Interval methods for nonlinear identification and robust control

Luc Jaulin[1], Isabelle Braems[2] and Eric Walter[2]

[1]LISA, University of Angiers, 62 avenue Notre Dame du lac, 49000 Angiers, France
jaulin@univ-angers.fr

[2] L2S, CNRS-Supelec-Université de Paris-Sud, 91192 Gif-sur-Yvette, France
{braems,walter}@lss.supelec.fr

Keywords: bounded-error estimation, constraint propagation, interval analysis, reliable methods, robust control, set computation.

Abstract: Interval methods can provide guaranteed solutions to difficult nonlinear problems, such as the global optimization of non-convex criteria or the characterization of sets defined by nonlinear inequalities. They can even deal with problems involving quantifiers, as encountered for example in robust control design. For high dimensional problems, their efficiency can be considerably improved by resorting to constraint propagation. In this paper, key ideas of interval analysis and constraint propagation are presented and applied to two problems. The first one is the guaranteed characterization of the set of all parameter vectors that are consistent with experimental data up to bounds on the acceptable errors. The second one is the design of a PI controller robustly stabilizing a set of models that may have been obtained as the solution to the first problem.

1 Introduction

The aim of this paper is to show that a relatively new approach, namely *interval constraint propagation* (ICP), makes it possible to solve some difficult nonlinear problems of automatic control efficiently. ICP combines *interval computation* [16] and *constraint propagation* [14]. The combination of these two tools has been first presented independently by Cleary [4] and Davis [5]. It will be shown that the resulting algorithms

- are very easy to understand and implement,

- lead to guaranteed results even when the function involved are nonlinear, although only numerical computations are used,
- can be very efficient even for high dimensional problems.

Interval constraint propagation will be briefly recalled in Section 2. Its application to bounded-error estimation will be illustrated on a very simple example in Section 3. A branch-and-bound algorithm to compute the projection of a set defined by inequalities is given in Section 4. This algorithm will then be used in Section 5, to characterize the set of all PI controllers that robustly stabilize a system, the uncertainty of which may have been characterized by bounded-error estimation as suggested in Section 3.

2 Interval constraint propagation

To understand the basic idea of interval constraint propagation, consider three variables x, y and z and assume that they belong to some prior feasible domains

$$\begin{array}{rcl} x & \in & [x^-, x^+] = [1, 5], \\ y & \in & [y^-, y^+] = [3, 8], \\ z & \in & [z^-, z^+] = [-1, 6]. \end{array} \tag{1}$$

Assume also that these three variables are linked by the constraint

$$\text{ADD} : z = x + y. \tag{2}$$

Remark 1 *The constraint* ADD *is usually defined in a more formal way as a subset of* $\mathbb{R}^3$*:*

$$\text{ADD} \triangleq \{(x, y, z) \in \mathbb{R}^n | z = x + y\}, \tag{3}$$

but generally written as in (2) for short. Since ADD *is a subset of* $\mathbb{R}^3$ *(or equivalently, since the addition involves three variables), we shall say that* ADD *is a ternary constraint.*

ADD can be used to contract the *prior* feasible domains for x, y and z by removing inconsistent values in the domains of x, y and z. For instance, the value -1 for z is inconsistent with ADD because

$$\forall x \in [1, 5] \text{ and } \forall y \in [3, 8],\ x + y \neq -1. \tag{4}$$

It is a very simple matter to contract the feasible domain for the variables by taking the constraint ADD into account.

- For the variable z : since $x \in [1,2]$ and $y \in [3,8]$ then $x+y \in [4,10]$. Now, since $z = x+y$ and $z \in [-1,6]$, $z \in [-1,6] \cap [4,10] = [4,6]$.
- For the variable x : since $z \in [4,6]$ and $y \in [3,8]$ then $z-y \in [4-8, 6-3] = [-4,3]$. Now, since $x = z-y$ and $x \in [1,5]$, $x \in [1,5] \cap [-4,3] = [1,3]$.
- For the variable y : since $z \in [4,6]$ and $x \in [1,3]$ then $z-x \in [4-3, 6-1] = [-1,5]$. Now, since $y = z-x$ and $y \in [3,8]$ then $y \in [3,8] \cap [-1,5] = [3,5]$.

We thus get contracted domains for x, y and z given by $[x] = [1,3]$, $[y] = [1,5]$, $[z] = [4,6]$. The constraint can now be satisfied for any instantiation of anyone of these variables inside its domain, provided that the values of the two variables are suitably chosen.

The contraction illustrated above can be described by the following procedure, where PADD stands for *Projection of the constraint* ADD.

Algorithm PADD(inout: $[z],[x],[y]$)
1 $[z] := [z] \cap ([x] + [y])$;
2 $[x] := [x] \cap ([z] - [y])$;
3 $[y] := [y] \cap ([z] - [x])$.

(5)

In this procedure, the addition and the difference of two intervals are defined by:

$$\begin{array}{rcl} [x^-, x^+] + [y^-, y^+] & = & [x^- + y^-, x^+ + y^+], \\ [x^-, x^+] - [y^-, y^+] & = & [x^- - y^+, x^+ - y^-]. \end{array} \tag{6}$$

PADD is often called a *projection procedure* [3] because it computes the projections onto the x, y and z axis of the subset of $\mathbb{R}^3$ given by

$$\text{ADD} \cap ([x] \times [y] \times [z]) = \{(x,y,z) \in [x] \times [y] \times [z] | z = x+y\} . \tag{7}$$

The projection procedure developed for ADD can be extended to other ternary constraints such as MULT: $z = x * y$, or equivalently

$$\text{MULT} \triangleq \{(x,y,z) \in \mathbb{R}^n | z = x * y\} . \tag{8}$$

The resulting projection procedure becomes

Algorithm PMULT(inout: $[z], [x], [y]$)
1 $[z] := [z] \cap ([x] * [y])$;
2 $[x] := [x] \cap ([z] * 1/[y])$;
3 $[y] := [y] \cap ([z] * 1/[x])$.

