An Efficient Implementation of the SIVIA Algorithm in a High-Level Numerical Programming Language*

Pau Herrero, Pantelis Georgiou and Christofer Toumazou
Center for Bio-Inspired Technology, Institute of Biomedical Engineering, Imperial College London, SW7 2AZ, United Kingdom
{pherrero;pantelis;c.toumazou}@imperial.ac.uk

Benot Delaunay and Luc Jaulin
Lab-STICC, ENSTA-Bretagne, 29806 Brest cedex 9, France
{benoit.delaunay;luc.jaulin}@ensta-bretagne.fr

Abstract

High-level, numerically oriented programming languages such as Matlab, Scilab or Octave are popular and well-established tools in the scientific and engineering communities. However, their computational efficiency sometimes limits their use in certain areas where intensive numerical computations are required, such as interval analysis. In this paper, we present an efficient implementation of the well known Set Inverter via Interval Analysis (SIVIA) algorithm in Matlab that has a computational efficiency comparable to its C++ counterpart. Such implementation aims at promoting and facilitating the use of SIVIA algorithm by the aforementioned communities. The source code of a Matlab implementation is freely distributed.

Keywords: Interval analysis, set inversion, high-level numerical language, Matlab
AMS subject classifications: 65-00

1 Introduction

Set Inverter via Interval Analysis (SIVIA) [8] is a well-known algorithm in the interval analysis [12] community that characterises the solution set of a system of non-linear real constraints by enclosing it between internal and external unions of interval boxes (pavings). SIVIA has been applied successfully in many areas of engineering such as control engineering and robotics [7]. The need for an efficient implementation of SIVIA is driven by its high computational complexity due to its branch-and-bound nature, which in the worst case is exponential on the number of variables [8].

*Submitted: September 13, 2012; Revised: October 15, 2012; Accepted: October 19, 2012.

Typical programming languages used to implement SIVIA have been C++, Fortran 90 and ADA, which provide good computational efficiency, in terms of time and memory, and have operator overloading capabilities, that allow implementing a user-friendly interval arithmetic that facilitates the writing of arithmetic expressions involving interval variables. However, these programming languages present the disadvantage of having a relatively slow learning curve and may have portability issues between different platforms, something that limits their use in many areas of science and engineering.

On the other side, high-level, numerically oriented (HLNO) programming languages such as Matlab, Scilab and Octave are extensively used by engineers, physicists and mathematicians due to their user-friendliness, portability, good technical support, extensive number of toolboxes, big online community of users, good documentation and powerful data plotting tools. However, these languages are not particularly known for their computational efficiency, mainly due to their interpreted nature, and can be specially inefficient if they are not used in the way they are meant to be used, *e.g.,* using explicit *for loops* instead of array computations or built-in functions. Therefore, an efficient implementation of SIVIA in a HLNO programming language is desired to facilitate and promote its use in the scientific and engineering communities.

Different attempts to implement SIVIA algorithm using HLNO programming languages have been done. The SCS Toolbox [15] is a Matlab implementation of SIVIA which based on the interval arithmetic library Intlab [13]. However, the computational efficiency of this implementations remains very low compared to other existing implementations in C++. In this paper, we present a novel implementation paradigm for SIVIA algorithm that is well suited to current available HLNO programming languages that include a matrix library and operator overloading capability. The proposed implementation has been coded in Matlab and its computational efficiency has been proven to be comparable to an existing C++ implementations. Such implementation can be ported easily to any other of the existing HLNO programming languages. Similarly to the original SIVIA implementation, the novel implementation allows the use of interval contractors [6], which helps reduce the complexity of the algorithm.

The paper is organised as follows. Section 2 reviews the notion of *set inversion* and the original implementation of the SIVIA algorithm. Section 3 presents a novel vector implementation of SIVIA optimised for the currently available HLNO programming languages. Then, the concept of using *interval contractors* within SIVIA [6], and its use within the novel SIVIA implementation, is presented in Section 4. Section 5 introduces some tips about a Matlab implementation of the proposed algorithm. Section 6 presents some numerical results of the presented implementation in Matlab and compares them with a SIVIA implementation in C++. Finally, some applications are presented in Section 7 to show the potential of the tool. Section 8 summarises the conclusions about the presented work and introduces some ongoing work.

