set of parameters, and find a valid one (checked by CSC).

How to solve a QCSP

CSC-FPS Algorithm [Djaballah et al., 2017]

Validated interval algorithm for computing barrier functions (uncontrolled case). Close to our problem.

Computable Sufficient Condition (CSC) - Feasible Point Searcher (FPS):

- FPS (branching algorithm on p): explore the set of parameters, and find a valid one (checked by CSC).
- CSC (branching algorithm on x): from a box $[p]$ checks whether (a.) a $\tilde{p} \in [p]$ satisfy the QCSP, (b.) $\forall p \in [p]$, $\exists\ x$ contradicting the constraints.

How to solve a QCSP

CSC-FPS Algorithm [Djaballah et al., 2017]

Validated interval algorithm for computing barrier functions (uncontrolled case). Close to our problem.

Computable Sufficient Condition (CSC) - Feasible Point Searcher (FPS):

- FPS (branching algorithm on p): explore the set of parameters, and find a valid one (checked by CSC).
- CSC (branching algorithm on x): from a box $[p]$ checks whether (a.) a $\tilde{p} \in [p]$ satisfy the QCSP, (b.) $\forall p \in [p]$, $\exists\ x$ contradicting the constraints.

(further reductions of parameter and state space via contractors [Chabert and Jaulin, 2009])

CSC-FPS: discussion

Limitations for our application

I. CSC cannot contradict (2), *i.e.* prove that a p satisfy:

$$\exists x \in K,\ h_p(x) = 0 \wedge \forall u \in \mathcal{U},\ \dot{h}_p(x,u) \geqslant 0.$$

II. Does not consider the "quality" of the returned parameter.
III. Uncontrolled case.

CSC-FPS: discussion

Limitations for our application

I. CSC cannot contradict (2), *i.e.* prove that a p satisfy:

$$\exists x \in K,\ h_p(x) = 0 \wedge \forall u \in \mathcal{U},\ \dot{h}_p(x,u) \geqslant 0.$$

II. Does not consider the "quality" of the returned parameter.
III. Uncontrolled case.

Propositions

I. Additional validated tests to reject invalid $[p]$.
II. Optimization loop over the parameter space.
III. Sampling (under-approximation of the original QCSP).

$\Longrightarrow$ Best FPS - CSC algorithm.

What we propose for I.

I. : Finding counter-examples

In [Ishii et al., 2012], a Branch & Prune method for solving problems as negation of (2) (uncontrolled case) → Use of parametric interval Newton for finding counter-examples to (2).

What we propose for I.

I. : Finding counter-examples

In [Ishii et al., 2012], a Branch & Prune method for solving problems as negation of (2) (uncontrolled case) → Use of parametric interval Newton for finding counter-examples to (2).

Parametric interval Newton

An interval contractor $\mathcal{N}_{[y]}(.) : \mathbb{IR}^n \to \mathbb{IR}^n$, with respect to a system of equations $h(y,z) = 0, h : \mathbb{R}^{q+n} \to \mathbb{R}^n$ which satisfies

$$\forall [z] \in \mathbb{IR}^n,\ \mathcal{N}_{[y]}([z]) \subset [z] \implies \forall y \in [y],\ \exists z \in [z],\ h(y,z) = 0$$

What we propose for I.

I. : Finding counter-examples

In [Ishii et al., 2012], a Branch & Prune method for solving problems as negation of (2) (uncontrolled case) → Use of parametric interval Newton for finding counter-examples to (2).

