

# Adaptive sampling-based approximation of the sign of multivariate real-valued functions

Nicolas Perrin, Olivier Stasse, Florent Lamiraux and Eiichi Yoshida

**Abstract**—In this technical report, we present an algorithm which approximates the sign of multivariate real-valued functions through adaptive sampling, and demonstrate a convergence result. The algorithm was specifically designed for real-time robotics applications where quick evaluations are often needed.

**Index Terms**—approximation algorithm, adaptive sampling, quadratic programming.

## I. INTRODUCING THE ALGORITHM

Our algorithm is intended to approximate (or learn) the sign of  $C$ , a continuous mapping from a bounded hyper-rectangle  $B_0 \subset \mathbb{R}^m$  to  $\mathbb{R}$  (bounded hyper-rectangles will be called ‘boxes’ from now on). We suppose that the zero level surface  $\{\mathbf{x} \in B_0 | C(\mathbf{x}) = 0\}$  (the “frontier”) is of Lebesgue measure zero.

$C$  is typically an important function that needs to be frequently called by a real-time application, and decisions are made depending on the sign of the numbers returned by  $C$ . Yet,  $C$  is originally implemented with a quite time consuming algorithm, and the current computation cost makes it unsuitable for real-time applications. Therefore we would like to approximate the sign of  $C(\mathbf{x})$  offline in order to then be able to quickly guess it online, without having to actually go through any lengthy computation. This is the kind of context in which the algorithm presented here is most useful.

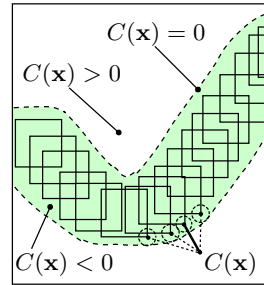
For example in robotics or computer graphics, the algorithm can be used in order to build implicit swept volume approximations that can be evaluated very fast (in that case the result obtained is a bit similar to an Adaptive Distance Field, see [1]). Indeed, if collisions have to be checked over and over for the same trajectory of an object in different environments, it might be very useful to approximate the sign of the corresponding distance field (i.e. the distance to the volume swept by the object along the trajectory). If the result of the approximation is then stored in a very efficient data structure, a lot of time can subsequently be saved during the collision checks. Fig. 1 shows the corresponding mapping  $C$  for a trajectory followed by a cube; it also displays the result of our approximation algorithm on this case example.

Back to the general case, the goal of the algorithm is to learn the sign of  $C$  through adaptive sampling. The algorithm is a slight variant of the one introduced in [4], and is articulated around three principles:

N. Perrin is with CNRS/LAAS, Universit de Toulouse UPS, INSA, INP, ISAE 7 avenue du colonel Roche, F-31077 Toulouse, France, and CNRS-AIST Joint Robotics Laboratory, UMI3218/CRT, Tsukuba, Japan.

O. Stasse and E. Yoshida are with CNRS-AIST Joint Robotics Laboratory, UMI3218/CRT, Tsukuba, Japan.

F. Lamiraux is with CNRS/LAAS, Universit de Toulouse UPS, INSA, INP, ISAE 7 avenue du colonel Roche, F-31077 Toulouse, France.



On the left, a 2D representation of a cube moving along a discretized trajectory. We denote its successive configurations by  $c_0, c_1, \dots, c_q$ . For a point  $\mathbf{x}$  of the Euclidean space,  $C(\mathbf{x})$  is defined as the minimum distance from  $\mathbf{x}$  to any configuration of the cube, minus a fixed margin  $\tau$ . The margin is important to avoid errors due to the discretization, and besides, it makes the level set  $\{\mathbf{x} \in \mathbb{R}^3 | C(\mathbf{x}) = 0\}$  smoother, thus easier to approximate.

The 2D plot on the bottom-left shows how the approximation algorithm recursively divides the Euclidean space into small boxes in order to adaptively approximate the surface  $C(\mathbf{x}) = 0$ . The approximated surface defines an approximation of the volume swept by the cube.

A view of this swept volume approximation is displayed on the 3D plot on the bottom right.

