Guaranteed set computation with subpavings

Michel Kieffer, Luc Jaulin, Isabelle Braems, Eric Walter

GUARANTEED SET COMPUTATION WITH SUBPAVINGS

Michel Kieffer[1]
Luc Jaulin[2]
Isabelle Braems[1]
and Éric Walter[1]
[1] *Laboratoire des Signaux et Systèmes, CNRS — Supélec — Université Paris-Sud Plateau de Moulon, 91192 Gif-sur-Yvette, France*
{kieffer, braems, walter}@lss.supelec.fr
[2] *Laboratoire d'Ingénierie des Systèmes Automatisés, Université d'Angers, France*
jaulin@sciences.univ-angers.fr

Keywords: Binary trees, bounded errors, image evaluation, recursive algorithms, set inversion, subpavings, trees

Abstract This paper is about the approximate representation of compact sets using subpavings, *i.e.* unions of non-overlapping boxes, and about computation on these sets, with particular attention to implementation issues. Some basic operations such as evaluating the intersection or union of two subpavings, or testing whether a box belongs to a subpaving are first presented. The binary tree structure used to describe subpavings then allows a simple implementation of these tasks by recursive algorithms. Algorithms are presented to evaluate the inverse and direct images of a set described by a subpaving. In both cases, a subpaving is obtained that is guaranteed to contain the actual inverse or direct image of the initial subpaving. The effectiveness of these algorithms in characterizing possibly nonconvex on even nonconnected sets is finally illustrated by simple examples.

1. INTRODUCTION

In the interval community, boxes (or interval vectors) are often used to contain the solutions of global optimization problems or of systems of equations. These solution boxes usually have a small volume. On the other hand, problems such as characterizing the stability domain of controllers or estimating parameters in the bounded-error context may have large compact sets as solutions, for which enclosure in a single box would not be detailed enough.

This paper presents results on the description of compact sets by union of nonoverlapping boxes or subpavings. After a brief description of an example motivating the approach in Section 2, subpavings are introduced in Section 3. Particular attention is paid to implementation. Principles and properties of inverse and direct image evaluation of sets are presented in Sections 4 and 5. An example illustrating some features of these algorithms is described in Section 6, before some final remarks and perspectives.

2. WHY DEAL WITH SETS?

The aim of this section is to illustrate the interest of set characterization by an example of problem of practical interest in the context of bounded-error estimation. Assume that the measured output $y(t)$ of a physical system is described by a parametric model $M(\mathbf{p}), \mathbf{p} \in \mathbb{R}^p$, with output $y_{\mathrm{m}}(t, \mathbf{p})$, where $\mathbf{p}$ is a vector of unknown parameters. The model output should resemble the system output as much as possible. The model may be tuned by adjusting $\mathbf{p}$. To achieve this task, n measurements of the system output are collected at time t_i, $i = 1, \ldots, n$. Bounded-error parameter estimation consists of finding all values of $\mathbf{p}$ such that the error between the system and model outputs $e(t_i, \mathbf{p}) = y(t_i) - y_{\mathrm{m}}(t_i, \mathbf{p})$ remains within some prespecified bounds $[\underline{e}_i, \overline{e}_i]$ for $i = 1, \ldots, n$. A value of $\mathbf{p}$ satisfying $e(t_i, \mathbf{p}) \in [\underline{e}_i, \overline{e}_i]$ for $i = 1, \ldots, n$, or equivalently $y_{\mathrm{m}}(t_i, \mathbf{p}) \in [y(t_i) - \overline{e}_i, y(t_i) - \underline{e}_i]$ for $i = 1, \ldots, n$, is said to be *acceptable*. The interval $[y_i] = [y(t_i) - \overline{e}_i, y(t_i) - \underline{e}_i]$ thus contains all acceptable model outputs at time t_i. Bounded-error parameter estimation aims at characterizing the *set* of all acceptable parameter vectors $\mathbb{X} = \{\mathbf{p} \,|\mathbf{y}_{\mathrm{m}}(\mathbf{p}) \in [\mathbf{y}]\}$, where $\mathbf{y}_{\mathrm{m}}(\mathbf{p})$ is the vector of all model outputs $(y_{\mathrm{m}}(t_1, \mathbf{p}), \ldots, y_{\mathrm{m}}(t_n, \mathbf{p}))^{\mathrm{T}}$ and where $[\mathbf{y}]$ is the box $([y_1], \ldots, [y_n])^{\mathrm{T}}$. This problem may be interpreted as a set-inversion problem, as $\mathbb{X}$ may also be written as $\mathbb{X} = \mathbf{y}_{\mathrm{m}}^{-1}([\mathbf{y}])$.

Bounded-error parameter estimation may thus be seen as the charaterization of a possibly nonconvex or even non-connected set. Many other problems in control also require the characterization of sets, for instance bounded-error state estimation or the determination of value sets in robust control.