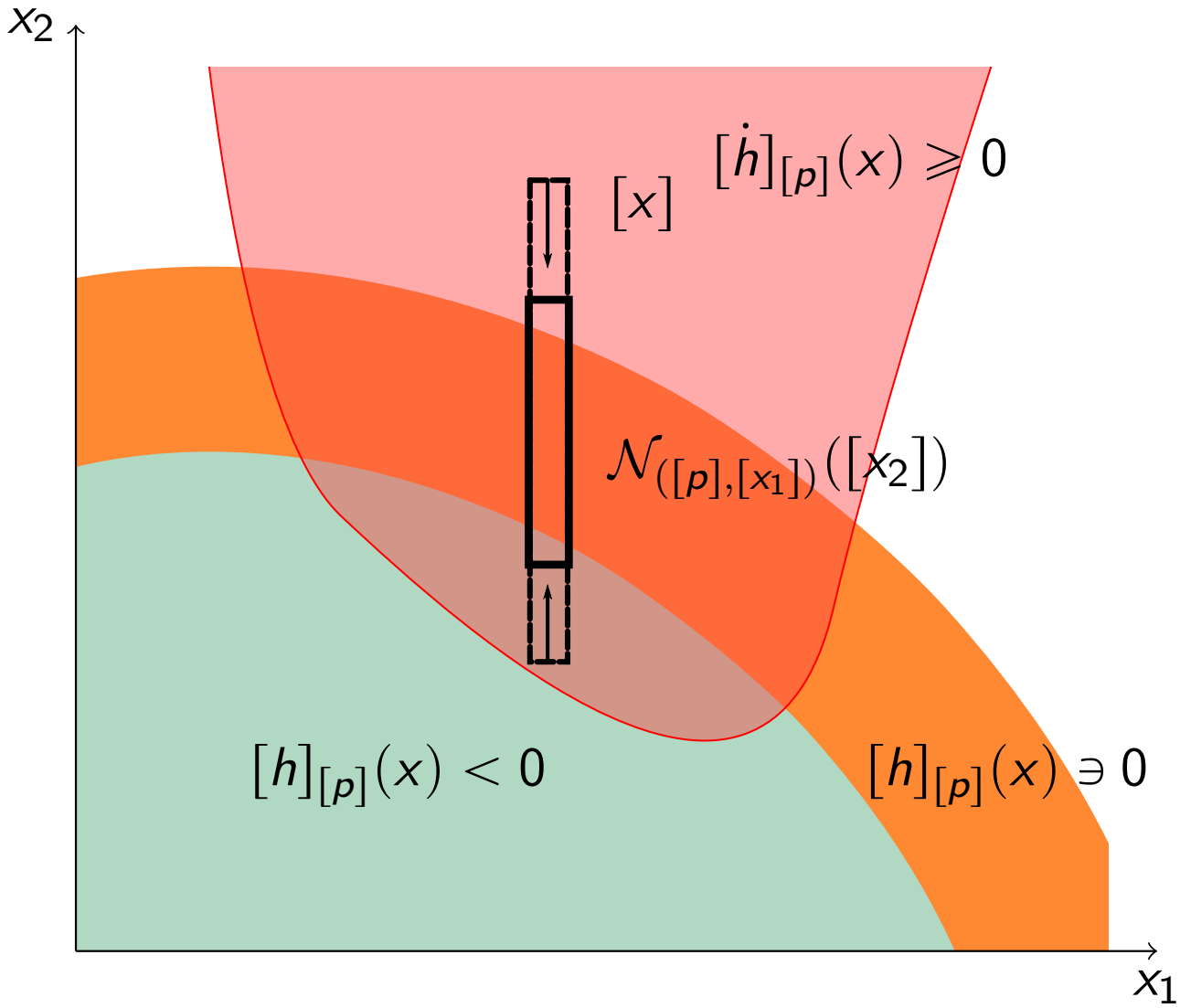
Parametric interval Newton

An interval contractor $\mathcal{N}_{[y]}(.) : \mathbb{IR}^n \to \mathbb{IR}^n$, with respect to a system of equations $h(y,z) = 0, h : \mathbb{R}^{q+n} \to \mathbb{R}^n$ which satisfies

$$\forall [z] \in \mathbb{IR}^n,\ \mathcal{N}_{[y]}([z]) \subset [z] \implies \forall y \in [y],\ \exists z \in [z],\ h(y,z) = 0$$

In our case: y corresponds to all parameters p and $n-1$ states x, and z is the remaining state.

Illustration



What we propose for II

II. : Optimizing via Branch & Bound

FPS loop is now a Branch & Bound :

- Each iteration selects a preferred box $[p]$ (from the search tree).
- Validates with CSC a point $p \in [p]$ or try to reject all $[p]$. If p is valid, cuts the branch that are worse.
- Stops when a sufficiently good point $\hat{p}$ is found.

What we propose for II

II. : Optimizing via Branch & Bound

FPS loop is now a Branch & Bound :

- Each iteration selects a preferred box $[p]$ (from the search tree).
- Validates with CSC a point $p \in [p]$ or try to reject all $[p]$. If p is valid, cuts the branch that are worse.
- Stops when a sufficiently good point $\hat{p}$ is found.

Specificities

- Definition of a "best" parameter p. Ideal: max $\mathrm{vol}\, \mathcal{H}_p$. Practice: min an approximate function $g(p)$.
- Efficient exploration strategy. Following [Neveu et al., 2016], we chose two orderings of the parameter boxes: min LB and max UB.

What we propose for III.

III. Relaxation by sampling feasible controls

Relaxation of (2): consider a finite set of control functions $\{u_1, u_2, \ldots, u_k\}$, then (2) can be relaxed to:

$$\forall x \in K \; h_p(x) \neq 0 \bigvee_{i=1}^{k} \langle \nabla_x h_p(x), f(x, u_i) \rangle < 0.$$

Choices for the u_i: constant (interval) function, state feedback, ...

Numerical results

Two phase algorithm implemented in C++

- using Ibex[2] implementing interval analysis tools for solving CSPs [Chabert and Jaulin, 2009],
- and using DynIbex [3] providing rigorous numerical integration tools using Runge-Kutta methods for the improvement phase [Alexandre dit Sandretto and Chapoutot, 2016].

[2] `http://www.ibex-lib.org/`

[3] `http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

Car on the hill

Consider the system:

$$\begin{cases} \dot{x} = v \\ \dot{v} = -9.81 \sin\left(\frac{1.1\sin(1.2x) - 1.2\sin(1.1x)}{2}\right) - 0.7v + u, \end{cases}$$

with $K = ([-1, 13] \times [-6, 6])^T$ and $u \in [-3, 3]$.