2 Set Inversion and the SIVIA Algorithm

Let f be a function from $\mathbb{R}^n \to \mathbb{R}^p$ and let $\mathbb{Y}$ be a subset of $\mathbb{R}^p$, where $(n,p) \in \mathbb{N}^{*2}$. Set inversion is the characterisation of the set defined by:

$$\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) \in \mathbb{Y}\} = f^{-1}(\mathbb{Y}). \tag{1}$$

For any $\mathbb{Y} \in \mathbb{R}^p$, for any function f admitting an *inclusion function* $[f](.)$ from $\mathbb{IR}^n \to \mathbb{IR}^p$ [12], being $\mathbb{IR}$ the set of real intervals; and by choosing an *inclusion test* $[t]$ defined

by:

$$[t](\mathbf{x}) = \begin{cases} \mathit{true} & \textbf{if } [f]([\mathbf{x}]) \subset \mathbb{Y}, \\ \mathit{false} & \textbf{if } [f]([\mathbf{x}]) \cap \mathbb{Y} = \emptyset, \\ \mathit{undecided} & \textbf{otherwise}. \end{cases} \tag{2}$$

Set Inverter Via Interval Analysis (SIVIA) [8] approximates the set defined by Equation (1) by means of 3 sets of axis-aligned boxes of $\mathbb{R}^n$ $(\mathcal{S}, \mathcal{N}, \mathcal{E})$ of $\mathbb{R}^n$, also referred to as *pavings*, such that:

$$\mathcal{S} \subset \mathbb{X} \subset (\mathcal{S} \cup \mathcal{E}), \tag{3}$$

$$(\mathcal{N} \cap \mathbb{X}) = \emptyset, \tag{4}$$

$$\forall [\mathbf{x}] \in \mathcal{E},\ \mathit{Width}([\mathbf{x}]) < \epsilon, \tag{5}$$

where *Width* is a real valued function that returns the maximum relative width of an interval box $[x]$ with respect to the initial box $[x_0]$, that is:

$$\mathit{Width} : [\mathbf{x}] = \bigotimes_{i \in [1,n]} [x_i] \longmapsto \max_{i \in [1,n]} \frac{\mathit{width}([x_i])}{\mathit{width}([x_{0_i}])}, \tag{6}$$

with *width* defined for a single interval as $\mathit{width} : [x] = [a, b] \mapsto |b - a|$; and ϵ is an arbitrary positive number that allows control of the accuracy of the approximated set. Algorithm (1) describes the classic implementation of SIVIA algorithm.

Algorithm 1 Classic implementation of SIVIA Algorithm

Require: • $\mathbb{Y} \subset \mathbb{R}^p$ • $[\mathbf{x_0}] \in \mathbb{IR}^n$ • $[f] : \mathbb{IR}^n \to \mathbb{IR}^p$ • $\epsilon > 0$
Ensure: • $\mathcal{S}$, $\mathcal{N}$ and $\mathcal{E}$ such that: • $\mathcal{S} \subset (\mathbb{X} \cap [\mathbf{x_0}]) \subset \mathcal{S} \cup \mathcal{E}$ • $\mathcal{N} \cap \mathbb{X} = \emptyset$
• $\mathit{width}([\mathbf{x}]) < \epsilon$ $(\forall [\mathbf{x}] \in \mathcal{E})$

```
1: function SIVIA([f], 𝕐, [x0], ϵ)
2:     S ← N ← E ← ∅
3:     L ← {[x0]}
4:     while L ≠ ∅ do
5:         [x] ← pop(L)              ▷ pop: Retrieves and removes the first interval box from a list
6:         if [f]([x]) ⊂ 𝕐 then push(S, [x])          ▷ push: Adds an interval box to a list
7:         else if [f]([x]) ∩ 𝕐 = ∅ then push(N, [x])
8:         else if Width([x]) < ϵ then push(E, [x])   ▷ Width: Returns width of largest interval
9:         else
10:            {[x1], [x2]} = Bisect([x])            ▷ Bisect: Bisects a box and returns resulting boxes
11:            push(L, [x1])
12:            push(L, [x2])
13:        end if
14:    end while
15:    return (S, N, E)
16: end function
```

2.1 Example

Consider the set inversion problem defined by

$$\mathbb{X} = \left\{ (x \times y) \in ([-3,3] \times [-3,3]) \mid x^2 + y^2 + x \cdot y \in [1,2] \right\}. \tag{7}$$

Figure (1) shows a graphical representation of the result provided by the SIVIA algorithm to the set inversion problem stated by Equation (7) with $\epsilon = 0.01$. Blue boxes correspond to $\mathcal{N}$ (non-solutions), red boxes $\mathcal{S}$ (solutions) and yellow boxes to $\mathcal{E}$ (undecided).

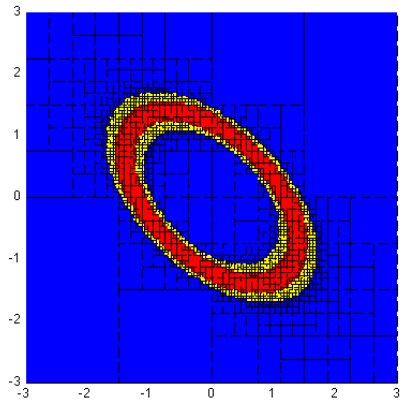

Figure 1: Graphical representation of the result provided by SIVIA algorithm for the set inversion problem defined by Equation (7) with $\epsilon = 0.01$, where blue boxes correspond to $\mathcal{N}$, red boxes $\mathcal{S}$ and yellow boxes to $\mathcal{E}$.