(9)

In this procedure, the multiplication and the inversion of intervals are defined by

$$\begin{array}{lcl} [x^-, x^+] * [y^-, y^+] & = & [\min(x^-y^-, x^-y^+, x^+y^-, x^+y^+), \\ & & \max(x^-y^-, x^-y^+, x^+y^-, x^+y^+)], \\ \frac{1}{[x^-, x^+]} & = & \mathbb{R} \text{ if } 0 \in [x^-, x^+] \text{ and } [\frac{1}{x^+}, \frac{1}{x^-}] \text{ otherwise.} \end{array} \tag{10}$$

Elementary functions can be interpreted as binary constraints and receive a similar treatment. Consider for instance, the binary constraint

$$\text{EXP} \triangleq \{(x, y) \in \mathbb{R}^n | y = \exp(x)\} . \tag{11}$$

The associated projection procedures becomes

Algorithm PEXP(inout: $[y], [x]$)
1 $[y] := [y] \cap \exp([x])$;
2 $[x] := [x] \cap \log([x])$.

In this procedure,

$$\begin{array}{lcl} \exp([x^-, x^+]) & = & [\exp(x^-), \exp(x^+)], \\ \log([x^-, x^+]) & = & \emptyset \text{ if } x^+ \leq 0, \\ & = & [\log(x^-), \log(x^+)] \text{ if } x^- > 0, \\ & = &]-\infty, \log(x^+)] \text{ if } 0 \in [x^-, x^+]. \end{array} \tag{12}$$

Projection procedures can be developed for any ternary constraint such as ADD: $z = x+y$, MULT: $z = x * y$ or binary constraint such as EXP: $y = \exp(x)$. Any constraint for which such a projection procedure is available will be called a *primitive constraint*.

Consider n variables $x_1, \ldots, x_n$ linked by m primitive constraints $\mathcal{C}_1, \ldots, \mathcal{C}_m$. For each variable x_i, it is assumed that a prior feasible domain $[x_i] = [x_i^-, x_i^+]$ is available (this domain may be equal to $]-\infty, \infty[$ if no information is available on x_i). The principle of interval constraint propagation (ICP) is to perform a projection of all constraints.

This operation is repeated until no more significant contraction can be performed. The following recursive algorithm performs this task.

Algorithm ICP(inout: $[x_1], \ldots, [x_n]$)
1 $[a_1] := [x_1], \ldots, [a_n] := [x_n]$;
2 for $j = 1$ to m, project$(\mathcal{C}_j)$;
3 if $\max_i \left(a_i^+ - x_i^+ + x_i^- - a_i^-\right) \geq \varepsilon$ then ICP$([x_1], \ldots, [x_n])$.

Step 1, stores the initial domains for the x_i's for a later evaluation of the performance achieved by the contractions performed at Step 2. At Step 2, all constraints are projected in order to contract the current domains. At Step 3, the algorithm recursively calls itself if the contractions are still deemed efficient, with ε a positive stopping parameter to be tuned by the user.

Theorem 1 *The algorithm ICP stops in less than* $\frac{mn}{\varepsilon} w([\mathbf{x}])$ *projections, where* $[\mathbf{x}]$ *is the box* $[x_1] \times \cdots \times [x_n]$ *and* $w([\mathbf{x}])$ *is the width of the box* $[\mathbf{x}]$*, i.e.,*

$$w([\mathbf{x}]) \triangleq \max_{i \in \{1,\ldots,n\}} \left(x_i^+ - x_i^-\right). \tag{13}$$

Proof : Define ℓ_k as the value of $\left(x_1^+ - x_1^-\right) + \cdots + (x_n^+ - x_n^-)$ at the kth iteration of ICP. While the algorithm iterates, $\ell_k \leq \ell_{k-1} - \varepsilon$. Thus, $\forall k$, $\ell_k \leq \ell_0 - k\varepsilon$. Now, $\ell_0 - k\varepsilon$ becomes negative when $\frac{\ell_0}{\varepsilon} < k$. Therefore, since ICP terminates for a positive ℓ_k, it cannot take more than ℓ_0/ε iterations. Since $\ell_0 < nw([\mathbf{x}])$, and since at each iteration m projections are performed, no more than $\frac{mn}{\varepsilon} w([\mathbf{x}])$ projections are performed. ■

Remark 2 *For simplicity, a very simple version of ICP has been presented here. Its efficiency could be improved, for instance, by choosing a clever order for the projections at Step 2 [2].*

3 Application to bounded-error estimation

Consider the electronic circuit consisting of one battery and two resistors represented in Figure 1. Assume that measures have been collected on this circuit, leading to the following relations :

$$\begin{aligned} E &\in [23V, 26V],\ I \in [4A, 8A],\ U_1 \in [10V, 11V], \\ U_2 &\in [14V, 17V],\ P \in [124W, 130W], \end{aligned} \tag{14}$$