Car on the hill

Consider the system:

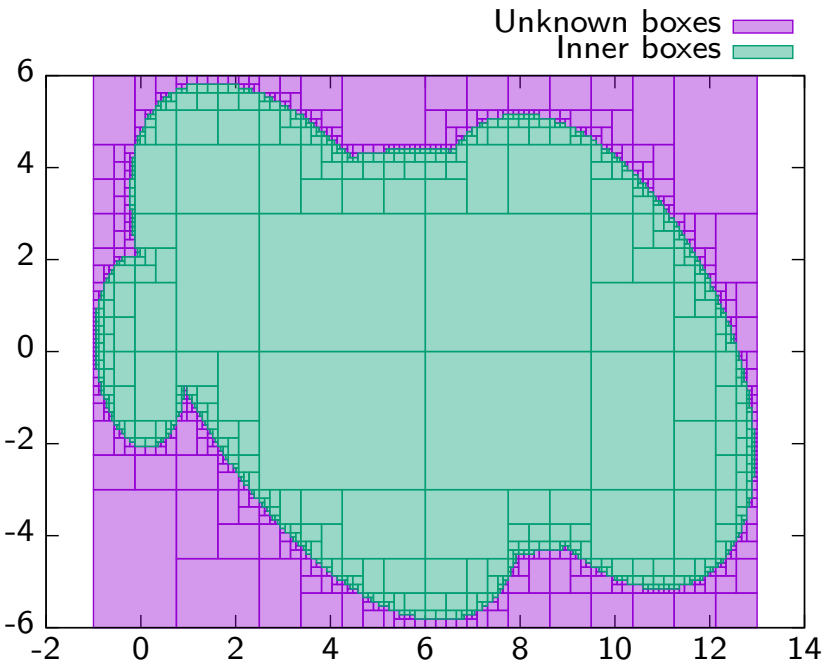
$$\begin{cases} \dot{x} = v \\ \dot{v} = -9.81 \sin\left(\frac{1.1\sin(1.2x) - 1.2\sin(1.1x)}{2}\right) - 0.7v + u, \end{cases}$$

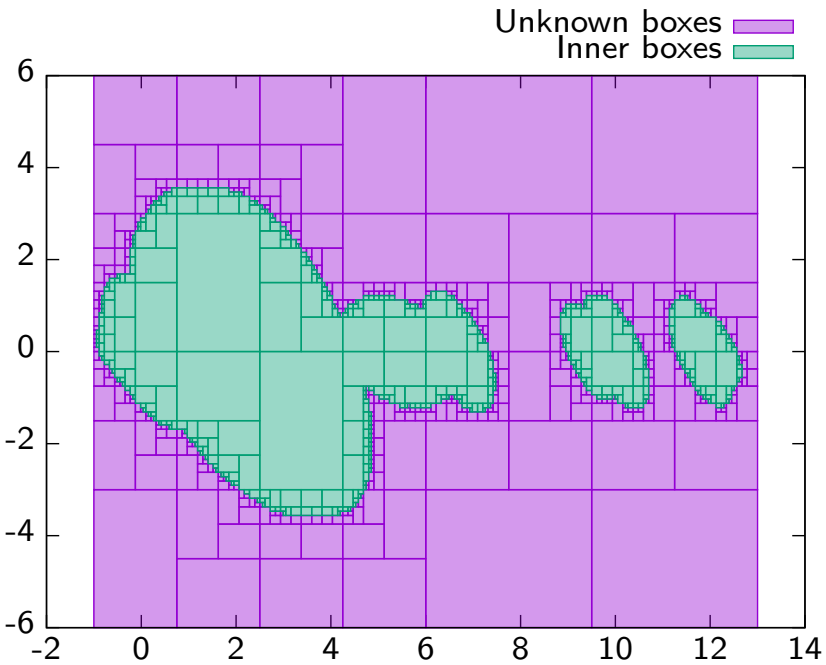
with $K = ([-1, 13] \times [-6, 6])^T$ and $u \in [-3, 3]$.
Considering equilibrium points $(x^*, 0)$ with control $u = 0$, we use the following 3-parametric template function:

$$h_p(x, v) := p_1(x - x^*)^2 + p_3 v^2 + p_2(x - x^*)v - 1.$$

Minimization of the norm of (p_1, p_3).
First phase stopped after 10 seconds for each equilibrium points.