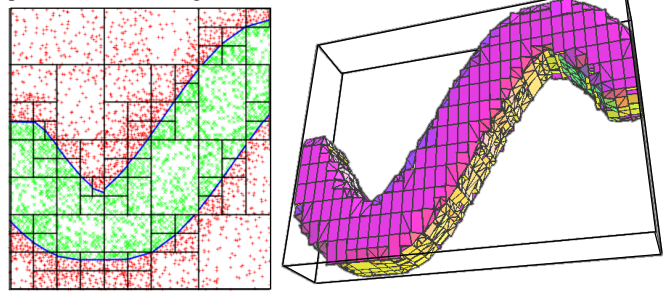


Fig. 1. Case example: implicit approximation of the volume swept by a cube.

- 1) Recursive Stratified Sampling: the initial input space  $B_0$  is recursively partitioned into small boxes (a tree structure keeps the trace of all splittings; the current partition is formed by the “leaf-boxes”). Basically, a box will be split when the local approximation process fails, which can happen only when both positive and negative samples are contained in box. Thus, the boxes will be split a lot and become small near the frontier, i.e. the zone where  $C$  changes its sign. This property will enable an effective implementation of adaptive sampling focusing on the frontier.
- 2) Farthest Point-like Sampling in two different scales:
  - While selecting a leaf-box for sampling: we roughly estimate for each leaf-box our “confidence in the sign” of the local approximation, and select among boxes of lowest confidence.
  - While sampling inside a leaf-box: since there will be a limited number of samples inside a leaf-box, we can use a naive technique for an approached farthest point-like sampling.
- 3) Solving small Quadratic Programming problems to ob-

tain local approximations with correct sign on the training data.

Let us call  $f$  the result of the approximation.  $f$  is constantly updated during the execution of the algorithm, and like  $C$ , it is a function from  $B_0 \subset \mathbb{R}^m$  to  $\mathbb{R}$ . We denote by  $f|_B$  the restriction of  $f$  to a box  $B$  contained in  $B_0$ . Below, Algorithm 1 describes the algorithm structure, and leaves just four questions, which we answer respectively in the four next sections: “how to pick a new leaf-box?”, “how to get new samples?”, “how to split the boxes?”, and “how to locally approximate?”.

## II. HOW TO PICK A NEW LEAF-BOX

It is very important to find a good method to pick new leaf-boxes, because this choice is the core of the adaptiveness of the sampling. We want to focus on the boxes where we know that  $C$  changes its sign, but without forgetting to sample on other boxes where a change of sign might have to be discovered yet. It is very difficult to decide at which rate one should sample near the frontier and at which rate one should sample in regions where only positive (or only negative) samples have been seen so far.

In [4], the mapping  $C$  was assumed to be  $K$ -Lipschitz for some  $K \in \mathbb{R}$ , and a formula for the “confidence in the sign” of a local approximation was defined, with the aim of picking only leaf-boxes with lowest confidence. The formula used works well in general, but causes the algorithm to fail to converge with some particular mappings  $C$ . In the variant presented here, we use a different heuristic which this time comes with a proof of convergence (section VI).

The set of leaf-boxes – let us call it *Set\_of\_Leaves* – is divided in three disjoint subsets: the set of positive leaf-boxes (containing only samples on which  $C$  is positive), the set of negative leaf-boxes (containing only samples on which  $C$  is negative), and the set of “frontier” leaf-boxes (containing either no sample or both positive and negative samples). We denote these subsets by *Set\_of\_PositiveLeaves*, *Set\_of\_NegativeLeaves*, and *Set\_of\_FrontierLeaves*. We also denote by *max\_value* the maximum absolute value of  $C$  seen on the samples collected so far. For a leaf-box  $B$  containing the samples  $(s_1, C(s_1)), \dots, (s_k, C(s_k))$ , with  $k \geq 1$ , we define  $co(B)$ , which is inspired by the value  $conf(B)$  used in [4] (the “confidence in the sign”), but leads to better convergence properties. The definition is divided into two cases:

- 1) If  $B$  is a leaf-box with at least one sample on it,  $co(B)$  is defined as follows (the measure of Lebesgue, or volume of  $B$  is simply the product of its  $m$  lengths):

$$co(B) = \frac{k}{\text{volume of } B} + \frac{1}{\text{max\_value}} \times \frac{\sum_{i=1}^k |C(s_i)|}{\text{volume of } B} \quad (1)$$

- 2) If  $B$  is a leaf-box with no sample, it has necessarily just been created, and in that case either  $B$  is the input space,  $B_0$ , and we pose  $co(B) = (\text{volume of } B_0)^{-1}$ , or  $B$  has been created by splitting its parent box  $B'$ , and we pose:

$$co(B) = \frac{co(B')}{2}, \quad (2)$$

where  $co(B')$  is the value just before the split.