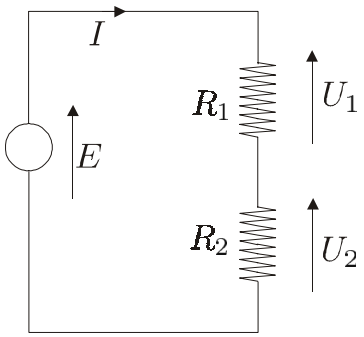

Figure 1: A simple circuit used to illustrate bounded-error estimation

where P is the power delivered by the battery. Nothing is known about the values of the resistors except that they are positive. Thus the prior domains for R_1 and R_2 are $]0, \infty[$. These quantities are related by the following constraints:

$$P = EI;\ E = (R_1 + R_2)\, I;\ U_1 = R_1 I;\ U_2 = R_2 I;\ E = U_1 + U_2.$$

Some of these constraints are redundant, but detecting this is not required by the method. To the contrary, redundancy makes constraint propagation more efficient. Since the second constraint is not primitive, it is decomposed by introducing an auxiliary variable, say R, as follows:

$$E = (R_1 + R_2)\, I \text{ is decomposed into } (E = RI;\ R = R_1 + R_2)\,.$$

Constraint propagation can then be performed by executing several times, say q, the following projections:

$$\begin{array}{l} \text{PMULT}\,(P, E, I)\,;\text{PADD}\,(R, R_1, R_2)\,;\text{PMULT}\,(E, R, U)\,; \\ \text{PMULT}\,(U_1, R_1, I)\,;\text{PMULT}\,(U_2, R_2, I)\,;\text{PADD}\,(E, U_1, U_2)\,; \end{array}$$

It can be shown (see page 92 of [8]) that if q tends to infinity the contracted domains that are finally obtained do not depend on the ordering of the projections.

To show how easy the implementation is and to motivate the reader to try his own example in few minutes, a short program is given in Table 1. It is also available at : *http://www.istia.univ-angers.fr/~jaulin/propagR1R2.sce* and runs under the Scilab language [1], freely available at *http://www-rocq.inria.fr/scilab/*. This stand-alone program computes intervals containing R_1 and R_2 by ICP.

// **interval arithmetic**	
function z=inter(x,y);	z=[max(x(1),y(1)),min(x(2),y(2))];
	endfunction
function z=add(x,y);	z=x+y; endfunction
function z=dif(x,y);	z=[x(1)-y(2),x(2)-y(1)]; endfunction
function z=mult(x,y);	v=[x(1)*y,x(2)*y]; z=[min(v),max(v)];
	endfunction
function y=invert(x);	y=[-%inf,%inf];
	if prod(x)>0, y=[1/x(2),1/x(1)];
	endfunction
function z=div(x,y);	a=invert(y); z=mult(x,a);
	endfunction
// **projection operators**	
function [c,a,b]=padd(z,x,y);	c=inter(z,add(x,y)); a=inter(x,dif(z,y));
	b=inter(y,dif(z,x)); endfunction
function [c,a,b]=pmult(z,x,y)	c=inter(z,mult(x,y)); a=inter(x,div(z,y));
	b=inter(y,div(z,x)); endfunction
// **Main program**	
R=[-%inf,%inf]; R1=[0,%inf]; R2=[0,%inf];	
E=[23,26]; I=[4,8]; U1=[10,11]; U2=[14,17]; P=[124,130];	
for i=1:10,	
[R,R1,R2]=padd(R,R1,R2); [P,E,I]=pmult(P,E,I);	
[E,R,I]=pmult(E,R,I); [U2,R2,I]=pmult(U2,R2,I);	
[U1,R1,I]=pmult(U1,R1,I); [E,U1,U2]=padd(E,U1,U2);	
end;	

Table 1 : A SCILAB program using ICP for estimation

In less that 0.1 second on a Pentium 300, this program generates the following results:

$$\begin{aligned} R_1 &\in [1.846; 2.307]\,,\ R_2 \in [2.584; 3.355],\ I \in [4.769; 5.417]\,; \\ U_1 &\in [10; 11]\,,\ U_2 \in [14; 16]\,,\ E \in [24; 26]\,,\ P \in [124; 130]\,. \end{aligned}$$

ICP made it possible to get rather accurate intervals containing R_1 and R_2. The domains for I and U_2 have also been contracted, whereas the domains for U_1 and P have been left unchanged.

This example is of course very simple and was only meant to facilitate understanding of the concept of ICP. The same type of techniques can be employed for the estimation of the parameters and/or state of a nonlinear dynamical system, see for instance [7] and [8]. To employ interval techniques in the context of joint identification and robust control, it becomes necessary to distinguish the uncertain parameter vector $\mathbf{p}$ of the model of the process from the tuning parameter vector $\mathbf{c}$ of the controller to be connected with this process. This will be made possible by the technique presented in Section 4, which will be applied in Section 5 to robust stabilization.

4 A projection algorithm

Consider the set

$$\mathbb{S} \triangleq \{(\mathbf{c}, \mathbf{p}) \in [\mathbf{c}] \times [\mathbf{p}] \mid f(\mathbf{c}, \mathbf{p}) \leq 0\}, \tag{15}$$

where $[\mathbf{c}]$ is a box of $\mathbb{R}^{n_\mathrm{c}}$, $[\mathbf{p}]$ is a box of $\mathbb{R}^{n_\mathrm{p}}$ and f is a function mapping $\mathbb{R}^{n_\mathrm{c}} \times \mathbb{R}^{n_\mathrm{p}}$ into $\mathbb{R}$. The projection $\mathbb{S}_\mathrm{c}$ of $\mathbb{S}$ onto $\mathbb{R}^{n_\mathrm{c}}$ is defined by

$$\mathbb{S}_\mathrm{c} \triangleq \{\mathbf{c} \in [\mathbf{c}] \mid \exists \mathbf{p} \in [\mathbf{p}], f(\mathbf{c}, \mathbf{p}) \leq 0\}. \tag{16}$$

This section is devoted to an algorithm computing inner and outer approximations of $\mathbb{S}_\mathrm{c}$. To simplify the presentation, we shall first introduce the notion of contractors.