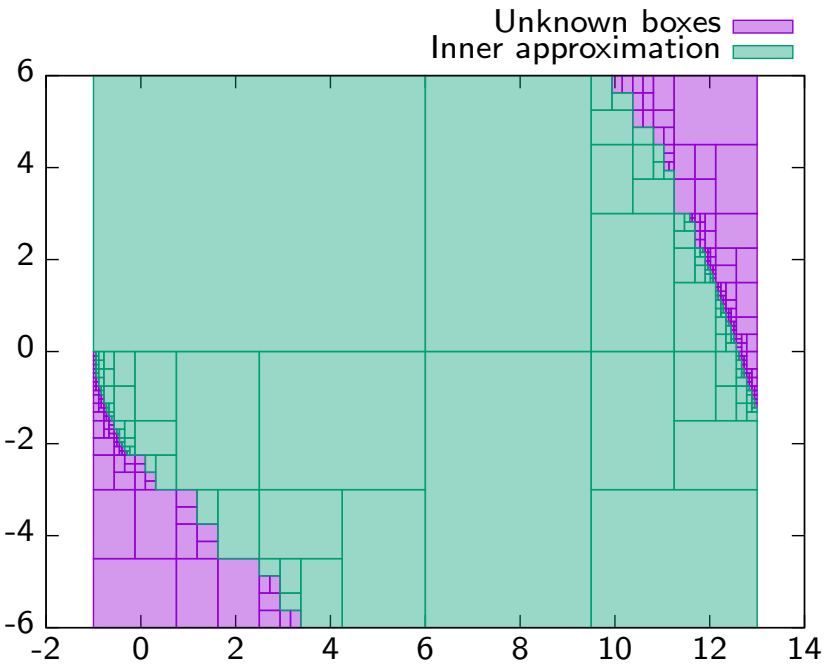
Results

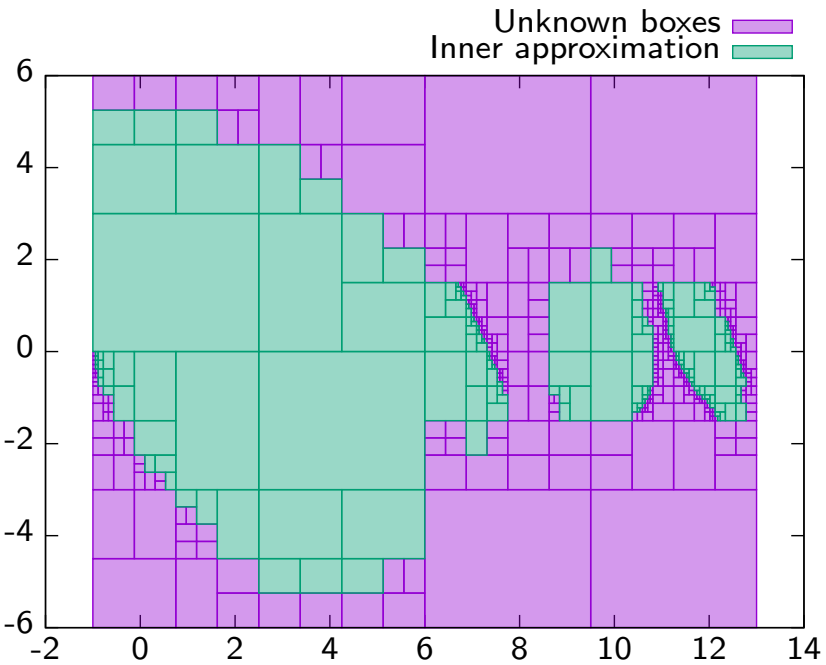




Computed initial inner-approximation of $\mathrm{Viab}_S(K)$: with our new method (left, $\sim$ 30s); with the original method (right, $\sim$ 5s).