3. HOW TO DEAL WITH SETS?

Even if an exact description of $\mathbb{X}$ is sometimes possible, see, *e.g.*, [16], this is far from being always the case. When $\mathbb{X}$ is a convex polytope, techniques are available to enclose it in ellipsoids, boxes, simpler polytopes, etc. See the references in [12], [13], [14], [17] for more details.

This paper focuses on the enclosure of compact sets that are not necessarily polytopes in *unions of non-ovelapping boxes*, with special attention to nonlinear problems. Such a description can, at least in principle, approximate compact sets as accurately as desired in the sense, *e.g.*, of the standard Hausdorff distance [1]. Boxes presents the advantage of being very easily manipulated by computers, as they form the heart of interval analysis.

3.1. REPRESENTING UNIONS OF BOXES

It is important to organize the storage of these boxes in memory in order to facilitate further processing (such as taking the intersection or union of solution sets, evaluating their image by a function, *etc.*). The first idea would be to store the boxes in a *list*. However, this structure would not be very efficient for tasks such as checking whether a box is included in the set formed by the union of the boxes belonging to a given list.

To allow a more efficient organization, we shall require that all the boxes to be considered result from successive bisections of a *root* box $[\mathbf{x}]_0 \subset \mathbb{R}^n$, according to some canonical bisection rule. Such bisection rule may, for instance, be that each box $[\mathbf{x}]$ is cut across its main component j, defined as $j = \min\{i\,|w([x_i]) = w([\mathbf{x}])\}$, where $w(.)$ denotes the width of an interval or a box. The boxes resulting from the bisection of $[\mathbf{x}]$ are $L[\mathbf{x}] = \left([x_1], \ldots, [\underline{x}_j, (\underline{x}_j + \overline{x}_j)/2], \ldots [x_n]\right)$ and $R[\mathbf{x}] = \left([x_1], \ldots, [(\underline{x}_j + \overline{x}_j)/2, \overline{x}_j], \ldots [x_n]\right)$. A union of boxes obtained in this manner will be called a *regular subpaving* [3], [8], [9]. The set of regular subpavings whose root box is $[\mathbf{x}]$ will be denoted by $\mathcal{RSP}([\mathbf{x}])$.

Many interval algorithms naturally provides solutions that are regular subpavings.

3.2. BINARY TREES AND REGULAR SUBPAVINGS

Regular subpavings extend quadtrees and octtrees of computer geometry (see, *e.g.*, [15]) to higher dimensions, and the same type of technique based on binary trees can be used. The binary tree will be used to describe the boxes of the regular subpaving and how they were bisected and selected from the root box. A binary tree $\mathcal{T}$ is a finite collection of *nodes*. $\mathcal{T}$ may be empty, or may consist of a single node or of two subtrees: the *left* and *right* subtrees, respectively denoted by $L\mathcal{T}$ and $R\mathcal{T}$. Here, each node represents a box $[\mathbf{x}]$, which may be the root box of the subpaving or

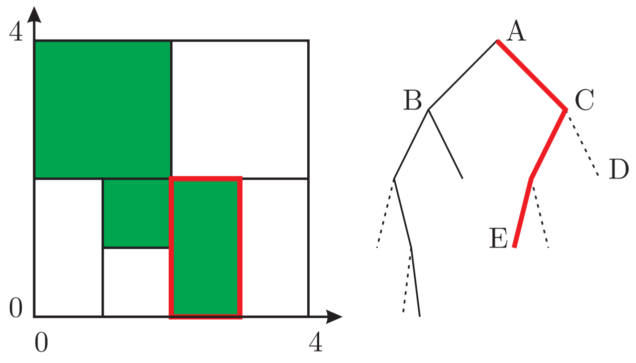

Figure 1 A subpaving and its binary tree representation. The branch in bold represent the successive bisections and selections of $[\mathbf{x}]_0 = [0,4] \times [0,4]$ to get $LLR\,[\mathbf{x}]_0 = [2,3] \times [0,2]$

a box obtained from the root box by bisections. The shape of the tree is determined by the bisections and selections which have lead to the boxes of the regular subpaving, see Figure 3.2. The root node A of the tree $\mathcal{T}$ represented on the right corresponds to the root box $[\mathbf{x}]_0 = [0,4] \times [0,4]$ of the subpaving represented on the left. The fork stemming from A indicates a bisection of $[\mathbf{x}]_0$. A has two subtrees, the roots of which are the nodes B and C. These *sibling nodes* (they stem from the same node) respectively represent $L\,[\mathbf{x}]_0 = [0,2] \times [0,4]$ and $R\,[\mathbf{x}]_0 = [2,4] \times [0,4]$. The node C has only one subtree, as the box $[2,4] \times [2,4]$ corresponding to D does not belong to the subpaving. The node E has no children, it is a *leaf*, which corresponds to $LLR\,[\mathbf{x}]_0 = [2,3] \times [0,2]$. Each leaf represents a box belonging to the subpaving. A regular subpaving is *minimal* if it has no sibling nodes that are leaves.