2.2 Implementing SIVIA in a HNLO Programming Language

Implementing SIVIA in a HLNO programming language, as presented in Algorithm (1), is fairly straightforward if an interval arithmetic library is available [13]. However, its computational efficiency, compared to an equivalent implementation in C++, is dramatically inferior. For example, let us consider the problem stated by Equation (7) with an absolute $\epsilon = 10^{-4}$. A C++ implementation of SIVIA solves such a problem in about 0.55 s (Intel Core 2 Duo E8500 (3.16 Ghz) - 4 GB RAM), while an equivalent Matlab implementation of the same algorithm, on the same computer, takes 1700 s. The reason of such inefficiency is due to the fact that HLNO programming languages are parsed and the code is interpreted in real-time. Languages like C++ and Fortran are faster because they are compiled ahead of time into the computer's native language. The advantages of parsing in real time are greater platform independence, robustness, and easier debugging. The disadvantages are a significant loss in speed, increased overhead, and limited low-level control. To compensate the speed loss, HLNO languages offer means to help speed up code, such as:

- To vectorize computations by replacing parallel operations with vector operations.
- To minimise function calls. Every time a function is called, a HNLO language incurs some overhead to find and parse the file and to create a local workspace for the function's local variables.
- To avoid frequent reallocations of matrices. If a matrix is resized repeatedly, like within a loop, this overhead becomes noticeable.

The SIVIA algorithm relies upon a *while loop* that iterates and at each iteration evaluates the *inclusion test* and the *Bisect* functions until the list $\mathcal{L}$ becomes empty. For the stated example, the algorithm carries out $527,299$ iterations. Therefore, for an

efficient implementation of the SIVIA algorithm in a HNLO programming language, a paradigm shift is required.

3 Vector Implementation of SIVIA

HLNO programming languages, such as Matlab, are well-known for being efficient when executing vector and matrix computations. With this idea in mind, we present a novel implementation of SIVIA, referred to as VSIVIA, that aims to minimise the use of explicit loops and function calls in the code and favours the use of vector computations, which at the same time minimises the function calls. The idea behind VSIVIA is to evaluate all the boxes of the list $\mathcal{L}$ in a vector way instead of processing them one by one, as it is done in the classic implementation. The motivation behind this strategy is based on the assumption that HLNO languages, and in particular Matlab, are computationally efficient for matrix and vector operations . Therefore, the *inclusion test* $[t]$, *Width* and *Bisect* functions from Algorithm (1) need to be extended to their vector form. Let the vector *inclusion test* $[\mathbf{t}](\mathcal{L})$ be defined as:

$$[\mathbf{t}](\mathcal{L}) = \begin{cases} \boldsymbol{in} & \textbf{if } [\mathbf{f}](\mathcal{L}) \subset \mathbb{Y}, \\ \boldsymbol{out} & \textbf{if } [\mathbf{f}](\mathcal{L}) \cap \mathbb{Y} = \emptyset, \\ \boldsymbol{undecided} & = \neg\textbf{in} \wedge \neg\textbf{out}, \end{cases} \tag{8}$$

where $[\mathbf{f}](\mathcal{L})$ is a vector inclusion function from $\mathbb{IR}^{n\times m} \mapsto \mathbb{IR}^{p\times m}$, m is the number of interval boxes in $\mathcal{L}$; and **in**, **out** and **undecided** are vectors of boolean variables of the same size as $\mathcal{L}$. The implementation of the VSIVIA algorithm is shown in Algorithm (2), where $\textbf{Width}(\mathcal{L})$ returns a vector of doubles of the same size as $\mathcal{L}$ corresponding to the width of largest intervals of each one of the boxes contained in $\mathcal{L}$; and $\textbf{Bisect}(\mathcal{L})$ bisects all the boxes contained in $\mathcal{L}$ by their larger (relative to the initial box $[\mathbf{x_0}]$) interval dimension and returns the corresponding list of boxes which is double the size of $\mathcal{L}$. Note that *logical indexing* [11] is used to point to the interval boxes that need to be considered from the corresponding vector.

Algorithm 2 Vector implementation of SIVIA algorithm (VSIVIA) algorithm

Require: • $\mathbb{Y} \subset \mathbb{R}^p$ • $[\mathbf{x_0}] \in \mathbb{IR}^n$ • $[\mathbf{f}] : \mathbb{IR}^n \to \mathbb{IR}^p$ • $\epsilon > 0$
Ensure: • $\mathcal{S}$, $\mathcal{N}$ and $\mathcal{E}$ such as: • $\mathcal{S} \subset (\mathbb{X} \cap [\mathbf{x_0}]) \subset \mathcal{S} \cup \mathcal{E}$ • $\mathcal{N} \cap \mathbb{X} = \emptyset$
• $width([\mathbf{x}]) < \epsilon$ $(\forall [\mathbf{x}] \in \mathcal{E})$