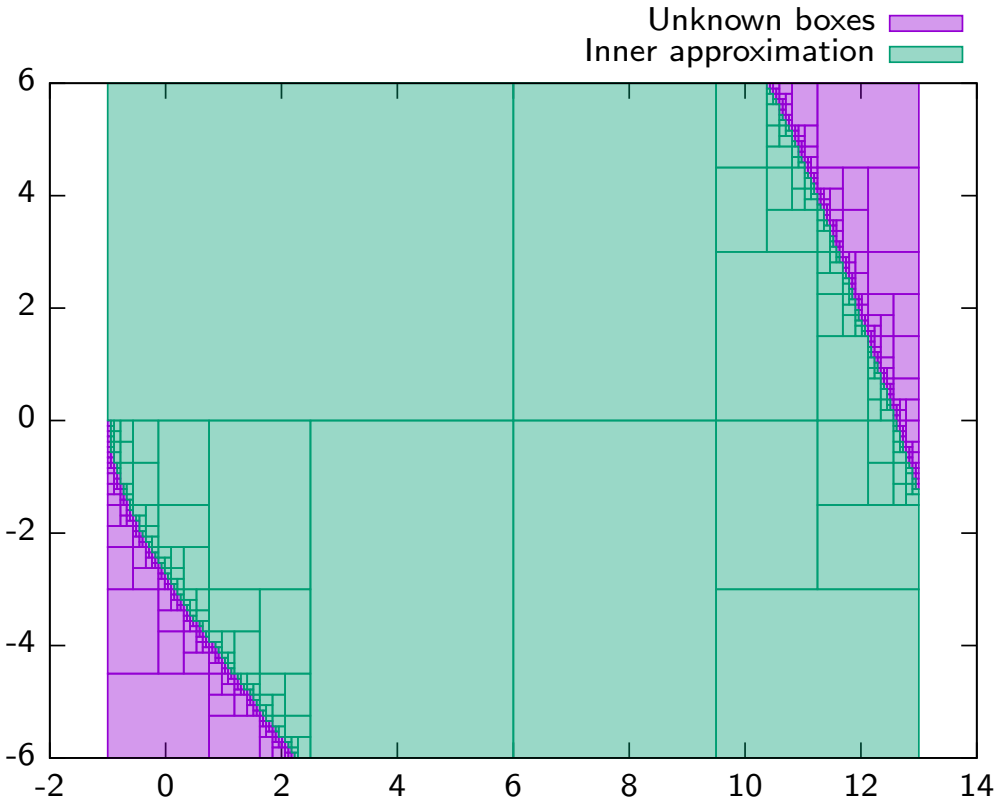
Results





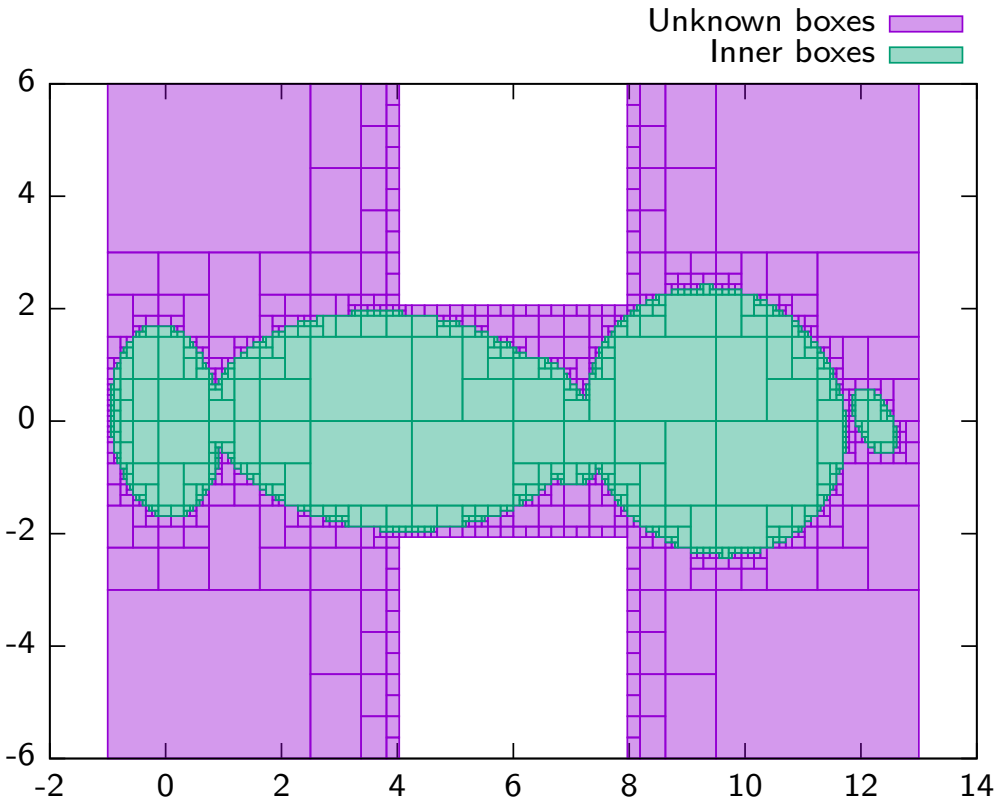
After 5 iteration of second phase: with our new method (left); with the original method (right).

Results



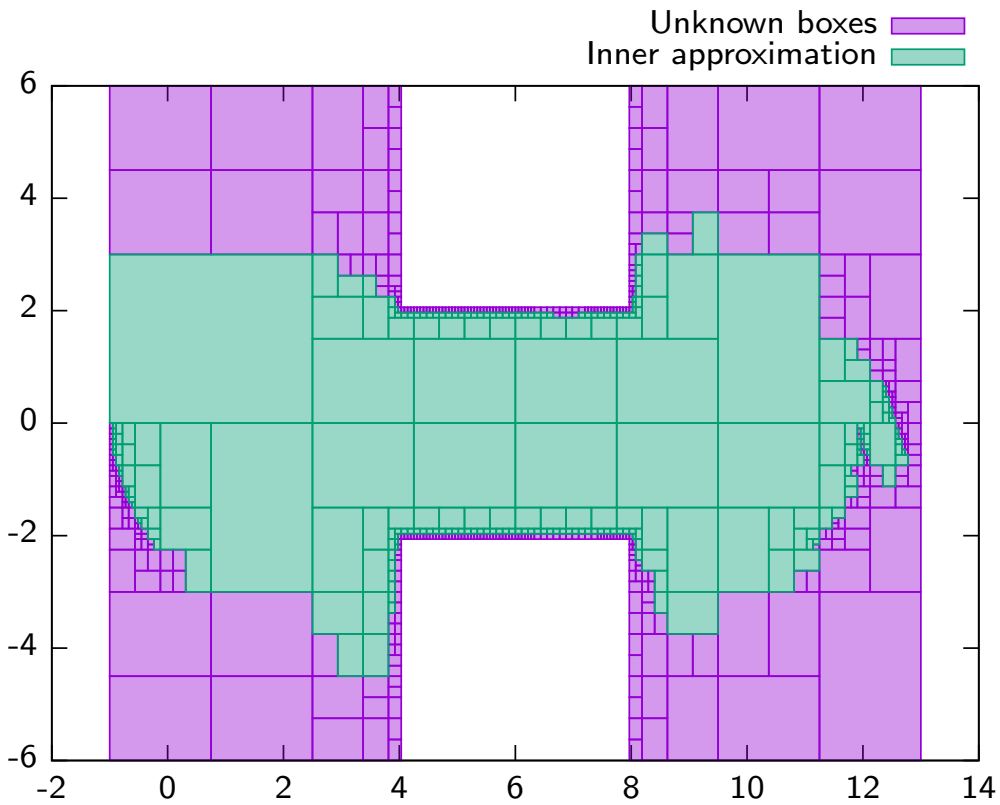
Final inner-approximation: 150 s with our new first phase, 300 s with the original one.