4.1 Contractors for sets

A *box*, or *interval vector,* $[\mathbf{x}]$ of $\mathbb{R}^n$ is a vector with interval components:

$$[\mathbf{x}] = [x_1^-, x_1^+] \times \cdots \times [x_n^-, x_n^+] = [x_1] \times \cdots \times [x_n]. \tag{17}$$

The set of all boxes of $\mathbb{R}^n$ will be denoted by $\mathbb{IR}^n$. The operator $\mathcal{C}_\mathbb{S} : \mathbb{IR}^n \to \mathbb{IR}^n$ is a *contractor* for the set $\mathbb{S}$ of $\mathbb{R}^n$ if it satisfies

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \begin{cases} \mathcal{C}_\mathbb{S}([\mathbf{x}]) \subset [\mathbf{x}] & \text{(contractance)}, \\ \mathcal{C}_\mathbb{S}([\mathbf{x}]) \cap \mathbb{S} = [\mathbf{x}] \cap \mathbb{S} & \text{(correctness)}. \end{cases} \tag{18}$$

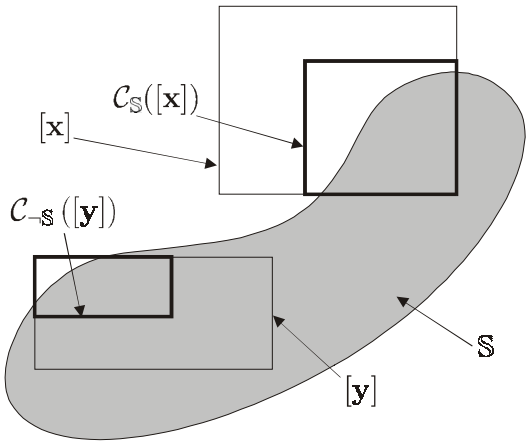

Figure 2: Action of contractors for $\mathbb{S}$ and $\neg\mathbb{S}$

Figure 2 illustrates the action of a contractor for $\mathbb{S}$ operating on $[\mathbf{x}]$ and a contractor for its complementary set $\neg\mathbb{S}$ operating on $[\mathbf{y}]$. ICP can be used to build contractors. Consider for instance the set

$$\mathbb{S} \triangleq \{\mathbf{x} \in \mathbb{R}^n | x_1 + x_1 x_2 \leq 1 \text{ and } \sin(x_1 + x_2) \leq 0\} \tag{19}$$

or equivalently

$$\mathbb{S} \triangleq \{\mathbf{x} \in \mathbb{R}^n | \max(x_1 + x_1 x_2 - 1, \sin(x_1 + x_2)) \leq 0\}. \tag{20}$$

The constraint defining $\mathbb{S}$ in (20) can be decomposed into a set of primitive constraints, via the introduction of auxiliary variables

$$a = x_1 x_2,\ b = x_1 + a,\ c = b - 1,\ d = x_1 + x_2,\ e = \sin(d) \text{ and } f = \max(c, e). \tag{21}$$

The prior domains of these auxiliary variables are

$$a \in\ ]-\infty, \infty[,\ b \in]-\infty, \infty[,\ c \in]-\infty, \infty[, \tag{22}$$

$$d \in\ ]-\infty, \infty[,\ e \in]-\infty, \infty[\text{ and } f \in]-\infty, 0]. \tag{23}$$

If ICP is applied to this set of constraints, the domains for the variables will be contracted. A possible contractor for $\mathbb{S}$ is thus given by the following procedure.

Algorithm $\mathcal{C}_{\mathbb{S}}$ (inout: $[x_1], [x_2]$)	
1	$[a] := [b] := [c] := [d] := [e] :=]-\infty, \infty[$; $[f] :=]-\infty, 0[$;
2	while contraction is significant,
3	PMULT$([a], [x_1], [x_2])$; PADD$([b], [x_1], [a])$;
4	PADD $([b], [c], [1, 1])$; PADD $([d], [x_1], [x_2])$;
5	PSIN $([e], [d])$; PMAX $([f], [c], [e])$;
6	end while.

(24)

A similar approach can be applied to a large class of sets defined by nonlinear inequalities. If these inequalities are be linked by Boolean operators, *min* and *max* constraints can be used to obtain expressions that are free of such operators. For instance, the complementary set $\neg\mathbb{S}$ of $\mathbb{S}$ is given by

$$\neg\mathbb{S} = \{\mathbf{x} \in \mathbb{R}^n | x_1 + x_1 x_2 > 1 \text{ or } \sin(x_1 + x_2) > 0\} \quad (25)$$

$$= \{\mathbf{x} \in \mathbb{R}^n | \max(x_1 + x_1 x_2 - 1, \sin(x_1 + x_2)) > 0\}. \quad (26)$$

A contractor for $\neg\mathbb{S}$ is thus given by the following procedure.

