PARALLELIZATION METHOD FOR A CONTINUOUS PROPERTY

PAWEŁ PILARCZYK

Abstract. An automated method of general purpose is introduced for computing a rigorous estimate of a bounded region in $\mathbb{R}^n$ whose points satisfy a given property. The method is based on calculations conducted in interval arithmetic and the constructed approximation is built of rectangular boxes of variable sizes. An efficient strategy is proposed, which makes use of parallel computations on multiple machines and refines the estimate gradually. It is proved that under certain assumptions the result of computations converges to the exact result as the precision of calculations increases. Time complexity of the algorithm is analyzed, and the effectiveness of this approach is illustrated by constructing a lower bound of the set of parameters for which an overcompensatory nonlinear Leslie population model exhibits more than one attractor, which is of interest from the biological point of view. This paper is accompanied by efficient and flexible software written in C++ whose source code is freely available at http://www.pawelpilarczyk.com/parallel/.

1. Introduction

In various applications of numerical computations, an estimate of a region in $\mathbf{R}^n$ of points which satisfy certain properties is often of interest. For example, finding a set of parameters of a dynamical system for which certain dynamical features can be observed is of great importance from the point of view of applications (see [1] for a nontrivial example). However, efficient computation of a good approximation of such a region is in general a difficult task.

With the use of interval arithmetic [9] and rigorous computational methods, one can numerically verify certain properties for entire intervals instead of single numbers. This makes it possible to process all the points contained in a rectangular box in $\mathbf{R}^n$ simultaneously, by substituting the edges of the box into formulas in place of coordinates of the points. In this way, one can obtain a result valid for all the points in the box, computed at one fell swoop.

A straightforward approach based on interval arithmetic for computing a rigorous estimate for a set A of points in some bounded rectangular set $R \subset \mathbf{R}^n$ which satisfy a given property (predicate) P would be to subdivide the region R uniformly into a family of (small) boxes, and to verify the property P numerically for each box (see [2] for an example of using this strategy). Then one would take these boxes for which the verification was successful as an estimate for the set A. However, determining the right size of the subdivision of R into boxes (also called *grid*)

2000 *Mathematics Subject Classification.* Primary: 65D99; Secondary: 68W10, 65G40.

Key words and phrases. rigorous numerics, parallel computing, heuristics, algorithm, iterative method, interval arithmetic.

Communicated by Richard Scwartz.

P. Pilarczyk was partially supported by the JSPS Postdoctoral Fellowship No. P06039 and by Grant-in-Aid for Scientific Research (No. 1806039), Ministry of Education, Science, Technology, Culture and Sports, Japan.

The final version of this preprint has been published in *Foundations of Computational Mathematics*, Vol. 10, No. 1, 2010, pp. 93–114, DOI: 10.1007/s10208-009-9050-8, and is available at www.springerlink.com.

is often a matter of a good guess. If the grid is too coarse, then not only the obtained approximation of A is rough, but also the computations for these boxes can easily fail (because of overestimates) instead of revealing the delicate properties that might only be detected if smaller boxes are taken. On the other hand, too fine a grid induces the need for running an excessive amount of computations necessary to process all the boxes in the entire range R, many of which are outside A. In contrast to processing full boxes, conducting the computations for single points (e.g., corners of the boxes), yields the most precise computations, because there are no overestimates introduced by the positive size of intervals. However, in this way one cannot obtain rigorous results for the entire boxes.

Taking the above-mentioned arguments into consideration, we combine the power of computations for single points (further called *probes*) and the ultimate goal of completing the computations for entire boxes into an algorithm which constructs a rigorous approximation of the region of interest built of boxes of various sizes. The algorithm first creates a rough approximation of the region with respect to some fixed grid, and then refines the result by subdividing the boxes for which it was not possible to prove the given property but chances for success are reasonably high, which is determined by testing probes at the corners of boxes of interest. The details of this algorithm are provided in Section 4. Time complexity analysis conducted in Section 5 reveals the superiority of our method in comparison with the "naive" approach based on subdividing R into boxes with respect to a fixed grid.

Although this method can be applied to obtain a rigorous lower bound for the set A of points satisfying an arbitrary property P that is computationally verifiable, it yields good results if P is in some sense "continuous", that is, if it has a feature in the following spirit: If P is satisfied for some point then it is also satisfied for all the points in its vicinity. This translates to the shape of A, which cannot be too complicated, e.g., it could be either convex with nonempty interior, or at least every point in A should be close to some relatively large box contained in A. A sample definition of this notion is given in Section 6 for the purpose of proving the convergence of this method, but it is the intuitive meaning explained above which is important if one considers using our method for a specific application. In particular, this method seems to be inappropriate if one expects that A might be a fractal or if it is likely that the interior of A might be empty.