The important properties of  $co(B)$  are that first, it increases when the density of samples (i.e.  $\frac{k}{\text{volume of } B}$ ) on the leaf-box increases, and second,  $co(B)$  has a component which depends on the values of  $C$  on the samples: this helps to drive the sampling towards regions where  $C$  takes relatively small values. Indeed, considering only the density of samples is not enough: if  $B_1$  is a box which contains 10 samples on which the value of  $C$  is about 5, and  $B_2$  a box of same size with 9 samples on which the value of  $C$  is about 1, even though the density of samples is higher on  $B_1$ , it seems natural to sample first on  $B_2$ , because intuitively  $C$  has more chances to change its sign on  $B_2$ .  $co(B)$  gives us an arbitrary rule to know when to stop sampling on  $B_2$  and start sampling on  $B_1$ . The third interesting property comes from the use of *max\_value* and is that if we simply multiply  $C$  by a non-zero constant, the behaviour of the sampling would remain unchanged: without randomness it would lead exactly to the same samples and the same leaf-boxes. Finally, one of the main advantage of this definition of  $co(B)$  is that it is extremely easy to compute.

Algorithm 2 shows how  $co(B)$  is used to pick a new leaf-box. We see that by default, this algorithm gives the same importance to the two following tasks:

- 1) Select a leaf-box among *Set\_of\_FrontierLeaves*, i.e. try to make the approximation more precise on the known parts of the frontier of  $C$ .
- 2) Select a leaf-box among *Set\_of\_PositiveLeaves*  $\cup$  *Set\_of\_NegativeLeaves*, i.e. try to discover new parts of the frontier of  $C$ .

Two remarks:

- In some cases, it is better to customize the Algorithm 2. For example, when  $C$  is a distance minus a small threshold, the positive and negative regions are not symmetric and should not be treated so.  $C$  will mainly be equal to  $-\tau$  ( $\tau$  being the threshold) on the negative region, which gives few information; thus in that case it seems appropriate to focus more on the positive region than the negative one. Another example is when the frontier is known to have a simple shape: in that case it is arguably better to pick “frontier” leaf-boxes with a probability greater than 50%, thus boosting the focus on the frontier.
- Let us call selecting a leaf-box among *Set\_of\_FrontierLeaves* the task 1), and selecting a leaf-box among *Set\_of\_PositiveLeaves*  $\cup$  *Set\_of\_NegativeLeaves* the task 2). Looking for a good tradeoff between tasks 1) and 2) is very similar to some issues raised by the so-called multi-armed bandit problem, where a gambler is using a slot machine with multiple levers. When pulled, each lever provides a reward drawn from a distribution associated with that specific lever. The objective of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls. The crucial tradeoff the gambler faces at each trial is between “exploitation” of the lever that has the highest expected payoff and “exploration” to get more information about the expected payoffs of the

**Algorithm 1** The algorithm structure.

---

```

1:  $f \leftarrow$  the constant function 0.
2:  $Set\_of\_Leaves \leftarrow \{B_0\}$ 
3: while (1) do
4:   Pick (using Algorithm 2) a leaf-box  $B$  out of  $Set\_of\_Leaves$  (and remove it from  $Set\_of\_Leaves$ ).
5:   Get new samples in  $B$ .
6:   if there are only positive samples in  $B$  then
7:     Define  $f|_B$  as any positive constant on  $B$ , for example choosing the minimum value of  $C$  on the samples in  $B$ .
8:   else if there are only negative samples in  $B$  then
9:     Define  $f|_B$  as any negative constant on  $B$ , for example choosing the maximum value of  $C$  on the samples in  $B$ .
10:  else
11:     $Temporary\_Stack \leftarrow \{B\}$ 
12:    while  $Temporary\_Stack \neq \emptyset$  do
13:      Pop a leaf-box  $B_{current}$  out of  $Temporary\_Stack$ 
14:      Try to locally approximate  $C$  on  $B_{current}$ , i.e. try to define a new value of  $f|_{B_{current}}$ , the main constraint being that  $f|_{B_{current}}$  must have a correct sign on all the samples in  $B_{current}$ .
15:      if this attempt does not succeed then
16:        Split  $B_{current}$  into two son boxes of equal dimensions:  $B_{left}$  and  $B_{right}$ , and push them both in  $Temporary\_Stack$ .
17:      else
18:        Put  $B_{current}$  in  $Set\_of\_Leaves$ .
19:      end if
20:    end while
21:  end if
22: end while