Algorithm $\mathcal{C}_{\neg\mathbb{S}}$ (inout: $[x_1], [x_2]$)	
1	$[a] := [b] := [c] := [d] := [e] :=]-\infty, \infty[$; $[f] :=]0, \infty[$;
2	while contraction is significant,
3	PMULT$([a], [x_1], [x_2])$; PADD$([b], [x_1], [a])$;
4	PADD $([b], [c], [1, 1])$; PADD $([d], [x_1], [x_2])$;
5	PSIN $([e], [d])$; PMAX $([f], [c], [e])$;
6	end while.

(27)

4.2 Projection algorithm

The algorithm PROJECT presented in Table 2 computes two subpavings (*i.e.*, unions of non-overlapping boxes) $\mathcal{L}^-$ and $\Delta\mathcal{L}$ such that the set $\mathbb{S}_c$ defined by (16) satisfies

$$\mathcal{L}^- \subset \mathbb{S}_c \subset \mathcal{L}^- \cup \Delta\mathcal{L}. \quad (28)$$

This algorithm is not trivial to understand without some experience on branch-and-prune algorithms. Unfamiliar readers may consult Chapter 5 of [8], which contains many such algorithms with an increasing level of difficulty.

In Table 2 $w([\mathbf{x}])$ is the width of the box $[\mathbf{x}]$, i.e., the length of its largest side(s). To *bisect* $[\mathbf{c}]$ means to split it into two boxes along a symmetry plane orthogonal to a side of maximum length. The principle PROJECT is to update three lists : (i) the stack $\mathcal{L}$ containing boxes of $[\mathbf{c}] \times [\mathbf{p}]$ still to be studied, (ii) the list $\mathcal{L}^-$ containing boxes of $\mathbb{R}^{n_c}$ that have been proved to be inside $\mathbb{S}_c$ and (iii) the list $\Delta\mathcal{L}$ containing boxes of $\mathbb{R}^{n_c}$ for which nothing is known but which are deemed too small to be bisected further (because their width is smaller than ε). The property

$$\mathbb{S}_c \subset \ \text{proj}_c\left(\mathcal{L}\right) \cup \mathcal{L}^- \cup \Delta\mathcal{L}.$$

is satisfied whenever the algorithm passes through Step 2. When PROJECT terminates $\mathcal{L}$ is empty and thus (28) is satisfied. The procedure OUTSIDE, called at Step 4 and given in Table 3, is a contractor for the set $\mathbb{S}$ to be projected. The procedure INSIDE called at Step 6 and given in Table 4, returns a subbox $\overline{[\mathbf{c}]}$ of $[\mathbf{c}]$ such that $[\mathbf{c}] \cap \neg\mathbb{S}_c = \overline{[\mathbf{c}]} \cap \neg\mathbb{S}_c$. Note that, contrary to OUTSIDE, INSIDE does not modify $[\mathbf{c}]$ and $[\mathbf{p}]$.

Algorithm PROJECT(in: $[\mathbf{c}], [\mathbf{p}]$; out: $\mathcal{L}^-, \Delta\mathcal{L}$)	
1	$\mathcal{L} := \{([\mathbf{c}], [\mathbf{p}])\}\,; \mathcal{L}^- := \emptyset; \Delta\mathcal{L} := \emptyset;$
2	while $\mathcal{L} \neq \emptyset$,
3	pop a pair of boxes $([\mathbf{c}], [\mathbf{p}])$ out of $\mathcal{L}$;
4	OUTSIDE$([\mathbf{c}], [\mathbf{p}])\,;$
5	if $[\mathbf{c}] = \emptyset$, go to 2;
6	$\overline{[\mathbf{c}]}$:= INSIDE$([\mathbf{c}], [\mathbf{p}])\,;$
7	if $\overline{[\mathbf{c}]} = \emptyset$ then $\mathcal{L}^- = \mathcal{L}^- \cup \{[\mathbf{c}]\}$; go to 2;
8	if $w\left([\mathbf{c}]\right) > \varepsilon$,
9	bisect $[\mathbf{c}]$ into $[\mathbf{c}_1]$ and $[\mathbf{c}_2]$;
10	$\mathcal{L} := \mathcal{L} \cup \{([\mathbf{c}_1], [\mathbf{p}])\,, ([\mathbf{c}_2], [\mathbf{p}])\}$;
11	else, $\Delta\mathcal{L} := \Delta\mathcal{L} \cup \{[\mathbf{c}]\}$;
12	end while.

Table 2 : Algorithm providing an enclosure of the projection of $\mathbb{S}$

In Table 3, $\mathcal{L}$ is a last-in-first-out list of non-overlapping boxes of $\mathbb{R}^{n_c} \times \mathbb{R}^{n_p}$. Check that whenever the algorithm reaches Step 2,

$$\left([\mathbf{c}_0^{\text{init}}] \times [\mathbf{p}_0^{\text{init}}]\right) \cap \mathbb{S} = \left(\mathcal{L} \cup \left([\mathbf{c}_0] \times [\mathbf{p}_0]\right)\right) \cap \mathbb{S},$$

where $[\mathbf{c}_0^{\text{init}}]$ and $[\mathbf{p}_0^{\text{init}}]$ denote the initial boxes sent as input arguments of OUTSIDE. When OUTSIDE terminates, $\mathcal{L}$ is empty and thus the correctness property $\left([\mathbf{c}_0^{\text{init}}] \times [\mathbf{p}_0^{\text{init}}]\right) \cap \mathbb{S} = \left([\mathbf{c}_0] \times [\mathbf{p}_0]\right) \cap \mathbb{S}$, is satisfied, as required by (18). Note that $[\mathbf{c}]$ is never bisected in OUTSIDE.