Since the computations for each box in $\mathbf{R}^n$ may be time-consuming, which is especially the case if a region of parameters is computed for which some complicated property is satisfied, it is a good idea to spread the computations over several machines. A model of parallel computations that seems to be most suitable for this application is introduced in Section 3. It is a centralized scheme in which the main process is responsible for determining small rectangular regions in $\mathbf{R}^n$ that are to be analyzed, and the other processes do the verification for these regions. The task of the main process is also to interpret the results of this verification and incorporate these results into the computed area. To make good use of the power of parallel computations, the algorithm must be constructed in such a way that multiple probes and boxes can be sent for analysis simultaneously. Moreover, the fact that the results of computations for these probes and boxes can arrive in an unpredictable order must also be taken into consideration. These issues are addressed in the algorithm proposed in Section 4.

Although the method introduced in this paper may seem at first glance to be based on some heuristic techniques and intuitive observations, it works very well in practice, and we prove that it is mathematically sound. In particular, in Section 5 we conduct its time complexity analysis, and in Section 6 we prove its convergence.

We show that under certain assumptions on the property which is to be verified and on the computational method used to verify it, the increase in numerical accuracy and in the depth of computations results in a better approximation of the computed region in terms of the Hausdorff distance between the constructed set and the actual region of interest.

In order to show the power of this method, in Section 7 we analyze a specific example of a dynamical system that describes a certain biological model. We compute a rigorous lower bound for the region in the parameter space for which it is possible to prove the existence of at least two disjoint attraction basins. This feature is of great importance for the modeled ecosystem, as we explain in Section 7.

The source code of a software implementation of the method introduced in this paper, including the analyzed examples, is freely available for download at [10] under the terms of the GNU General Public License. The C++ programming language was chosen for this implementation, mainly because it is a widely acclaimed standard with the excellent GNU C++ compiler and many specialized software libraries, including interval arithmetic packages, freely available for virtually any computer architecture. Another argument in favor of using the C++ language is that well-written code in C++ is often much faster than software prepared in other programming languages, although the latter might have various advantages over C++, like being more flexible or more mathematically-oriented. Moreover, the strategy of using C++ for academic computations also eliminates the need for using specialized mathematical software, rarely available without prohibitively high license fees.

2. Preliminaries

In this section some notation and definitions are introduced, which are going to be used in this presentation.

Let $\mathcal{R} \subset \mathbf{R}$ denote an a priori chosen finite set of rational numbers, further called *representable numbers*, that can be encoded in the computer using a specific representation, which will remain fixed throughout the paper. The only assumption which we make about this representation is that a certain number of powers of 2 are representable, and also that the division of representable numbers which are not too small in absolute value by a limited number of positive powers of 2 results in numbers which are also representable. These assumptions are necessary for the subdivision framework defined in Section 4, and in Section 6 where the convergence of the method is analyzed. In practice, we consider the 64-bit representation of numbers introduced in the IEEE 754 standard, for which these assumptions are satisfied. The fact that this standard is followed by many manufacturers of processors (e.g., Intel, AMD, Sparc) makes this representation easy to use in applications. However, other representations can also be used, especially if higher precision of computations is required.

A closed interval $I = [x_1, x_2]$ is called *representable* if $x_1, x_2 \in \mathcal{R}$. If $x_1 = x_2$ then the interval $I = [x_1, x_2] = \{x_1\}$ is called *degenerate*. The set of all closed representable intervals (including the degenerate ones) will be further denoted by $\mathcal{I}$. A set $Q \subset \mathbf{R}^n$ is called a *representable rectangular set* if it is the product of some closed representable intervals: $Q = I_1 \times \cdots \times I_n$ for $I_1, \ldots, I_n \in \mathcal{I}$. It is called *degenerate* if at least one of the intervals $I_1, \ldots, I_n$ in the product is degenerate. The *size* of Q is defined as $\max\{|I_1|, \ldots, |I_n|\}$, where $|I_i|$ denotes the length of the interval I_i for $i = 1, \ldots, n$. The set of all representable rectangular sets contained in some $R \subset \mathbf{R}^n$ is denoted by $\mathcal{B}(R)$. A set $A \subset \mathbf{R}^n$ is called *representable* if it is a (finite) union of some representable rectangular sets. Note that since $\mathcal{R}$ is finite,

so are the sets $\mathcal{I}$ and $\mathcal{B}(R)$ for any $R \subset \mathbf{R}^n$. A representable set $A \subset \mathbf{R}^n$ is called a *lower estimate* for another set $B \subset \mathbf{R}^n$ if $A \subset B$.