```

---

other levers. In our problem, the task 1) can be seen as the "exploitation", and the task 2) as the "exploration". This similarity could be the starting point of a more theoretical foundation for the Algorithm 2, and indeed a few examples of adaptive sampling techniques based on Bayesian or frequentist analyses of the bandit problem exist in the litterature, such as in [3]. Yet in our case the payoff is not very clear, or at least not known in advance (it should quantify the improvement of the frontier approximation), and furthermore one of our priorities, computational efficiency, limits the range of possible methods.

### III. HOW TO GET NEW SAMPLES

Line 5 of Algorithm 1: new samples must be chosen inside the leaf-box  $B$ .

In the default version of the algorithm,  $N$  new samples are simply chosen independently and uniformly inside  $B$ ,  $N$  being decided by the user.

In a more advanced version, we can first draw more than  $N$  samples (e.g.  $10N$ ), and then select a group of samples that are far from each other and far from the samples already in  $B$ . This version can be chosen if the evaluation of  $C$  is much longer than the computation needed to obtain such samples.

In an even more advanced version, we allow the samples to be drawn also a bit outside of  $B$ . This technique has already been efficiently used in several adaptive algorithms, such as the one presented in [2] for example; it allows samples to spill into neighboring boxes and enables the adaptive sampling method to "crawl" along the frontier of  $C$ .

### IV. HOW TO SPLIT THE BOXES

We simply split the boxes cyclically. We give an order to the dimensions:  $dim. 1, dim. 2, \dots, dim. m$ . The initial box  $B_0$  is split in half orthogonally to  $dim. 1$ , the children boxes will be split orthogonally to  $dim. 2$ , then  $dim. 3$ , etc. When a box is split orthogonally to  $dim. m$ , its children boxes will be split orthogonally to  $dim. 1$ , and so on...

If the input space has 2 or 3 dimensions, specific implementations with quadrees or octrees can be advantageously used.

### V. HOW TO LOCALLY APPROXIMATE

Here is the context: a leaf-box  $B_{current}$  contains a finite number of samples, and we want to find a local approximation which has a correct sign on these samples, or return "fail".

The three first cases are simple: if there is no sample in  $B_{current}$ , we don't modify  $f|_{B_{current}}$ ; if there are only positive samples in  $B_{current}$ , we define  $f|_{B_{current}}$  as any positive constant on  $B_{current}$  (e.g. the minimum value of  $C$  on the samples in  $B_{current}$ ); if there are only negative samples in  $B_{current}$ , we define  $f|_{B_{current}}$  as any negative constant on  $B_{current}$  (e.g. the maximum value of  $C$  on the samples in  $B_{current}$ ).

In the last possible case,  $B_{current}$  contains both positive and negative samples. Again, the user has set a parameter  $N_{max} > N$ , and if the box contains more than  $N_{max}$  samples, we return "fail" (and the box will be split, which will lead to two smaller boxes with hopefully less samples, and local approximations will be attempted again on each of them). This parameter  $N_{max}$  should be chosen so that only a small portion