Results: non-convex set K



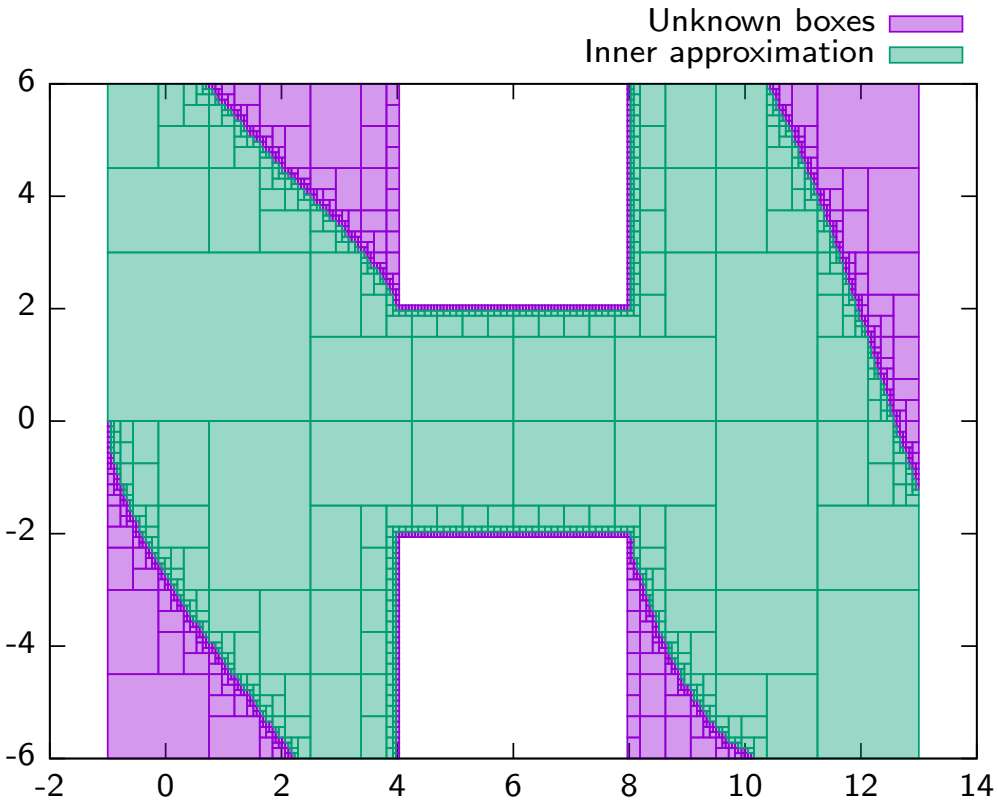
Initial inner-approximation with our new method ($\sim 50s$).

Results: non-convex set K



After 5 iterations of the second phase.

Results: non-convex set K



Final result ($\sim$ 400s).