1: **function** VSIVIA([**f**], $\mathbb{Y}$, [$\mathbf{x_0}$], ϵ)
2: $\mathcal{S} \leftarrow \mathcal{N} \leftarrow \mathcal{E} \leftarrow \emptyset$
3: $\mathcal{L} \leftarrow \{[\mathbf{x_0}]\}$
4: **while** $\mathcal{L} \neq \emptyset$ **do**
5: **in** $\leftarrow$ ([**f**]($\mathcal{L}$) $\subset \mathbb{Y}$) ▷ **in**: Vector of booleans same size as $\mathcal{L}$ (logical index)
6: **out** $\leftarrow$ ([**f**]($\mathcal{L}$) $\cap \mathbb{Y} = \emptyset$) ▷ **out**: Vector of booleans same size as $\mathcal{L}$ (logical index)
7: $push(\mathcal{S}, \mathcal{L}(\textbf{in}))$ ▷ $push$: adds elements to a list
8: $push(\mathcal{N}, \mathcal{L}(\textbf{out}))$
9: $\mathcal{U} \leftarrow \mathcal{L}(\neg\textbf{in} \wedge \neg\textbf{out})$ ▷ $\mathcal{U}$: Undecided boxes
10: **eps** $\leftarrow$ (**Width**($\mathcal{U}$) $< \epsilon$) ▷ **Width**: returns widths of largest intervals dimensions.
11: $push(\mathcal{E}, \mathcal{U}(\textbf{eps}))$ ▷ **eps**: logical index
12: $\mathcal{L} \leftarrow$ **Bisect**($\mathcal{U}(\neg$**eps**$)$) ▷ **Bisect**: bisects each box in $\mathcal{U}$, returns list twice as long
13: **end while**
14: **return** $(\mathcal{S}, \mathcal{N}, \mathcal{E})$
15: **end function**

4 Using Interval Contractors within VSIVIA

Interval constraint propagation (ICP) is an approach that combines constraint propagation techniques [10] from artificial intelligence and interval analysis [12] to solve constraint satisfaction problems (CSPs) on intervals [2]. The main idea behind ICP consists of removing these parts of an a prior feasible interval domain (interval box) that are are not consistent with the set of involved constraints in a CSP. ICP can be used within the SIVIA algorithm to reduce its computational complexity by reducing the number of boxes to be bisected [6] and potentially reducing its computation time.

4.1 Interval Contractors

A degenerate box made with a single point $\mathbf{x}$ will be denoted by $\{\mathbf{x}\}$ or simply by $\mathbf{x}$. The function $\mathcal{C} : \mathbb{IR}^n \to \mathbb{IR}^n$ is an interval contractor if:

$$\begin{array}{lll} (i) & \forall [\mathbf{x}] \in \mathbb{IR}, \mathcal{C}([\mathbf{x}]) \subset [\mathbf{x}] & (contraction) \\ (ii) & \forall \mathbf{x} \in [\mathbf{x}], (\mathcal{C}(\{\mathbf{x}\}) = \{\mathbf{x}\}) \Rightarrow (\{\mathbf{x}\} \in \mathcal{C}([\mathbf{x}])) & (consistency) \\ (iii) & \mathcal{C}(\{\mathbf{x}\}) = \emptyset \Rightarrow (\exists\, \epsilon > 0, \forall \mathbf{x} \subset B(\mathbf{x}, \epsilon), \mathcal{C}(\{\mathbf{x}\}) = \emptyset) & (weak\ continuity) \end{array} \quad (9)$$

where $B(\mathbf{x}, \epsilon)$ is the ball with center $\mathbf{x}$ and radius ϵ. A box $[\mathbf{x}]$ is said to be insensitive to $\mathcal{C}$ if $\mathcal{C}([\mathbf{x}]) = [\mathbf{x}]$. From Property (i), boxes can only be contracted. From Property (ii), an insensitive point x is never removed by $\mathcal{C}$. From Property (iii), the set of all insensitive $\mathbf{x}$ is closed.

Different consistency techniques, such as *2B-consistency*, *3B-consistency*, *Box-consistency* and *Hull-consistency* [1], can be used to build an interval contractor, and its suitability is usually problem-dependent. In this work, a *forward-backward* contractor [6] (*2B-consistency*) has been considered for its simplicity of implementation.