---

**Algorithm 2** The default algorithm for picking a new leaf-box.

---

```

1: Flip a coin.
2: if the flip is heads AND  $Set\_of\_FrontierLeaves$  is not empty then
3:   Choose a leaf-box  $B_{new} \in Set\_of\_FrontierLeaves$  s.t.  $co(B_{new}) = \min \{co(B) | B \in Set\_of\_FrontierLeaves\}$ .
4: else
5:   Flip another coin.
6: if the flip is heads AND  $Set\_of\_PositiveLeaves$  is not empty then
7:   Choose a leaf-box  $B_{new} \in Set\_of\_PositiveLeaves$  s.t.  $co(B_{new}) = \min \{co(B) | B \in Set\_of\_PositiveLeaves\}$ .
8: else
9:   Choose a leaf-box  $B_{new} \in Set\_of\_NegativeLeaves$  s.t.  $co(B_{new}) = \min \{co(B) | B \in Set\_of\_NegativeLeaves\}$ .
10: end if
11: end if

```

---

of the computation time is spent on local approximations (whose complexity depends on the number of samples); most of the computation time should be spent at evaluating  $C$ .

Here, we present a simple technique for local approximation based on Quadratic Programming. The convergence (Theorem VI.1) of the algorithm does not depend on local approximations, but good local approximations will speed up the convergence rate.

Let us call  $\{(s_1, C(s_1)), \dots, (s_l, C(s_l))\}$  the samples in  $B_{current}$ .

In the technique presented here,  $f|_{B_{current}}$  is being searched among the elements of a finite dimension vector space chosen by the user (this vector space must contain constant functions). Let us describe the basic case of the affine functions which take the form:

$$g(\mathbf{x}) = g(x_1, x_2, \dots, x_m) = \langle w, (x_1, x_2, \dots, x_m, 1) \rangle, \quad (3)$$

with  $w \in \mathbb{R}^{m+1}$  and where  $\langle \cdot, \cdot \rangle$  denotes the dot product on  $\mathbb{R}^{m+1}$ . We solve the following Quadratic Programming problem:

$$\begin{aligned} & \text{minimize } \sum_i (g(s_i) - C(s_i))^2 \\ & = \langle w, \mathbf{M}w \rangle + \langle d, w \rangle + \sum_i C(s_i)^2 \\ & \text{subject to } \begin{cases} \forall i \mid C(s_i) > 0, & g(s_i) > 0 \\ \forall i \mid C(s_i) \leq 0, & g(s_i) < 0 \end{cases} \end{aligned} \quad (4)$$

where  $\mathbf{M}$  is a symmetric positive  $(m+1) \times (m+1)$  matrix, and  $d \in \mathbb{R}^{m+1}$ .

The solution  $g$  found defines the new value for  $f|_{B_{current}}$ . If no solution is found, “fail” is returned.

Our implementation used the solver QL (see [5]).

## VI. CONVERGENCE RESULT

Each leaf-box than can appear during the execution of the algorithm can be defined by a finite list of booleans coding a path in the infinite tree. Therefore there is a bijection between these boxes and  $\mathbb{N}$ , and we can denote them by  $B^{(1)}, B^{(2)}, B^{(3)}, \dots$

The execution of the algorithm is entirely determined by an infinite sequence of independant coin flips randomly drawn, and for each box  $B^{(i)}$ , an infinite sequence of independant samples uniformly drawn in it (when new samples are needed

in a box  $B$ , we just follow the sequence of samples corresponding to this box). Let us call  $\mathcal{R}_{flips}$  the sequence of coin flips, and  $\mathcal{R}_{samples}^{(i)}$  the sequence of samples on box  $B^{(i)}$ . Let us denote by  $\mathcal{E}(\mathcal{R}_{flips}, (\mathcal{R}_{samples}^{(i)})_{i \in \mathbb{N}})$  the execution of the algorithm with random sequences  $\mathcal{R}_{flips}$  and  $(\mathcal{R}_{samples}^{(i)})_{i \in \mathbb{N}}$ , and let us also denote by  $f_i$  the approximation obtained after  $i$  iterations of the while loop (line 3 of Algorithm 1).

It can be shown that with probability 1, the following properties are verified:

- 1) In the sequence of coin flips there are infinitely many “tails” and infinitely many “heads”, and what’s more, any finite sequence appears infinitely many times (e.g. “tails-heads”, or “heads-heads-tails”, etc.).
- 2) For each box  $B^{(i)}$ ,  $\mathcal{R}_{samples}^{(i)}$  contains infinitely many points in any non-empty open subset of  $B^{(i)}$ .