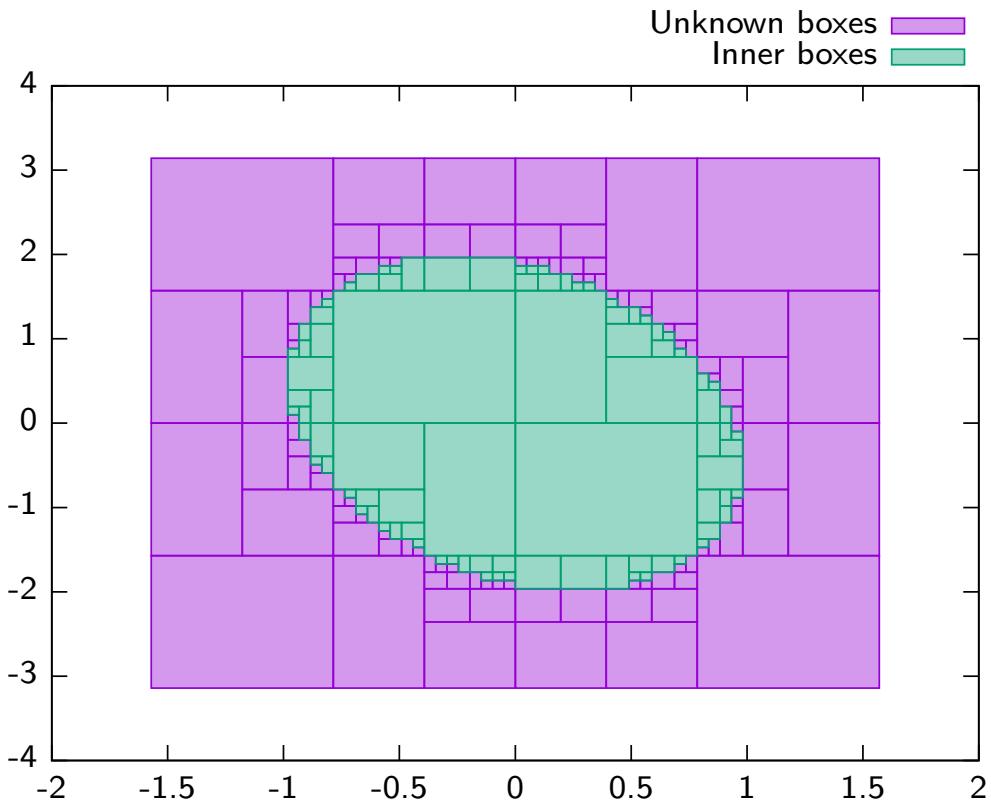
Balancing on a rod

Consider the system:

$$\begin{cases} \dot{y}_1 = y_2(t) \\ \dot{y}_2 = \cos(y_1(t))\left(u_1(t)y_1(t) + u_2(t)y_2(t)\right) + 9.81\sin(y_1(t)). \end{cases}$$

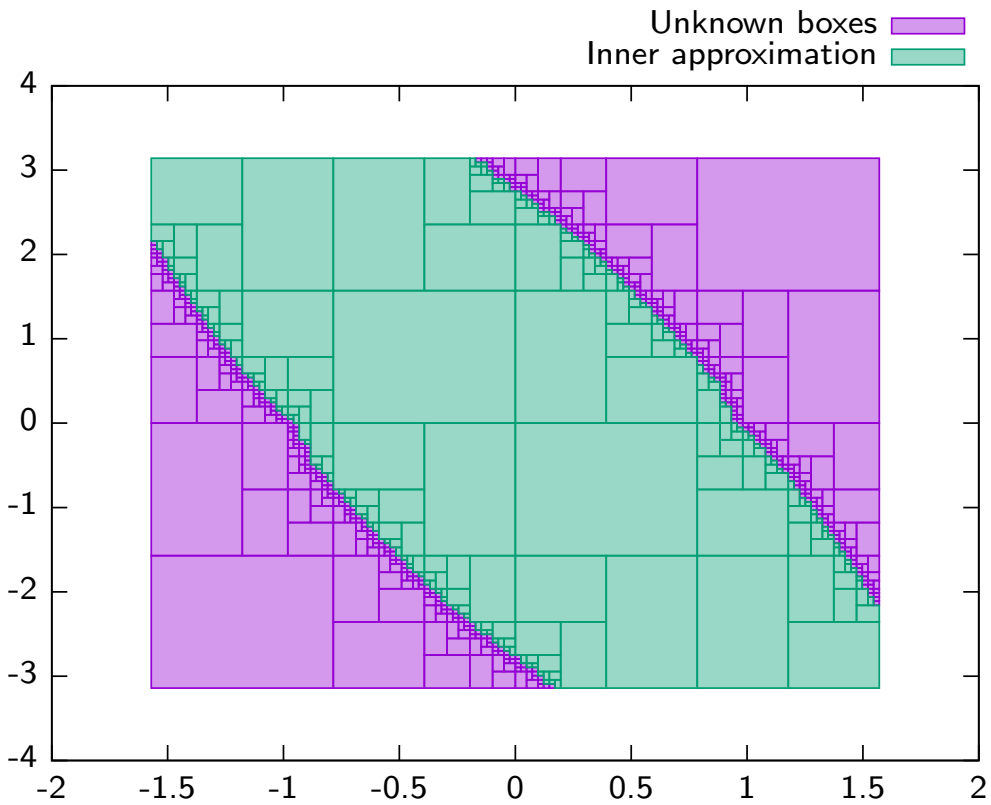
using $K = [-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\pi, \pi]$ and $\mathcal{U} = [-15, 15] \times [-2, 2]$.
Same template centered on the origin.

Balancing on a rod



Initial inner-approximation with our new method ($\sim 30s$).

Balancing on a rod



Final inner-approximation ($\sim 680s$).

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,
- higher cost, but trades favorably with the gain in the improvement phase.

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,
- higher cost, but trades favorably with the gain in the improvement phase.

Future work:

- Scalability is an issue (mainly in parameter space). Few effective contractors on p.

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,
- higher cost, but trades favorably with the gain in the improvement phase.

Future work:

- Scalability is an issue (mainly in parameter space). Few effective contractors on p.
- Identifying promising templates.

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,
- higher cost, but trades favorably with the gain in the improvement phase.

Future work:

- Scalability is an issue (mainly in parameter space). Few effective contractors on p.
- Identifying promising templates.
- Quickly finding feasible parameters (see e.g. [Hladík and Ratschan, 2014]).

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,
- higher cost, but trades favorably with the gain in the improvement phase.

Future work:

- Scalability is an issue (mainly in parameter space). Few effective contractors on p.
- Identifying promising templates.
- Quickly finding feasible parameters (see e.g. [Hladík and Ratschan, 2014]).
- (meta)Heuristic based optimization loop. Avoiding complete search in the parameter space.