In order not to indefinitely contract a box, the efficiency of a contraction $[\mathbf{x_c}] = \mathcal{C}([\mathbf{x}])$ is defined as

$$\eta = \min_{i \in [1,n]} \frac{width([x_{c_i}])}{width([x_i])}. \quad (10)$$

Algorithm 3 Forward-Backward Contractor

Require: • $\mathbb{Y} \subset \mathbb{IR}^p$ • $[\mathbf{x_0}] \in \mathbb{IR}^n$ • $[f] : \mathbb{IR}^n \to \mathbb{IR}^p$ • $[f^{-1}] : \mathbb{IR}^p \to \mathbb{IR}^n$ • $\zeta > 0$
Ensure: $[\mathbf{x}] \subseteq [\mathbf{x_0}]$

```
1: function FORWARD_BACKWARD_CONTRACTOR(𝕐, [x0], [f], [f^-1], ζ)
2:     [x] ← [−∞, +∞]^n
3:     while contraction([x0], [x]) < ζ do        ▷ contraction([x0], [x]) = min_{i∈[1,n]} width([x_i]) / width([x_0i])
4:         [x] ← [x0]
5:         𝕐 ← 𝕐 ∩ [f]([x])                        ▷ Forward propagation
6:         [x] ← [x0] ∩ [f^-1](𝕐)                  ▷ Backward propagation
7:     end while
8:     return [x]
9: end function
```

From this definition of efficiency, a tolerance threshold ζ is defined in such a way that the contraction process is stopped as soon as η becomes greater than ζ. Listing (3) shows an implementation of a *forward-backward* contractor. Such a contractor can

be used easily within the classic implementation of SIVIA algorithm on every retrieved box from list $\mathcal{L}$ (line 5 of Algorithm (1)). Note that if the resulting contraction of a box is an empty set, the contraction over this box stops, and it is considered as a non-solution box, that is therefore added to the set $\mathcal{N}$.

4.2 Vector Forward-Backward Contractor

To use an interval contractor efficiently within VSIVIA, a vector interval contractor is required. Therefore, before being bisected and after being evaluated by the inclusion test (see Algorithm (2)), all the interval boxes from $\mathcal{U}(\neg\mathbf{eps})$ are contracted in a vector way. Listing (4) describes a vector implementation of the proposed *forward-backward* interval contractor. *Logical indexing* is used to point to the interval boxes that need to be further contracted. Also note that the resulting boxes that are reduced to an empty set by the contractor need to be added to the list $\mathcal{N}$ of Algorithm (2).

Algorithm 4 vector Forward-Backward Contractor

Require: • $\mathbb{Y} \subset \mathbb{IR}^{p\times m}$ • $[\mathbf{f}^{-1}] : \mathbb{IR}^{p\times m} \to \mathbb{IR}^{n\times m}$ • $\mathcal{L}_0 \in \mathbb{IR}^{n\times m}$ • $\zeta > 0$ • $[\mathbf{f}] : \mathbb{IR}^{n\times m} \to \mathbb{IR}^{p\times m}$

Ensure: $\mathcal{L} \subseteq \mathcal{L}_0$

1: **function** VECTOR-FRWD-BKWRD-CONTRACTOR($\mathbb{Y}$, $\mathcal{L}_0$, $[\mathbf{f}]$, $[\mathbf{f}^{-1}]$, ζ)
2: $\quad\mathcal{L} \leftarrow [-\infty, +\infty]^{n\times m}$
3: $\quad$**ctrn** $\leftarrow [true]^{m}$
4: $\quad$**while ctrn** $\neq false$ **do** ▷ **ctrn**: logical index
5: $\quad\quad\mathcal{L}(\mathbf{ctrn}) \leftarrow \mathcal{L}_0(\mathbf{ctrn})$
6: $\quad\quad\mathbb{Y}(\mathbf{ctrn}) \leftarrow \mathbb{Y}(\mathbf{ctrn}) \cap [\mathbf{f}](\mathcal{L}(\mathbf{ctrn}))$ ▷ vector forward propagation
7: $\quad\quad\mathcal{L}(\mathbf{ctrn}) \leftarrow \mathcal{L}_0(\mathbf{ctrn}) \cap [\mathbf{f}^{-1}](\mathbb{Y}(\mathbf{ctrn}))$ ▷ vector backward propagation
8: $\quad\quad$**ctrn** $\leftarrow (contraction(\mathcal{L}_0, \mathcal{L}) < \zeta)$ ▷ $contraction(\mathcal{L}_0, \mathcal{L}) = \left\{ \min_{i\in[1,n]} \frac{width([\mathcal{L}(\mathbf{k})_\mathbf{i}])}{width([\mathcal{L}_\mathbf{0}(\mathbf{k})_\mathbf{i}])}, k \in [1, m] \right\}$
9: $\quad$**end while**
10: $\quad$**return** $\mathcal{L}$
11: **end function**

5 Matlab Implementation

To evaluate $[\mathbf{f}](\mathcal{L})$, a vector interval arithmetic was implemented in Matlab. Although the Intlab library [13] could have been used for this purpose, the implemented arithmetic also incorporates the so-called extended interval arithmetic, or Kaucher arithmetic [9], which allows using the theory of Modal Interval Analysis [3]. Another reason for implementing a new vector interval arithmetic was to be able to implement a vector interval contractor. Following the guidelines stated in Section 2 for writing efficient code in a HLNO language, the implemented interval arithmetic does not use any explicit *for loops* in the Matlab code. Instead of explicit *for loops*, Matlab *logical indexing* and the *bsxfun* built-in function were employed [11]. To facilitate the writing of interval arithmetic expressions, a Matlab class representing an interval vector was implemented. The implemented class has two attributes (*lower* and *upper*) consisting of two vectors of doubles, which represent the lower and upper bounds of an interval vector. This class overloads all the arithmetic operators to operate easily with interval vectors. The *forward-backward* interval contractor does not need to be introduced by the user because it is automatically generated by the code. A Matlab implementation

of the VSIVIA algorithm including the *forward-backward* interval contractor is freely distributed. The source code, a technical documentation, a user manual and several examples including the ones presented in this paper can be downloaded from [5].