Under these assumptions, we have the following:

**Theorem VI.1.** *For all compact subsets  $X \subset B_0$  such that  $\forall \mathbf{x} \in X, C(\mathbf{x}) \neq 0$ , there exists  $n_0 \in \mathbb{N}$  such that:*

$$\forall n > n_0, \forall \mathbf{x} \in X, \begin{cases} C(\mathbf{x}) > 0 \Rightarrow f_n(\mathbf{x}) > 0 \\ C(\mathbf{x}) < 0 \Rightarrow f_n(\mathbf{x}) < 0 \end{cases}$$

Since the frontier is assumed to be of Lebesgue measure zero, we have the immediate corollary:

**Corollary VI.1.** *The Lebesgue measure of  $Err_n$ , the set of points of  $B_0$  on which  $f_n$  has not a correct sign, tends to zero when  $n$  tends to  $+\infty$ .*

We will demonstrate Theorem VI.1 with the default version of the algorithm, the one where groups of samples are simply drawn uniformly, but with very small changes in the proof we obtain the same theorem with the algorithm variants. But before proving the theorem, we demonstrate three preliminary lemmas:

**Lemma VI.1.** *For any leaf-box  $B$ , we always have:*

$$co(B) \geq \frac{k+1}{4(\text{volume of } B)},$$

where  $k$  is the number of samples in  $B$ .

*Proof:* It is true for the initial value of  $co(B_0)$ , and when the equation (1) is used, we have  $co(B) \geq \frac{k}{(\text{volume of } B')}$ , with  $k \geq \frac{k+1}{4}$ .

When the equation (2) is applied, it means that  $B$  contains no sample, but the parent box  $B'$  necessarily contained samples (otherwise it wouldn't have been split), so we had  $co(B') \geq \frac{1}{(\text{volume of } B')}$  (by eq. (1)). Since the volume of  $B$  is half the volume of  $B'$ , we obtain indeed:

$$co(B) \geq \frac{1+0}{4(\text{volume of } B)}$$

**Lemma VI.2.** *Let  $B$  be a leaf-box after iteration  $i$  of Algorithm 1.*

*There exists  $j > i$  such that  $B$  will be selected by Algorithm 2 at iteration  $j$ .*

*Proof:* We prove it by contradiction. We assume that  $B$  belongs to *Set\_of\_PositiveLeaves*, but the reasoning would be the same with  $B \in \text{Set\_of\_NegativeLeaves}$  or  $B \in \text{Set\_of\_FrontierLeaves}$ .

Since  $B$  is never to be selected, the value  $co(B)$  will never change.

Because of Lemma VI.1, the number of leaf-boxes  $B^{(i)}$  which can verify  $co(B_i) \leq co(B)$  is finite (for the boxes  $B^{(i)}$  that are too small, namely such that  $co(B) < (4(\text{volume of } B_0))^{-1}$ , this inequality cannot be verified). Let us call them  $B^{(i_1)}, B^{(i_2)}, \dots, B^{(i_k)}$ .

Thanks to the assumption made on the sequence of coin flips, it can be shown that *Set\_of\_PositiveLeaves* will be selected infinitely many times by Algorithm 2. Each time, some samples will be put in one of the boxes  $B^{(i_1)}, B^{(i_2)}, \dots, B^{(i_k)}$ . Since for those boxes we have  $co(B^{(i_k)}) \geq \frac{1+0}{4(\text{volume of } B^{(i_k)})}$ , each one of them can only receive a finite number of samples (when the value  $co(B^{(i_k)})$  exceeds  $co(B)$ ,  $co(B^{(i_k)})$  cannot be chosen anymore).