Any function $P\colon X \to \{0,1\}$ is called a *property on* X. We say that the property P *is satisfied* for a point $x \in X$ if and only if $P(x) = 1$. The set $\{x \in X : P(x) = 1\}$ is called the *support* of P. Given a bounded set $R \subset \mathbf{R}^n$ and a property P on R, we say that a property $\mathcal{P}$ on $\mathcal{B}(R)$ is a *lower estimate for* P if for every $Q \in \mathcal{B}(R)$ such that $\mathcal{P}$ is satisfied for Q, the property P is satisfied for every $x \in Q$.

A computable lower estimate $\mathcal{P}$ for P can be used to confirm that P is satisfied for all $x \in Q$ for a specific $Q \in \mathcal{B}(R)$, if $\mathcal{P}$ is satisfied for Q. However, if $\mathcal{P}$ is not satisfied for Q then this provides no conclusion as to whether P is satisfied or not for the points in Q. In this way, one can use $\mathcal{P}$ to find explicitly a possibly large subset of R on which the property P is satisfied; this can be instantly achieved by simply taking the union of all $Q \in \mathcal{B}(R)$ such that $\mathcal{P}(Q) = 1$. However, since the number of elements of $\mathcal{B}(R)$ is typically huge, this approach is useless in practice. A much more efficient algorithm for this purpose is introduced in Section 4.

The method described in this paper assumes that a computable lower estimate $\mathcal{P}$ for P on R is *monotonic*, that is, if $\mathcal{P}$ is satisfied for some $Q \in \mathcal{B}(R)$, then $\mathcal{P}$ is also satisfied for every $Q' \in \mathcal{B}(R)$ such that $Q' \subset Q$. This assumption is very natural if $\mathcal{P}$ is defined as the output of some computational method for verifying P with the use of interval arithmetic, because smaller intervals in the input of the procedure for $\mathcal{P}$ generate smaller overestimates in the calculations aimed at confirming the property P.

3. Parallel computations framework

In this section we introduce a simple and reliable framework for centrally controlled parallel computations, which is suitable for the algorithm discussed in Section 4.

A task or program running on a single computer is called a *process*. In this framework, one of the processes is designed to control the computations and is called the *coordinator*. All the other processes which take part in the computations are called *workers*.

The computations are initialized by starting the coordinator process. In the software accompanying this paper, the coordinator reads the log of previously completed computations, so that it can resume the work that was interrupted, if any. This is a very useful feature in practice, but in general this step is optional. If no previous log is found then the computations are started from scratch. Then worker processes are launched (possibly at different machines). Each of them opens a network connection with the coordinator to join the computations and acquire initialization data, if any.

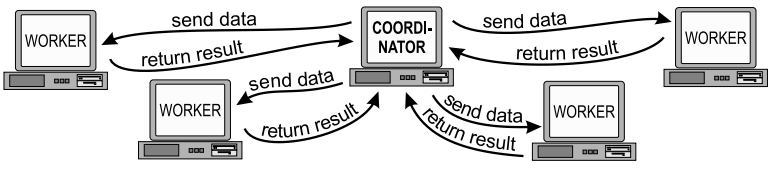

Figure 1. A simple framework for centrally controlled parallel computations.

During the computations, the coordinator prepares portions of data to be processed by workers. The coordinator should be able to prepare multiple portions of data before acquiring all the results of processing previously sent data, so that all the workers which take part in the computations can be kept busy. Once a

worker completes processing its portion of data, it sends the computed result to the coordinator over the network connection, and acquires another portion of data (see Figure 1).

For the purpose of the simplicity of usage, in the implementation of this model available in [10], the coordinator and the worker are programmed as separate classes in C++, and the network communication is hidden from the end user of the software. The only procedures that need to be programmed are the functions `Accept` and `Prepare` of the coordinator, and the functions `Initialize` and `Process` of the worker. The function `Accept` is called when a result of computations has been acquired from a worker and must be incorporated into the set of all the results of computations. The function `Prepare` is called each time a portion of data needs to be prepared for an idle worker. This function should return a data portion for processing or indicate that there is no more data that needs to be sent at the current state of the computations (which can change if new data is acquired from some worker). The function `Initialize` is called if initialization data is obtained from the coordinator after the worker process has joined the computations. The function `Process` is called when a piece of data acquired from the function `Prepare` and sent over the network must be processed by a worker. This function returns a portion of data which is then transmitted back to the coordinator.

New workers can connect to the coordinator not necessarily at the beginning of the computations, but also at a later time. This feature is useful if one initially runs the computations on a small number of processors, and at some point one decides to speed them up by using more resources, or if a group of additional processes are started at a computer cluster.

Moreover, if it happens that some worker process dies (e.g., by meeting the assigned wall time when running at a computer cluster) or its network connection with the coordinator is suddenly broken, then the data which was sent to the unfortunate worker is immediately sent to another worker, as soon as one becomes ready. Note that in this scheme no data is lost if a worker is dropped, in contrast to what typically happens in distributed computation solutions. Even better, resending the data is transparent, so this case need not be considered separately in algorithms that use this scheme.