Algorithm OUTSIDE(inout: $[\mathbf{c}_0], [\mathbf{p}_0]$)	
1	$\mathcal{L} := \{([\mathbf{c}_0], [\mathbf{p}_0])\}\,; \varepsilon_{\mathrm{p}} = w\,([\mathbf{c}_0])\,; [\mathbf{c}_0] := \emptyset; [\mathbf{p}_0] := \emptyset;$
2	while $\mathcal{L} \neq \emptyset$,
3	take a pair of boxes $([\mathbf{c}], [\mathbf{p}])$ in $\mathcal{L}$;
4	$\mathcal{C}_{\mathbb{S}}\,([\mathbf{c}], [\mathbf{p}])\,;$
5	if $w\,([\mathbf{p}]) > \varepsilon_{\mathrm{p}}$,
6	bisect $[\mathbf{p}]$ into $[\mathbf{p}_1]$ and $[\mathbf{p}_2]$;
7	$\mathcal{L} := \mathcal{L} \cup \{([\mathbf{c}], [\mathbf{p}_1])\,, ([\mathbf{c}], [\mathbf{p}_2])\}$;
8	else $[\mathbf{c}_0] := [\mathbf{c}_0] \sqcup [\mathbf{c}]$; $[\mathbf{p}_0] := [\mathbf{p}_0] \sqcup [\mathbf{p}]$;
9	end while.

Table 3 : Contractor for $\mathbb{S}$

INSIDE returns a subbox $\overline{[\mathbf{c}]}$ of $[\mathbf{c}]$ such that $[\mathbf{c}] \cap \neg\mathbb{S}_{\mathrm{c}} = \overline{[\mathbf{c}]} \cap \neg\mathbb{S}_{\mathrm{c}}$, *i.e.*, it implements a contractor for the complementary set $\neg\mathbb{S}_{\mathrm{c}}$ of $\mathbb{S}_{\mathrm{c}}$. The values of $\overline{[\mathbf{c}]}$ that have been removed inside the *repeat* loop of Step 4 all belong to $\mathbb{S}_{\mathrm{c}}$. This is illustrated by Figure 3. In this algorithm, only $[\mathbf{p}]$ is bisected. All intermediate boxes are stored inside $\mathcal{L}$, which is here a first-in-first-out list.

Algorithm INSIDE$\left(\text{in: } [\mathbf{c}_0], [\mathbf{p}_0]\text{;out } \overline{[\mathbf{c}]}\right)$	
1	$\mathcal{L} := \{[\mathbf{p}_0]\}\,; \overline{[\mathbf{c}]} := [\mathbf{c}_0];$
2	while $\mathcal{L} \neq \emptyset$,
3	take the first box $[\mathbf{p}]$ in $\mathcal{L}$;
4	repeat,
5	$\mathbf{m} := \mathrm{center}\,([\mathbf{p}])\,;\ \mathcal{C}_{\neg\mathbb{S}}\left(\overline{[\mathbf{c}]}, \mathbf{m}\right)\,;$
6	$[\mathbf{c}_1] := \overline{[\mathbf{c}]}; \mathcal{C}_{\mathbb{S}}\,([\mathbf{c}_1], [\mathbf{p}])\,;$
7	while a significant contraction of $\overline{[\mathbf{c}]}$ is performed;
8	if $\overline{[\mathbf{c}]} = \emptyset$, return;
9	if $w\,([\mathbf{p}]) > w\,([\mathbf{c}_0])$,
10	bisect $[\mathbf{p}]$ into $[\mathbf{p}_1]$ and $[\mathbf{p}_2]$;
11	store $[\mathbf{p}_1]$ and $[\mathbf{p}_2]$ at the end of $\mathcal{L}$;
12	end while.

Table 4 : Contractor for $\neg\mathbb{S}_{\mathrm{c}}$

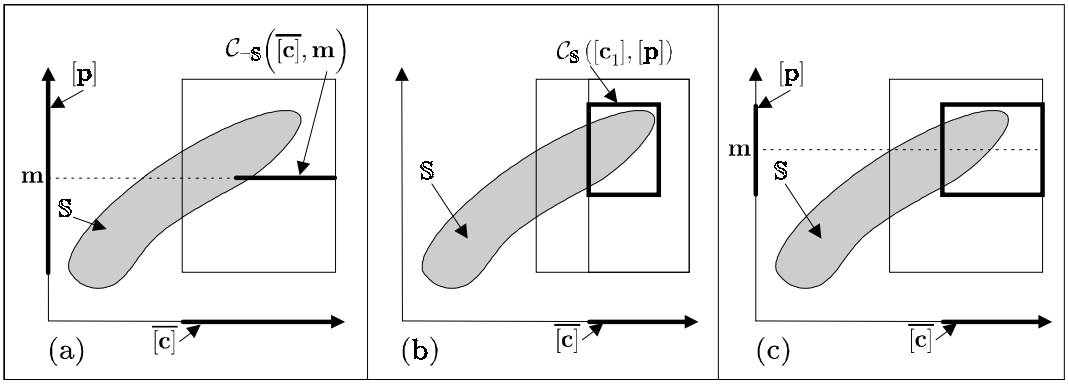

Figure 3: Principle of INSIDE; (a) $\overline{[\mathbf{c}]}$ is contracted without losing a single point of $\neg\mathbb{S}_{\mathrm{c}}$; (b) $[\mathbf{p}]$ is now contracted, but $\overline{[\mathbf{c}]}$ remains unchanged; (c) a new iteration can be started