So an infinite number of samples must be put in a finite number of boxes which can all only receive a finite number of samples. This is a contradiction, and we conclude that the box  $B$  will be selected. ■

**Lemma VI.3.** *Let  $B$  be a leaf-box that will eventually be split. Then the intersection between  $B$  and the frontier is not empty.*

*Proof:* If the intersection was empty, since  $C$  is continuous, it would either be negative on all points of  $B$ , or positive on all points of  $B$ . In that case  $f|_B$  would always be defined as a constant on  $B$ , and  $B$  would never be split. ■

Now we can prove the main theorem:

*Proof of Theorem VI.1:* Let  $X$  be a compact subset of  $B_0$  such that  $\forall \mathbf{x} \in X$ ,  $C(\mathbf{x}) \neq 0$ , and let us call  $d > 0$  the distance between  $X$  and the frontier. We call  $X_{d/2} \supset X$  the open set of points  $x \in B_0$  such that the distance between  $x$  and the frontier is greater than  $d/2$ . Thanks to Lemma VI.3 we know that after some iteration, all the leaf-boxes that have a non-empty intersection with  $X_{d/2}$  will never be split, because in the opposite case, the minimum size of boxes that will be split and have a non-empty intersection with  $X_{d/2}$  would endlessly decrease and tend to zero, which is a contradiction because since the distance between  $X_{d/2}$  and the frontier is

$d/2 > 0$ , no box too small can both contain points of  $X_{d/2}$  and points of the frontier.

So there exists  $n_0 \in \mathbb{N}^+$  such that after iteration  $n_0$ , all the leaf-boxes having a non-empty intersection with  $X_{d/2}$  are fixed. Let us consider one such leaf-box  $B^{(i)}$ . Thanks to lemma VI.2, we know that  $B$  will be picked an infinite number of times, and as a result will receive an infinite number of samples. Thus,  $B^{(i)}$  will never belong to *Set\_of\_FrontierLeaves*, because otherwise it would be eventually split (see section V: with more than  $N_{max}$  samples, leaf-boxes of *Set\_of\_FrontierLeaves* are automatically split). As a consequence,  $f|_{B^{(i)}}$  has a constant sign, which will never change. Besides, it is easy to show that the intersection between  $B^{(i)}$  and  $X_{d/2}$  contains an open subset, and since  $\mathcal{R}_{samples}^{(i)}$  contains infinitely many points in any non-empty open subset of  $B^{(i)}$ , we know that at some point a sample of  $X_{d/2}$  will be drawn. It follows that the sign of  $f|_{B^{(i)}}$  is equal to the sign of  $C$  on a point of  $X_{d/2} \cap B^{(i)}$ . But since  $C$  is continuous, it can only have a constant sign on  $X_{d/2} \cap B^{(i)}$ ; otherwise  $B^{(i)}$  we could easily obtain a contradiction by considering two open subsets of  $B^{(i)}$ , one on which  $C$  is positive, and one on which  $C$  is negative. Since we know that samples will eventually be drawn in these two open subsets,  $B^{(i)}$  would be transferred to *Set\_of\_FrontierLeaves*, which as we have proved cannot happen.

Hence, the sign of  $f|_{B^{(i)}}$  is equal to the sign of  $C$  on every point of  $X_{d/2} \cap B^{(i)}$ . By extending this result to all the leaf-boxes having a non-empty intersection with  $X_{d/2}$ , we deduce that after iteration  $n_0$ ,  $f$  has a correct sign on the entire set  $X_{d/2}$ , and *a fortiori* on  $X$ :

$$\forall n > n_0, \forall \mathbf{x} \in X, \begin{cases} C(\mathbf{x}) > 0 \Rightarrow f_n(\mathbf{x}) > 0 \\ C(\mathbf{x}) < 0 \Rightarrow f_n(\mathbf{x}) < 0 \end{cases}$$

■

## REFERENCES

- [1] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *27th annual conference on Computer graphics and interactive techniques (SIGGRAPH'00)*, pages 249–254, 2000.
- [2] Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. Multidimensional adaptive sampling and reconstruction for ray tracing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, New York, NY, USA, 2008. ACM.
- [3] P. Hardwick and Q.F. Stout. Flexible algorithms for creating and analyzing adaptive sampling procedures. In *Developments and Applications in Experimental Design, IMS Lec. Notes–Mono. Series 34*, pages 91–105, 1998.
- [4] N. Perrin, O. Stasse, F. Lamiroux, and E. Yoshida. Approximation of feasibility tests for reactive walk on hrp-2. In *IEEE Int. Conf. on Robotics and Automation*, pages 4243–4248, 2010.
- [5] K. Schittkowski. Q1: A fortran code for convex quadratic programming - user's guide, version 2.11. Technical report, University of Bayreuth, 2005.