Regular subpavings and their binary tree representations will be considered indifferently, and the vocabulary used for binary trees will also be used for subpavings. This type of representation allows complex tasks to be performed by very simple recursive algorithms, as we shall see.

3.3. BASIC OPERATIONS

The four basic operations on regular subpavings to be considered are reuniting sibling subpavings, taking the union or intersection of subpavings, and testing whether a box is included in a subpaving.

Reuniting sibling subpavings: this operation is intended to simplify the description of subpavings by making them minimal. Consider a box $[\mathbf{x}]$ and two regular subpavings $\mathbb{X} \in \mathcal{RSP}\,(L[\mathbf{x}])$ and $\mathbb{Y} \in \mathcal{RSP}\,(R[\mathbf{x}])$. These subpavings are siblings as they have the same parent box $[\mathbf{x}]$. The *reunited* subpaving $\mathbb{Z} \triangleq (\mathbb{X}|\mathbb{Y}) \in \mathcal{RSP}([\mathbf{x}])$ is defined and computed as follows:

```
**Algorithm** REUNITE(in: $\mathbb{X}, \mathbb{Y}, [\mathbf{x}]$, out: $\mathbb{Z} \triangleq (\mathbb{X}|\mathbb{Y})$)
  if $\mathbb{X} = L[\mathbf{x}]$ and $\mathbb{Y} = R[\mathbf{x}]$, then $\mathbb{Z} := [\mathbf{x}]$;
  else if $\mathbb{X} = \emptyset$ and $\mathbb{Y} = \emptyset$, then $\mathbb{Z} := \emptyset$;
  else, $L\mathbb{Z} := \mathbb{X}$ and $R\mathbb{Z} := \mathbb{Y}$.
```

Each of these instructions is trivial to implement with a binary tree representation. For instance, the instructions $L\mathbb{Z} := \mathbb{X}$ and $R\mathbb{Z} := \mathbb{Y}$ amount to grafting the trees $\mathbb{X}$ and $\mathbb{Y}$ to a node to form the tree $\mathbb{Z}$.

Intersecting subpavings: If $\mathbb{X} \in \mathcal{RSP}([\mathbf{x}])$ and $\mathbb{Y} \in \mathcal{RSP}([\mathbf{x}])$, then $\mathbb{Z} = \mathbb{X} \cap \mathbb{Y}$ is also a subpaving of $\mathcal{RSP}([\mathbf{x}])$. It only contains the nodes shared by the binary trees representing $\mathbb{X}$ and $\mathbb{Y}$, and can be computed by the following recursive algorithm:

```
**Algorithm** INTERSECT(in: $\mathbb{X}, \mathbb{Y}, [\mathbf{x}]$, out: $\mathbb{Z} = \mathbb{X} \cap \mathbb{Y}$)
  if $\mathbb{X} = \emptyset$ or $\mathbb{Y} = \emptyset$ then return $\emptyset$;
  if $\mathbb{X} = [\mathbf{x}]$ then return $\mathbb{Y}$;
  if $\mathbb{Y} = [\mathbf{x}]$ then return $\mathbb{X}$;
  return (INTERSECT($L\mathbb{X}, L\mathbb{Y}, L[\mathbf{x}]$)|INTERSECT($R\mathbb{X}, R\mathbb{Y}, R[\mathbf{x}]$));
```

Taking the union of subpavings: If $\mathbb{X} \in \mathcal{RSP}([\mathbf{x}])$ and $\mathbb{Y} \in \mathcal{RSP}([\mathbf{x}])$, then $\mathbb{Z} = \mathbb{X} \cup \mathbb{Y}$ also belongs to $\mathcal{RSP}([\mathbf{x}])$. $\mathbb{Z}$ is computed by putting together all nodes of the two binary trees representing $\mathbb{X}$ and $\mathbb{Y}$. Again, this can be done recursively:

```
**Algorithm** UNION(in: $\mathbb{X}, \mathbb{Y}, [\mathbf{x}]$, out: $\mathbb{Z} = \mathbb{X} \cup \mathbb{Y}$)
  if $\mathbb{X} = \emptyset$ or if $\mathbb{Y} = [\mathbf{x}]$ then return $\mathbb{Y}$;
  if $\mathbb{Y} = \emptyset$ or if $\mathbb{X} = [\mathbf{x}]$ then return $\mathbb{X}$;
  return (UNION($L\mathbb{X}, L\mathbb{Y}, L[\mathbf{x}]$)|UNION($R\mathbb{X}, R\mathbb{Y}, R[\mathbf{x}]$));
```