Conclusion

Improving validated algorithm [Monnet et al., 2016]:

- generalization of the first-phase: solving QCSP,
- higher cost, but trades favorably with the gain in the improvement phase.

Future work:

- Scalability is an issue (mainly in parameter space). Few effective contractors on p.
- Identifying promising templates.
- Quickly finding feasible parameters (see e.g. [Hladík and Ratschan, 2014]).
- (meta)Heuristic based optimization loop. Avoiding complete search in the parameter space.
- second phase: effective contractors.

Thank you for your attention

Code available here: `http://ben-martin.fr/hscc-2018-sources/`.

References I

[Alexandre dit Sandretto and Chapoutot, 2016] Alexandre dit Sandretto, J. and Chapoutot, A. (2016).
Validated explicit and implicit runge-kutta methods.
Reliable Computing, 22:79–103.

[Chabert and Jaulin, 2009] Chabert, G. and Jaulin, L. (2009).
Contractor programming.
Artificial Intelligence, 173(11):1079 – 1100.

[Deffuant et al., 2007] Deffuant, G., Chapel, L., and Martin, S. (2007).
Approximating viability kernels with support vector machines.
IEEE Transactions on Automatic Control, 52(5):933–937.

References II

[Djaballah et al., 2017] Djaballah, A., Chapoutot, A., Kieffer, M., and Bouissou, O. (2017).
Construction of parametric barrier functions for dynamical systems using interval analysis.
Automatica, 78:287 – 296.

[Girard, 2012] Girard, A. (2012).
Controller synthesis for safety and reachability via approximate bisimulation.
Automatica, 48(5):947 – 953.

[Hladík and Ratschan, 2014] Hladík, M. and Ratschan, S. (2014).
Efficient solution of a class of quantified constraints with quantifier prefix exists-forall.
Mathematics in Computer Science, 8(3):329–340.

References III

[Ishii et al., 2012] Ishii, D., Goldsztejn, A., and Jermann, C. (2012).
Interval-based projection method for under-constrained numerical systems.
Constraints, 17(4):432–460.

[Jaulin et al., 2001] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001).
Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics.
Springer-Verlag.

References IV

[Kaynama et al., 2012] Kaynama, S., Maidens, J., Oishi, M., Mitchell, I. M., and Dumont, G. A. (2012).
Computing the viability kernel using maximal reachable sets.
In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 55–64, New York, NY, USA. ACM.

[Korda et al., 2013] Korda, M., Henrion, D., and Jones, C. N. (2013).
Convex computation of the maximum controlled invariant set for discrete-time polynomial control systems.
In *52nd IEEE Conference on Decision and Control*, pages 7107–7112.

[Monnet et al., 2016] Monnet, D., Ninin, J., and Jaulin, L. (2016).
Computing an inner and an outer approximation of the viability kernel.
Reliable Computing, 22(1):138–148.

References V

[Neumaier, 1991] Neumaier, A. (1991).
Interval Methods for Systems of Equations.
Cambridge University Press.

[Neveu et al., 2016] Neveu, B., Trombettoni, G., and Araya, I. (2016).
Node selection strategies in interval Branch and Bound algorithms.
Journal of Global Optimization, 64(2):289–304.

[Reissig et al., 2017] Reissig, G., Weber, A., and Rungger, M. (2017).
Feedback refinement relations for the synthesis of symbolic controllers.
IEEE Transactions on Automatic Control, 62(4):1781–1796.

[Saint-Pierre, 1994] Saint-Pierre, P. (1994).
Approximation of the viability kernel.
Applied Mathematics and Optimization, 29(2):187–209.

References VI

[She and Xue, 2013] She, Z. and Xue, B. (2013).
Computing an invariance kernel with target by computing lyapunov-like functions.
IET Control Theory & Applications, 7(15):1932–1940.

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;
6. else if no feasible parameter exists go to 1.;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;
6. else if no feasible parameter exists go to 1.;
7. otherwise branch on $[p]_0$, go to 1.;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;
6. else if no feasible parameter exists go to 1.;
7. otherwise branch on $[p]_0$, go to 1.;

If never stopped by 5., limit precision is reached then it is unknown, otherwise no feasible parameter in $[p]$.

CSC-FPS (2)

CSC loop consists of a branching algorithm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;

CSC-FPS (2)

CSC loop consists of a branching algorithm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;

CSC-FPS (2)

CSC loop consists of a branching algorithm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;

CSC-FPS (2)

CSC loop consists of a branching algorithm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;
4. try to find a $x \in [x]_0$ not satisfying the constraints for all $p \in [p]$, if one found returns "no feasible parameter";

CSC-FPS (2)

CSC loop consists of a branching algorithm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;
4. try to find a $x \in [x]_0$ not satisfying the constraints for all $p \in [p]$, if one found returns "no feasible parameter";
5. otherwise branch on $[x]_0$, go to 1.

CSC-FPS (2)

CSC loop consists of a branching algorithm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;
4. try to find a $x \in [x]_0$ not satisfying the constraints for all $p \in [p]$, if one found returns "no feasible parameter";
5. otherwise branch on $[x]_0$, go to 1.

If never stopped by 4., limit precision reached then it is not decidable, otherwise $\tilde{p}$ satisfies the QCSP.