5 Application to robust control

Although relatively few papers have been dedicated to controller design via interval analysis see, *e.g.*, [12, 10, 11, 13, 9, 15], interest is growing, as illustrated by a recent special issue of *Reliable Computing* [6]. In this section, the projection algorithm presented in Section 4 will be used to characterize the set of all parameter vectors $\mathbf{c}$ of a parametric linear controller $\Gamma(\mathbf{c})$ that robustly stabilize an uncertain linear time-invariant model of a process. The parameter vector $\mathbf{c}$ of the controller can be chosen arbitrarily in a box $[\mathbf{c}] \subset \mathbb{R}^{n_{\mathrm{c}}}$. The uncertain process is assumed to be representable by a parametric model $\Sigma(\mathbf{p})$, the parameter vector $\mathbf{p}$ of which belongs to $[\mathbf{p}] \subset \mathbb{R}^{n_{\mathrm{p}}}$. The box $[\mathbf{p}]$ may have been obtained by bounded-error estimation techniques similar to that described in Section 3. Denote by $\Sigma(\mathbf{c},\mathbf{p})$ the closed-loop system. It is easy to find, for instance via the Routh criterion, a function r: $\mathbb{R}^{n_{\mathrm{c}}} \times \mathbb{R}^{n_{\mathrm{p}}} \rightarrow \mathbb{R}$ such that

$$\Sigma(\mathbf{p},\mathbf{c}) \text{ is stable } \Leftrightarrow r(\mathbf{c},\mathbf{p}) > 0. \tag{29}$$

In the case of the Routh criterion, $r(\mathbf{c},\mathbf{p})$ is given by the minimum of all entries of the first column of the Routh table associated with the characteristic polynomial of $\Sigma(\mathbf{p},\mathbf{c})$. Finding all robust controllers $\Gamma(\mathbf{c}), \mathbf{c} \in [\mathbf{c}]$ then amounts to characterizing the set

$$\begin{aligned} \mathbb{T}_{\mathrm{c}} &= \{\mathbf{c} \in [\mathbf{c}] \mid \forall \mathbf{p} \in [\mathbf{p}], r(\mathbf{c},\mathbf{p}) > 0\} &(30)\\ &= \{\mathbf{c} \in [\mathbf{c}] \mid \neg(\exists \mathbf{p} \in [\mathbf{p}], r(\mathbf{c},\mathbf{p}) \leq 0)\}, &(31) \end{aligned}$$

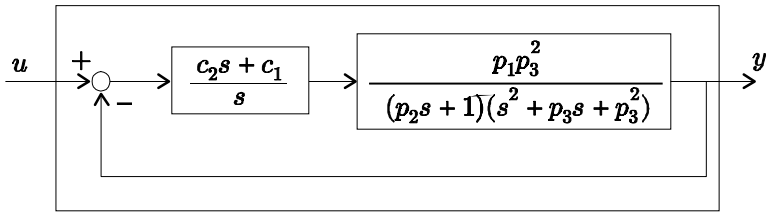

Figure 4: PI controller stabilizing an uncertain model of a process

where $\neg$ is the negation operator. The projection algorithm of Section 4 makes it possible to bracket the set

$$\mathbb{S}_c = \{\mathbf{c} \in [\mathbf{c}] \mid \exists \mathbf{p} \in [\mathbf{p}], r(\mathbf{c}, \mathbf{p}) \leq 0\} \tag{32}$$

between two subpavings $\mathcal{S}_c^-$ and $\mathcal{S}_c^+$ as

$$\mathcal{S}_c^- \subset \mathbb{S}_c \subset \mathcal{S}_c^+. \tag{33}$$

By complementing this equation, we get $\neg\mathcal{S}_c^+ \subset \neg\mathbb{S}_c \subset \neg\mathcal{S}_c^-$. Therefore $[\mathbf{c}]\cap\neg\mathcal{S}_c^+ \subset [\mathbf{c}]\cap\neg\mathbb{S}_c \subset [\mathbf{c}]\cap\neg\mathcal{S}_c^-$. Since $\mathbb{T}_c$ and $\mathbb{S}_c$ form a partition of $[\mathbf{c}]$, (see (31) and (16)) we have $[\mathbf{c}]\cap\neg\mathbb{S}_c = \mathbb{T}_c$ and thus

$$[\mathbf{c}]\cap\neg\mathcal{S}_c^+ \subset \mathbb{T}_c \subset [\mathbf{c}]\cap\neg\mathcal{S}_c^-. \tag{34}$$

The projection algorithm of Section 4 can be used to compute $[\mathbf{c}]\cap\neg\mathcal{S}_c^-$ and $[\mathbf{c}]\cap\neg\mathcal{S}_c^+$ thereby achieving the bracketing of $\mathbb{T}_c$.

As an illustration, assume that the transfer functions of the controller and uncertain model of the process are

$$R(s) = \frac{c_2 s + c_1}{s} \text{ and } G(s) = \frac{p_1 p_3^2}{(p_2 s + 1)(s^2 + p_3 s + p_3^2)}$$

The closed-loop model $\Sigma(\mathbf{p}, \mathbf{c})$ is represented in Figure 4. The parameter vector $\mathbf{c}$ of the controller can be chosen arbitrarily in the box $[\mathbf{c}] = [0, 1]^2$. The parameter vector $\mathbf{p}$ of the model of the process is only known to belong to the box $[0.9, 1.1]^{\times 3}$. The transfer function of $\Sigma(\mathbf{p}, \mathbf{c})$ is

$$H(s) = \frac{(c_2 s + c_1)\, p_1 p_3^2}{p_2 s^4 + (p_2 p_3 + 1)\, s^3 + (p_2 p_3^2 + p_3)\, s^2 + (p_3^2 + c_2 p_1 p_3^2)\, s + c_1 p_1 p_3^2}$$