Testing whether a box $[\mathbf{z}]$ is included in a subpaving: $\mathbb{X} \in \mathcal{RSP}([\mathbf{x}])$. This test is straightforward in four cases. It holds true if $[\mathbf{z}]$ is empty, or if $\mathbb{X}$ is reduced to a single box $[\mathbf{x}]$ and $[\mathbf{z}] \subset [\mathbf{x}]$. It holds false if $\mathbb{X}$ is empty and $[\mathbf{z}]$ is not, or if $[\mathbf{z}]$ is not in the root box of $\mathbb{X}$. These basic tests will first be applied to the root of the tree representing the subpaving. If none of the four simple cases is satisfied, these basic tests are recursively applied on the left and right subtrees. The following algorithm summarizes the process:

```
**Algorithm** INSIDE(in: $[\mathbf{z}], \mathbb{X}$, out: $t$)
  if $[\mathbf{z}] = \emptyset$ or if $\mathbb{X}$ is a box $[\mathbf{x}]$ and $[\mathbf{z}] \subset [\mathbf{x}]$ then return 1;
  if $\mathbb{X} = \emptyset$ or if $[\mathbf{z}] \cap$ root($\mathbb{X}$) $= \emptyset$ then return 0;
  return (INSIDE($[\mathbf{z}] \cap L[\mathbf{x}], L\mathbb{X}$) $\wedge$ INSIDE($[\mathbf{z}] \cap R[\mathbf{x}], R\mathbb{X}$));
```

Note that $\wedge$ corresponds to an interval version of the logical operator `AND`. When $[\mathbf{z}] \subset \mathbb{X}$ 1 is returned, when $[\mathbf{z}] \cap \mathbb{X} = \emptyset$ 0 is returned and when $[\mathbf{z}]$ overlaps the boundary of $\mathbb{X}$ $[0,1]$ is returned.

Remark 1 *Binary trees are a well-known data structure and many libraries provide this data type. However, in most cases, these libraries are intended to implement sorting algorithms, and thus not suited to the implementation of operations on sets. This is why we choose to implement subpavings from scratch using the* PROFIL/BIAS *library [11]. The* C++ *source code is freely available on request.* $\diamondsuit$

4. INVERSE IMAGE EVALUATION

Let $\mathbf{f}$ be a possibly nonlinear function from $\mathbb{R}^n$ to $\mathbb{R}^m$ and let $\mathbb{Y}$ be a regular subpaving included in $\mathbb{R}^m$. Inverse image evaluation is the characterization of $\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\} = \mathbf{f}^{-1}(\mathbb{Y})$. Set inversion of Section 2 is a special case of this problem.

For any subpaving $\mathbb{Y} \subset \mathbb{R}^m$ and for any function $\mathbf{f}$ admitting an inclusion function $[\mathbf{f}](.)$, a subpaving $\overline{\mathbb{X}}$ containing the set $\mathbb{X}$ can be obtained with the algorithm SIVIA (Set Inverter Via Interval Analysis, [6], [7]) that will now be described in the context of regular subpavings.

To compute $\overline{\mathbb{X}}$, SIVIA requires a (possibly very large) search subpaving $\mathbb{S}$ to which $\overline{\mathbb{X}}$ is guaranteed to belong. To facilitate presentation, Figure 4 describes the basic steps of SIVIA, in the case of a search subpaving reduced to a box $[\mathbf{x}_0]$. The general procedure is easily derived from this simplified example.

To obtain $\overline{\mathbb{X}}$, the same procedure will be applied to each node of $\mathbb{S}$. For any given node N of the binary tree describing $\mathbb{S}$, the image of the box $[\mathbf{x}_\mathrm{N}]$ corresponding to this node is evaluated by the inclusion function $[\mathbf{f}](.)$. Four cases may be encountered.

1. If $[\mathbf{f}]([\mathbf{x}_\mathrm{N}])$ has a nonempty intersection with $\mathbb{Y}$, but is not entirely in $\mathbb{Y}$, then $[\mathbf{x}_\mathrm{N}]$ may contain a part of the solution set (Figure 4a); $[\mathbf{x}_\mathrm{N}]$ and the associated node N are said to be *undetermined.* The same test should be recursively applied on the nodes stemming from N, if they exist. If N is a leaf, and if the width of $[\mathbf{x}_\mathrm{N}]$ is greater than a prespecified precision parameter ε, $[\mathbf{x}_\mathrm{N}]$ should be bisected (this implies to the growth of two offsprings from N) and the test should be recursively applied on these newly generated nodes.

2. If $[\mathbf{f}]([\mathbf{x}_N])$ has an empty intersection with $\mathbb{Y}$, $[\mathbf{x}_N]$ does not belong to the solution subpaving, and N can be cut off from the solution tree (Figure 4b).

3. If $[\mathbf{f}]([\mathbf{x}_N])$ is entirely in $\mathbb{Y}$, $[\mathbf{x}_N]$ belongs to the solution subpaving $\mathbb{X}$, and N is in the solution tree (Figure 4c).