6 Results

To test the performance of of the proposed vector implementation of SIVIA (VSIVIA), the following problem has been considered. Let the implicit equation in Cartesian coordinates for a torus radially symmetric about the z-axis be described by

$$\left(R - \sqrt{x^2 + y^2}\right)^2 + z^2 = r^2, \tag{11}$$

where R is the distance from the center of the tube to the center of the torus and r is the radius of the tube.

Now, consider the set inversion problem that characterises the volume of a torus with $R = 5$ and $r^2 \in [1, 2]$

$$\mathcal{P} := \left\{ \begin{array}{ccc} f & : & (x, y, z) \mapsto \left(5 - \sqrt{x^2 + y^2}\right)^2 + z^2 \\ [\mathbf{x_0}] & = & [-8, 8] \times [-8, 8] \times [-8, 8] \\ \mathbb{Y} & = & [1, 2] \end{array} \right. \tag{12}$$

Table (1), shows the computations times (Intel Core 2 Duo E8500 (3.16 GHz) - 4 GB RAM) required for solving the example stated by Equation (7) ($\epsilon = 0.01$) by the following implementations of the SIVIA algorithm: the classic SIVIA in Matlab; the vector SIVIA (VSIVIA) in Matlab; the classic SIVIA in C++; VSIVIA in C++ using the Armadillo linear algebra library [14]; and VSIVIA in Matlab with interval contractor ($\zeta = 0.5$). From Table (1), the following conclusions can be extracted. The vector implementation of SIVIA in Matlab (VSIVIA) brings a significant improvement in terms of computation time with respect to the classic SIVIA implementation in Matlab (classic (Matlab): 336s vs. vector (Matlab): 0.25s)). The VSIVIA implementation in Matlab is comparable to the C++ implementation (classic (C++): 0.25s vs. vector (Matlab): 0.24s). The VSIVIA implementation in C++ using the Armadillo linear algebra library does not improve the computation time with respect to the classical implementation of SIVIA in C++ (classic (C++): 0.24 vs. vector (C++): 0.25 s). The use of an interval contractor within VSIVIA does not bring an improvement in terms of computation time, although the total number of boxes being processed by the algorithm is reduced (without contractor: $198,591$ boxes vs. with contractor: $172,815$ boxes), because the computation time spent by the contractor is bigger that the time saved by VSIVIA. However, for certain problems, this reduction in the number of processed boxes can be translated into a reduction on computation time. Finally, although this test was not carried out, if the rounding mode is considered in the interval arithmetic library to provide guaranteed numerical results, the vector implementation can improve the computation time with respect to the classic implementation because of the reduction on the number of times the rounding mode needs to be changed (SIVIA: $1,390,137$ times vs. VSIVIA: 161 times). Note that this advantage is valid for any programming language.

The Matlab code needed to implement the example stated by Equation (7) is provided within the distributed Matlab package [5].

Language	**Algorithm**	**Computation time**	**# processed boxes**
Matlab	SIVIA	336 s	198,591
Matlab	VSIVIA	0.25 s	198,591
C++	SIVIA	0.24 s	198,591
C++	VSIVIA	0.25 s	198,591
Matlab	VSIVIA+Contractor	0.97 s	172,815

Table 1: Computation time (Intel Core 2 Duo E8500 (3.16 GHz) - 4 GB RAM) for solving the example stated by Equation (7) by different implementations of SIVIA and VSIVIA in Matlab and C++ ($\epsilon = 0.01$ and $\zeta = 0.5$).

7 Applications

In this section, two applications are introduced in order to show the scope of the proposed SIVIA implementation. The corresponding Matlab code needed to implement such applications is provided within the distributed Matlab package [5].