The first column of the corresponding Routh table is

$$\left(\begin{array}{c} p_2 \\ p_2p_3+1 \\ p_2p_3^2+p_3-\frac{p_2\left(p_3^2+c_2p_1p_3^2\right)}{p_2p_3+1} \\ p_3^2+c_2p_1p_3^2-\frac{(p_2p_3+1)^2\left(c_1p_1p_3^2\right)}{\left(p_2p_3^2+p_3\right)(p_2p_3+1)-p_2\left(p_3^2+c_2p_1p_3^2\right)} \\ c_1p_1p_3^2 \end{array}\right) \tag{35}$$

Since $p_2 > 0$, the closed-loop system $\Sigma(\mathbf{p}, \mathbf{c})$ is asymptotically stable if and only if all the entries of this column are positive, *i.e.*,

$$r\left(\mathbf{c}, \mathbf{p}\right) \triangleq \min \left(\begin{array}{c} p_2p_3+1 \\ p_2p_3^2+p_3-\frac{p_2\left(p_3^2+c_2p_1p_3^2\right)}{p_2p_3+1} \\ p_3^2+c_2p_1p_3^2-\frac{(p_2p_3+1)^2\left(c_1p_1p_3^2\right)}{\left(p_2p_3^2+p_3\right)(p_2p_3+1)-p_2\left(p_3^2+c_2p_1p_3^2\right)} \\ c_1p_1p_3^2 \end{array}\right) > 0$$

The complementary set

$$\mathbb{S}_{\mathrm{c}} \triangleq \left\{\mathbf{c} \in [0,1]^2 \mid \exists \mathbf{p} \in [0.9; 1.1]^{\times 3}, r\left(\mathbf{c}, \mathbf{p}\right) \leq 0\right\} \tag{36}$$

of the set of robust controller $\mathbb{T}_{\mathrm{c}}$ computed by PROJECT is depicted in Figure 5. This set has been obtained for $\varepsilon = 0.02$ in 470 seconds. The subpaving $\mathcal{T}_{\mathrm{c}}^{-}$ in white is an internal approximation of the set of all robustly stable controllers $\mathbb{T}_{\mathrm{c}}$ while $\Delta\mathcal{T}_c$ in light grey is an uncertainty layer in which no conclusion is reached.

References

[1] Scilab homepage, http://www-rocq.inria.fr/scilab/scilab.html.

[2] F. Benhamou, F. Goualard, L. Granvilliers, and J. F. Puget. Revising hull and box consistency. In *Proceedings of the International Conference on Logic Programming*, pages 230–244, Las Cruces, NM, 1999.

[3] F. Benhamou and L. Granvilliers. Automatic generation of numerical redundancies for nonlinear constraint solving. *Reliable Computing*, 3(3):335–344, 1997.

[4] J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

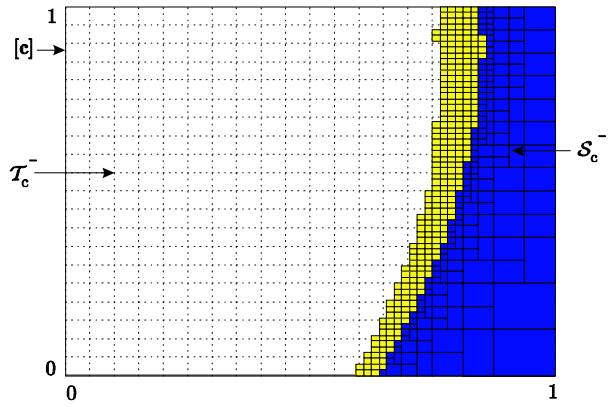

Figure 5: Internal approximation of the set $\mathbb{T}_c$ of all controllers that ensure robust stability (in white); uncertainty layer (in light grey); internal approximation of the complementary set $\mathbb{S}_c$ of $\mathbb{T}_c$ (in dark grey)

[5] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.

[6] J. Garloff and E. Walter, editors. Special Issue on Applications to Control, Signals and Systems. 2000. *Reliable Computing* **6**(3):229-362.

[7] L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed nonlinear estimation using constraint propagation on sets. *International Journal of Control*, 74(18):1772–1782, 2001.

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

[9] L. Jaulin and E. Walter. Guaranteed tuning, with application to robust control and motion planning. *Automatica*, 32(8):1217–1221, 1996.

[10] R. B. Kearfott. Interval mathematics techniques for control theory computations. In K. Bowers and J. Lund, editors, *Computation and Control. Proceedings of the Bozeman Conference*, volume 20 of *Progress in Systems and Control Theory*, pages 169–178. Birkhäuser, Boston, MA, 1989.

[11] N. A. Khlebalin. Interval automatic systems - theory, computer-aided design and applications. *Interval Computations*, 3:106–115, 1992.

[12] L. V. Kolev, V. M. Mladenov, and S. S. Vladov. Interval mathematics algorithms for tolerance analysis. *IEEE Transactions on Circuits and Systems*, 35(8):967–974, 1988.

[13] L.V. Kolev. An interval first-order method for robustness analysis. In *Procedings of the IEEE International Symposium on Circuits and Systems*, pages 2522–2524, Chicago, IL, 1993.

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

[15] S. A. Malan, M. Milanese, and M. Taragna. Robust analysis and design of control systems using interval arithmetics. *Automatica*, 33(7):1363–1372, 1997.

[16] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, PA, 1979.