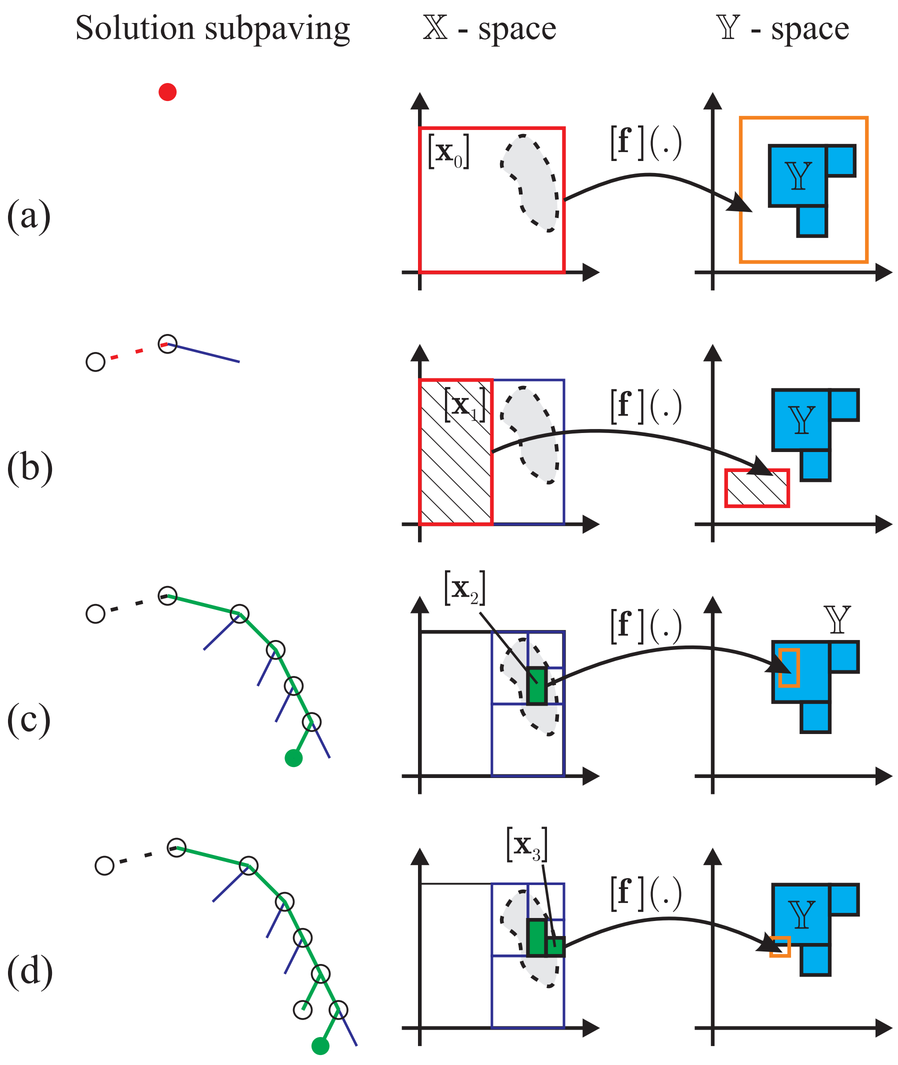

Figure 2 Four situations encountered by the Sivia algorithm (a) the box $[\mathbf{x}_0]$ to be checked is undetermined and will be bisected; (b) the box $[\mathbf{x}_1]$ does not intersect $\mathbb{Y}$ and is rejected; (c) the box $[\mathbf{x}_2]$ is entirely in $\mathbb{Y}$ and is stored in the solution subpaving; (d) the box $[\mathbf{x}_3]$ is undetermined but deemed to small to be bisected, it is also stored in the solution subpaving to set an outer approximation $\overline{\mathbb{X}}$ of $\mathbb{X}$ upon completion of the algorithm

4. The last case is depicted on Figure 4d. If the box considered is undetermined, but its width is lower than ε, then it is deemed small enough to be stored in the outer approximation $\overline{\mathbb{X}}$ of the solution subpaving.

The following algorithm summarizes this procedure.

```
Algorithm SIVIA(in: $[\mathbf{f}], \mathbb{Y}, \mathbb{S}, \varepsilon$, out: $\overline{\mathbb{X}}$)
  $[\mathbf{x}]$ :=root($\mathbb{S}$);
  $[test]$ := INSIDE($[\mathbf{f}]([\mathbf{x}]), \mathbb{Y}$);
  if $[test] = 0$ then return $\emptyset$;                        // Figure 4(b)
  if $[test] = 1$ then return $\mathbb{S}$;                        // Figure 4(c)
  if $w([\mathbf{x}]) < \varepsilon$ then return $\mathbb{S}$;                   // Figure 4(d)
  return (SIVIA($[\mathbf{f}], \mathbb{Y}, L\mathbb{S}, \varepsilon$)|SIVIA($[\mathbf{f}], \mathbb{Y}, R\mathbb{S}, \varepsilon$)); // Figure 4(a)
```

The real positive number ε is an accuracy parameter, which determines the maximum width of the boxes that compose $\overline{\mathbb{X}}$. Recall that the reunification operator (|) performs the union of two sibling subpavings. This allows SIVIA to return $\overline{\mathbb{X}}$ as a minimal subpaving.