7.1 Drug Concentration

Consider the problem of a bolus intravenous injection of a drug into a patient [4]. The solution of the ODE system that models the pharmacokinetics of the drug distribution between the central compartment (blood) and the peripheral compartment (tissue) and the elimination from the central compartment is:

$$y(t) = a \cdot e^{-\alpha \cdot t} + b \cdot e^{-\beta \cdot t}. \tag{13}$$

This expression depends on four parameters a, b, α, β, which can be used to express the distribution and elimination rates (k_{el} is the elimination rate, and k_{cp}, k_{pc} are distribution rates) as follows:

$$k_{pc} = \frac{\alpha\beta + b\alpha}{a+b}, \quad k_{el} = \frac{\alpha\beta}{k_{pc}}, \quad k_{cp} = \alpha + \beta - k_{pc} - k_{el}. \tag{14}$$

For a bolus intravenous injection of 800 mg of a drug into a patient, the data for the concentration in the central compartment over a period of time are given in Table (2).

t	0.1	0.25	0.5	0.75	1	1.5	2	2.5	3	4	6	8	10	12
$\tilde{y}$	16.1	14.3	12.0	10.3	9.0	7.2	6.1	5.2	4.6	3.7	2.5	1.7	1.18	0.81

Table 2: Drug concentration data over time for a bolus intravenous injection of 800 mg of a drug into a patient.

Considering a relative $\pm 5\%$ error in the data $\tilde{y}$, we aim at identifying the set of parameters (a, α, b, β) such that the model expressed by Equation (13) is consistent with the data set from Table (2). Such a problem can be stated as the following set inversion problem:

$$\mathbb{X} = \left\{ (a,b,\alpha,\beta) \in ([a],[b],[\alpha],[\beta]) \mid \hat{y}(t) = a \cdot e^{-\alpha \cdot t} + b \cdot e^{-\beta \cdot t} \right\}, \tag{15}$$

which can be solved by the SIVIA algorithm.

Assuming that $(a, \alpha, b, \beta) \in [1, 100] \times [0, 10] \times [1, 100] \times [0, 1]$, with a $\epsilon = 10^{-3}$ (relative), VSIVIA solves the problem in 8.6 seconds (Intel Core 2 Duo E8500 (3.16 Ghz) - 4 GB RAM) with a total of 465,629 boxes processed. The corresponding inner and outer approximation is given by the following bounding boxes:

$$Inner(\bigcup_{[\mathbf{x}]\in\mathcal{S}} [\mathbf{x}]) = [8.15, 11.16] \times [0.95, 1.82] \times [6.70, 8.55] \times [0.17, 0.20], \quad (16)$$

$$Outer(\bigcup_{[\mathbf{x}]\in\mathcal{E}} [\mathbf{x}]) = [5.44, 11.64] \times [0.14, 2.06] \times [6.12, 11.54] \times [0.16, 1.00]. \quad (17)$$

If a *forward-backward* contractor with a relative $\zeta = 0.5$ is used within VSIVIA, the inner and outer approximations expressed by Equation (18) are obtained in 21.4 seconds and a total of 414,421 boxes processed. The corresponding inner and outer approximation is given by the following bounding boxes:

$$Inner(\bigcup_{[\mathbf{x}]\in\mathcal{S}} [\mathbf{x}]) = [8.05, 11.24] \times [0.94, 1.86] \times [6.68, 8.61] \times [0.17, 0.21], \quad (18)$$

$$Outer(\bigcup_{[\mathbf{x}]\in\mathcal{E}} [\mathbf{x}]) = [6.36, 11.49] \times [0.16, 1.98] \times [6.35, 10.67] \times [0.16, 1.00]. \quad (19)$$

The use of an interval contractor does not represent any improvement in the computation time, but it reduces the number of processed boxes by 19%. Also note that the approximation of the solution set obtained with the contractor is slightly better than the one obtained without the contractor.

7.2 U.S. Census

Consider the census problem [4] that models the time variation of a population with limited growth, taking into account overcrowding and depletion of resources. It assumes that the relative growth rate is not constant: the growth rate decreases as the population approaches some fixed upper bound called the carrying capacity. The mathematical model is the logistic equation

$$\frac{1}{y} \cdot \frac{dy}{dt} = k \cdot (L - y), \;\; y(0) = y_0, \quad (20)$$

where y is the population, and L is the carrying capacity. The solution to this equation, obtained through the separation of variables method, is:

$$y = \frac{L \cdot y_0}{y_0 + (L - y_0) \cdot e^{-r \cdot t}}, \quad (21)$$

Given census data over a time period, one use of the model is to estimate the carrying capacity L. Table (3) gives the U.S. Census over the years 1790 to 1900, normalized to 0. Each data is subject to an error of 1.

We aim at identifying the set of parameters L, r and y_0 such that the model expressed by Equation (21) is consistent with the data set from Table (3). Assuming that $(L, r, y_0) \in [1, 1000] \times [0.00, 10] \times [0.1, 100]$, with a $\epsilon = 10^{-4}$ (relative), VSIVIA solves this problem in 26 seconds and a total of 1,741,497 boxes processed. The corresponding inner and outer approximation is given by the following bounding boxes:

t	0	10	20	30	40	50	
$\tilde{y}$	3.929	5.308	7.239	9.638	12.866	17.069	

t	60	70	80	90	100	110	120
$\tilde{y}$	23.191	31.433	39.818	50.155	62.947	75.994	91.9

Table 3: U.S. Census data over time

$$Inner(\bigcup_{[\mathbf{x}]\in\mathcal{S}}[\mathbf{x}]) = [169.9, 253.8] \times [0.029, 0.034] \times [3.53, 4.52], \tag{22}$$

$$Outer(\bigcup_{[\mathbf{x}]\in\mathcal{E}}[\mathbf{x}]) = [163.6, 265.8] \times [0.028, 0.034] \times [3.38, 4.58]. \tag{23}$$

If a *forward-backward* contractor with a $\zeta = 0.5$ is used within VSIVIA, the inner and outer approximations expressed by Equation (18) is obtained in 48 seconds and a total of $1,081,693$ boxes processed. The corresponding inner and outer approximation is given by the following bounding boxes:

$$Inner(\bigcup_{[\mathbf{x}]\in\mathcal{S}}[\mathbf{x}]) = [170.2, 253.7] \times [0.028, 0.034] \times [3.54, 4.51] \tag{24}$$

$$Outer(\bigcup_{[\mathbf{x}]\in\mathcal{E}}[\mathbf{x}]) = [164.7, 262.7] \times [0.028, 0.034] \times [3.41, 4.56]. \tag{25}$$

An interval contractor does not represent any improvement in the computation time, but it reduces the number of processed boxes by 38%. Also note the approximation of the solution set obtained with the contractor is slightly better than the one obtained without the contractor.

8 Conclusions

In this paper, we have demonstrated that a novel vector implementation of the Set Inversion via Interval Analysis (SIVIA) algorithm in a high-level, numerically oriented programming (HLNO) language implementation, such as Matlab, can be as efficient as a C++ implementation. We also have proven that the use of interval contractors, used to reduce the complexity of SIVIA, can be integrated easily within the presented vector implementation of SIVIA (VSIVIA). However, in terms of computation time, the use of interval contractors was not proven to be effective for the treated problems, although the number of bisections carried of by VSIVIA+Contractor were less than VSIVIA alone. It is important to remark that the enclosures of the solution set achieved by the VSIVIA+Contractor algorithm were tighter than the achieved ones by the VSIVIA algorithm alone. Although not presented in this paper, the proposed vector implementation has the potential to reduce the computational cost of changing the rounding model when a guaranteed interval arithmetic is considered. This is because the VSIVIA algorithm significantly reduces the number of times the rounding modes needs to be changed.

Due to the popularity of HLNO programming languages in many scientific and engineering communities, the proposed implementation will facilitate and promote the use of SIVIA algorithm in these communities. Even if the current implementation has been implemented in Matlab, it could be ported easily to any other HLNO programming language such as Scilab or Octave. The source code of a Matlab implementation, as well as a technical documentation, a user manual and several examples are freely distributed.

The VSIVIA+Contractor algorithm is being adapted to solve constrained global optimisations problems, and the use of local optimisation algorithms is being considered in order to speed up the computations without losing global optimality. The utilisation of a guaranteed interval arithmetic library, such as Intlab, is being considered for a future release of the Matlab implementation.

References

[1] F. Benhamou, F. Goualard, L. Granvilliers, and J.F. Puget. Revising hull and box consistency. *In Proc. ICLP, Conf. on Logic Programming*, pages 230–244, 1999.

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

[3] E. Gardenes, M.A. Sainz, L. Jorba, R. Calm, R. Estela, H. Mielgo, and A. Trepat. Modal intervals. *Reliable Computing*, 7(2):77–111, 2001.

[4] L. Granvilliers, J. Cruz, and P. Barahona. Parameter estimation using interval computations. *SIAM Journal on Scientific Computing*, 26(2):591–612, 2004.

[5] P. Herrero. *Pau Herrero's Research Page*. Imperial College London, `http://www3.imperial.ac.uk/people/p.herrero-vinias/research`, Online; accessed 8-October-2012.

[6] 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.

[7] 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.

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

[9] E Kaucher. Interval analysis in the extended interval space IR. *Computing*, Supplementum 2:33–49, 1980.

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

[11] Mathworks. *Matlab Information*. Mathworks, `http://www.mathworks.com/`, Online; accessed 13-July-2012.

[12] R. E. Moore and C. T. Yang. Interval analysis I. Technical Document LMSD-285875, Lockheed Missiles and Space Division, Sunnyvale, CA, USA, 1959.

[13] S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `http://www.ti3.tu-harburg.de/rump/`.

[14] C. Sanderson. *Armadillo Web Page.* NICTA, Australia, `http://arma.sourceforge.net/`, Online; accessed 13-July-2012.

[15] S. Tornil-Sin, V. Puig, and T. Escobet. Set computations with subpavings in MATLAB: the SCS toolbox. In *IEEE International Symposium on Computer-Aided Control System Design - Part of 2010 IEEE Multi-Conference on Systems and Control)*, pages 1403–1408, Yokohama, Japan, 2010.