The convergence of the initial version of this algorithm, allowing only inversion of boxes, has been studied in [6]. The proofs given there easily extend to the inversion of subpavings.

5. DIRECT IMAGE EVALUATION

Computing the direct image of a subpaving by a function is slightly more complicated than computing a reciprocal image, because interval analysis does not provide any inclusion test for the point test $t(\mathbf{y}) = (\mathbf{y} \in \mathbf{f}(\mathbb{X}))$ directly. Note that even this point test is very difficult to evaluate in general, contrary to the point test $t(\mathbf{x}) = \left(\mathbf{x} \in \mathbf{f}^{-1}(\mathbb{Y})\right)$ involved in set inversion. Indeed, to test whether $\mathbf{x} \in \mathbf{f}^{-1}(\mathbb{Y})$, it suffices to compute $\mathbf{f}(\mathbf{x})$ and to check whether it is in $\mathbb{Y}$. On the other hand, to test whether $\mathbf{y} \in \mathbf{f}(\mathbb{X})$, one must study whether the set of equations $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ admits at least one solution under the constraint $\mathbf{x} \in \mathbb{X}$, which is usually far from simple.

Assume that $\mathbf{f}$ is continuous and that an inclusion function $[\mathbf{f}]$ for $\mathbf{f}$ is available. The algorithm presented below generates a regular subpaving $\overline{\mathbb{Y}}$ that contains the image $\mathbb{Y}$ of a regular subpaving $\mathbb{X}$ by $\mathbf{f}$ (see also [8], [9]). Thus $\overline{\mathbb{Y}}$ is an outer approximation of $\mathbb{Y}$. The set $\mathbb{Y}$ is included into the box $[\mathbf{f}]([\mathbb{X}])$ and also into the image by the inclusion function of the smallest box containing $\mathbb{X}$. The algorithm proceeds in three steps, namely *mincing*, *evaluation*, and *regularization* (see Figure 5). As with SIVIA, the precision of the outer approximation will be governed by the real $\varepsilon > 0$ to be chosen by the user. During mincing, a non-minimal regular subpaving $\mathbb{X}_\varepsilon$ is built, such that the width of each of its boxes is less than ε. During evaluation, a box $[\mathbf{f}]([\mathbf{x}])$ is computed for each box $[\mathbf{x}]$ of $\mathbb{X}_\varepsilon$, and all the resulting boxes are stored into a list $\mathcal{Y}(\varepsilon)$. During regularization, a regular subpaving $\overline{\mathbb{Y}}(\varepsilon)$ is computed that contains the union of all boxes of $\mathcal{Y}(\varepsilon)$. This regularization can be viewed as a

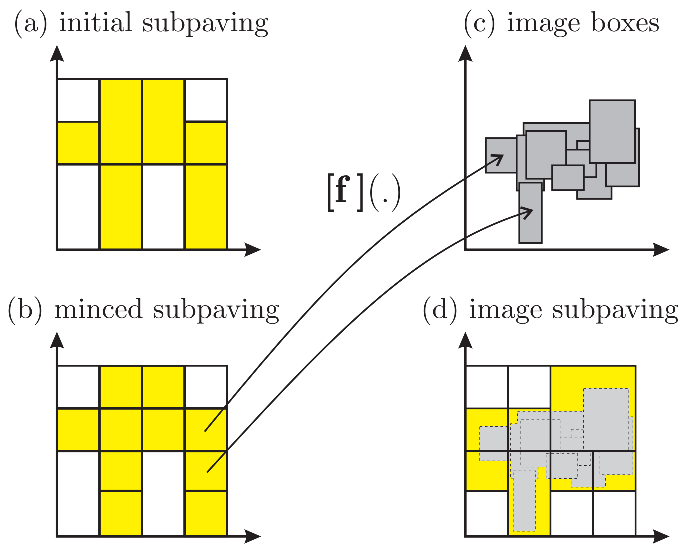

Figure 3 The three steps of ImageSp. $(a) \to (b)$: mincing; $(b) \to (c)$: evaluation; $(c) \to (d)$: regularization

call of Sivia to invert $\mathcal{Y}(\varepsilon)$ by the identity function. Indeed, since $\mathbf{f}(\mathbb{X}) \subset \mathcal{Y}(\varepsilon)$, which is equivalent to $\mathbf{f}(\mathbb{X}) \subset \mathrm{Id}^{-1}(\mathcal{Y}(\varepsilon))$, one has $\mathbf{f}(\mathbb{X}) \subset$ Sivia$([t], [\mathbf{f}]([\mathbb{X}]), \varepsilon)$, where $[t]$ is an inclusion function for $t(\mathbf{y}) = (\mathbf{y} \in \mathcal{Y}(\varepsilon))$, denoted by $[t]([\mathbf{y}]) = ([\mathbf{y}]\,[\in]\,\mathcal{Y}(\varepsilon))$. The resulting algorithm is as follows:

```
Algorithm ImageSp(in: $[\mathbf{f}]$, $\mathbb{X}$, $\varepsilon$, out: $\overline{\mathbb{Y}}$)
  $\mathbb{X}_\varepsilon$ :=mince($\mathbb{X}, \varepsilon$) ;
  $\mathcal{Y}(\varepsilon) = \emptyset$;
  For each $[\mathbf{x}] \in \mathbb{X}_\varepsilon$, $\mathcal{Y}(\varepsilon) := \mathcal{Y}(\varepsilon) \cup \{[\mathbf{f}]([\mathbf{x}])\}$ ;
  return Sivia($[t], [\mathbf{f}]([\mathbb{X}]), \varepsilon$);
```

Since $\mathcal{Y}(\varepsilon)$ is not a subpaving, implementation is not trivial, see [8] for details. The complexity and convergence properties of ImageSp have been described in [4] and [8].

6. EXAMPLES

The first example is the characterization of the set

$$\mathbb{X}_1 = \left\{(x_1, x_2) \in \mathbb{R}^2 \,\middle|\, x_1^4 - x_1^2 + 4x_2^2 \in [-0.1, 0.1]\right\}$$

This set-inversion-problem is solved by Sivia for $\mathbb{S} = [-3, 3] \times [-3, 3]$ and $\varepsilon = 0.1$. The resulting subpaving $\overline{\mathbb{X}}_1$ is represented on Figure 6(a).

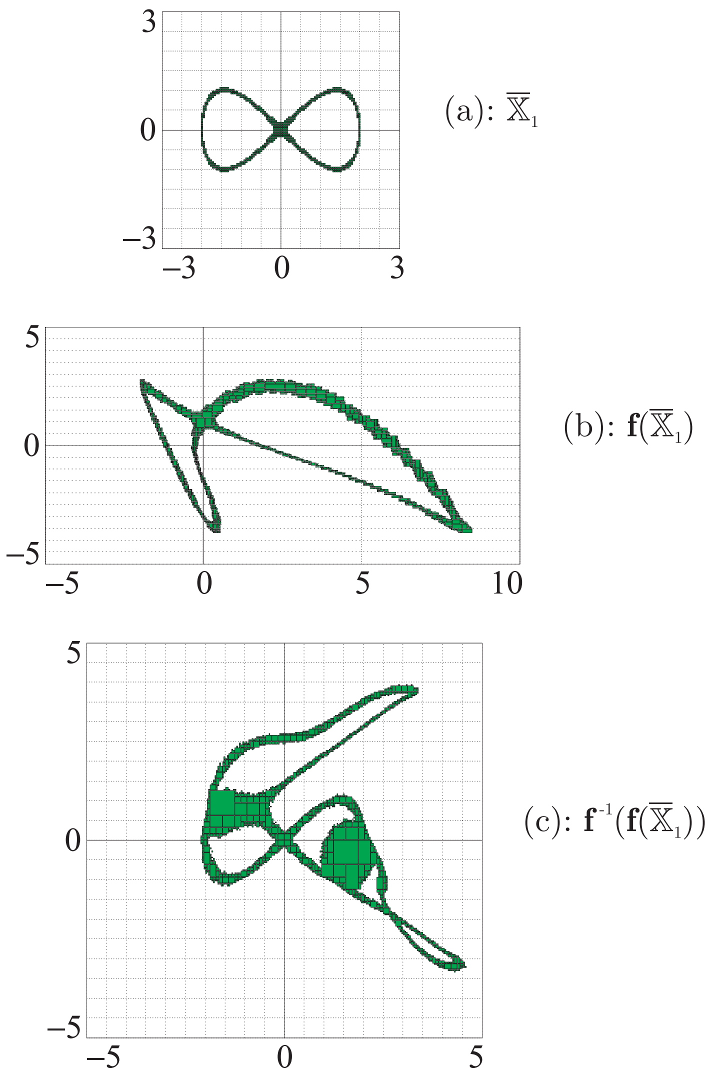

Figure 4 Illustration of the inverse and direct image evaluation algorithms

The second example is the evaluation of an outer approximation of the image $\mathbb{X}_2$ of $\mathbb{X}_1$ by the function

$$\mathbf{f}\left(x_1, x_2\right) = \left(\begin{array}{c} \left(x_1 - 1\right)^2 - 1 + x_2 \\ -x_1^2 + \left(x_2 - 1\right)^2 \end{array} \right).$$

With $\varepsilon = 0.1$, ImageSp yields the subpaving $\overline{\mathbb{X}}_2$ depicted on Figure 6(b).

The last example is the characterization of the image of $\overline{\mathbb{X}}_2$ by the inverse of $\mathbf{f}(.)$, *i.e.,* $\mathbb{X}_3 = \left\{\mathbf{f}^{-1}\left(\overline{\mathbb{X}}_2\right)\right\}$. The function $\mathbf{f}(.)$ is not invertible (in the common sense) in $\mathbb{R}^2$. Thus, an explicit form of $\mathbf{f}^{-1}(.)$ is not available for the whole search domain and the problem will be treated as a set inversion problem. Again, Sivia is used with $\mathbb{S} = [-5,5]\times[-5,5]$ and $\varepsilon = 0.1$. The solution subpaving $\overline{\mathbb{X}}_3$ is represented on Figure 6(c). We have $\overline{\mathbb{X}}_1 \subset \mathbf{f}^{-1}\left(\mathbf{f}\left(\overline{\mathbb{X}}_1\right)\right)$. The initial set $\overline{\mathbb{X}}_1$ is clearly present. The result is slightly fatter, due to error accumulation during inverse and direct image evaluation. Additional parts have appeared because $\mathbf{f}(.)$ is only invertible in a set-theoretic sense.

7. CONCLUSIONS

Regular subpavings form an attractive class of basic objects for the representation of compact sets and for computation on such sets. Simple tasks such as evaluating the union or intersection of two subpavings are very easily performed when these subpavings are represented by binary trees. More sophisticated operations such as inverse or direct image evaluation are also facilitated. Even if they are restricted to low-dimensional problems, ImageSp and Sivia have found application in nonlinear state estimation problems [9], [10] or in measurement problems such as grooves dimensioning using remote field eddy current inspection [2]. ImageSp is still a very preliminary algorithm that could easily be improved. Work is under way to take advantage of interval constraint propagation to improve state estimation algorithms, among others [5].

References

[1] M. Berger. *Geometry I and II.* Springer-Verlag, Berlin, 1987.

[2] S. Brahim-Belhouari, M. Kieffer, G. Fleury, L. Jaulin, and E. Walter. Model selection via worst-case criterion for nonlinear bounded-error estimation. *IEEE Trans. on Instrumentation and Measurement*, 49(3):653–658, 2000.

[3] L. Jaulin. *Solution globale et garantie de problèmes ensemblistes ; application à l'estimation non linéaire et à la commande robuste.* PhD dissertation, Université Paris-Sud, Orsay, 1994. Available at: http://istia/ jaulin/thesejaulin.zip.

[4] L. Jaulin. *Le Calcul Ensembliste Par Analyse Par Intervalles.* Habilitation à diriger des recherches, Université d'Orsay, 2000. Available at: http://istia/ jaulin/hdrjaulin.zip.

[5] L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed nonlinear estimation using constraint propagation on sets. *International*

Journal of Control (accepted for publication), 2000.

[6] L. Jaulin and E. Walter. Guaranteed nonlinear parameter estimation from bounded-error data via interval analysis. *Math. and Comput. in Simulation*, 35:1923–1937, 1993.

[7] L. Jaulin and E. Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.

[8] M. Kieffer. *Estimation ensembliste par analyse par intervalles, application à la localisation d'un véhicule.* PhD dissertation, Université Paris-Sud, Orsay, 1999.

[9] M. Kieffer, L. Jaulin, and E. Walter. Guaranteed recursive nonlinear state estimation using interval analysis. In *Proc. 37th IEEE Conference on Decision and Control*, pages 3966–3971, Tampa, December 16-18, 1998.

[10] M. Kieffer, L. Jaulin, E. Walter, and D. Meizel. Robust autonomous robot localization using interval analysis. *Reliable Computing*, 6:337–362, 2000.

[11] O. Knüppel. PROFIL/BIAS – A fast interval library. *Computing*, 53:277–287, 1994.

[12] M. Milanese, J. Norton, H. Piet-Lahanier, and E. Walter. *Bounding Approaches to System Identification.* Plenum Press, New York, 1996.

[13] J. P. Norton (Ed.). Special issue on bounded-error estimation: Issue 1. *Int. J. of Adaptive Control and Signal Processing*, 8(1):1–118, 1994.

[14] J. P. Norton (Ed.). Special issue on bounded-error estimation: Issue 2. *Int. J. of Adaptive Control and Signal Processing*, 9(1):1–132, 1995.

[15] H. Samet. *Design and Analysis of Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods.* Addison-Wesley, Reading, 1989.

[16] E. Walter and H. Piet-Lahanier. Exact recursive polyhedral description of the feasible parameter set for bounded-error models. *IEEE Trans. on Automatic Control*, 34(8):911–915, 1989.

[17] E. Walter (Ed.). Special issue on parameter identification with error bounds. *Mathematics and Computers in Simulation*, 32(5&6):447–607, 1990.