Additionally, it is also possible that the coordinating process itself does the computations dedicated for workers if none is connected.

To sum up, the number of active workers can easily change during the computations without causing any data loss, unlike in typical parallel computing solutions. This feature is very useful in practice, and may save a considerable amount of time when running extensive computations, mainly because it eliminates the need of restarting the entire job in case of failure of some processes taking part in the computations, or if the number of computers assigned to the task needs to be changed.

A software implementation of the parallel computations scheme described in this section is included in the software package [10]. A notable feature of this implementation is its architecture independence, which allows to run computations in a heterogeneous environment, using various types of machines simultaneously (like PCs, MACs, or SUNs), possibly running different operating systems (e.g., Windows, Linux, MacOS X, or Solaris). Detailed instructions for the usage of this software, together with some examples that clarify various aspects of this scheme and specific features of the software, are also included in [10].

4. The subdivision algorithm

Let R be a representable rectangular set in $\mathbf{R}^n$ and let P be a property on R. We are interested in estimating the support of P, that is, the set A of all the points in

R satisfying the property P. Assume that some lower estimate $\mathcal{P}$ for P is available, which provides an automatic method for verifying if the property P is satisfied on representable rectangular sets $Q \subset R$ in the following way: If it is computed that $\mathcal{P}(Q) = 1$ then $P(x) = 1$ for all $x \in Q$, and thus $Q \subset A$, but the result $\mathcal{P}(Q) = 0$ is inconclusive.

In this section we use the parallelization framework defined in Section 3 to introduce an algorithm for computing a rigorous lower estimate $A_{\approx}$ of the support A of P using the computable property $\mathcal{P}$. Before providing the details of the algorithm, we would like to remark that this method can also be used to compute a rigorous *upper estimate* of the set $B := R \setminus A$ on which the negation of the property P is satisfied, by taking the complement of the computed set $A_{\approx}$ with respect to the entire region R.

4.1. **Representable subdivisions.** We begin with introducing the terminology for describing representable subdivisions of the representable rectangular set R. Given a representable interval $I = (a, b)$ and an integer $q > 0$, we say that a set $\mathcal{S}$ of subintervals of I is called a *subdivision of I of order q* if $\mathcal{S}$ consists of 2^q non-degenerate representable intervals with disjoint interiors, whose union equals I. The set of endpoints of these intervals is called the *corresponding set of subdivision points*. For a representable rectangular set $R = I_1 \times \cdots \times I_n \subset \mathbf{R}^n$, a *subdivision of R of order q* is any set $\mathcal{S} := \big\{J_1 \times \cdots \times J_n : J_j \in \mathcal{S}_j \text{ for all } j = 1, \ldots, n\big\}$ of 2^{qn} representable rectangular subsets of R, where each $\mathcal{S}_j$ is some subdivision of I_j of order q. The set of all the corners of these sets is called the *corresponding set of subdivision points*.

Let $\mathcal{S}_q(R)$ denote the subdivision of $R = I_1 \times \cdots \times I_n$ of order q generated by the subdivision of each $I_j = (a_j, b_j)$ in which the subdivision points $x(i, j, q)$, for $i = 0, \ldots, 2^q$ and $j = 1, \ldots, n$, are computed by rounding the result of each arithmetic operation in the formula

$$a_j + (b_j - a_j)(i/2^q) \tag{1}$$

to the nearest representable number, or $\mathcal{S}_q(R) := \emptyset$ if $x(i_1, j_1, q) = x(i_2, j_2, q)$ for some $(i_1, j_1) \neq (i_2, j_2)$, which can happen because of rounding to representable numbers. Denote the corresponding set of subdivision points by $\mathcal{E}_q(R)$, or define $\mathcal{E}_q(R) := \emptyset$ if $\mathcal{S}_q(R) = \emptyset$. Whenever the set R is clear from the context, we will omit this argument and write $\mathcal{S}_q$ or $\mathcal{E}_q$ for short. If $q, r > 0$ and all the numbers $i/2^q$, $i/2^{q+r}$ are representable (which is usually true for binary representations of real numbers if the number $q + r$ is not too large, as discussed in the Introduction), then it is obvious that for any $S \in \mathcal{S}_q(R)$ the set $\{Q \in \mathcal{S}_{q+r}(R) : Q \subset S\}$ is a subdivision of S of order r, provided that $\mathcal{S}_{q+r}(R) \neq \emptyset$. This feature will be used in our algorithm.

4.2. **Processing rectangular sets.** Since we are going to use the parallel computations framework defined in Section 3, in order to provide an algorithm for our method, we need to define the following three functions called within that framework: `Process` for computing the property $\mathcal{P}$ on any representable rectangular set $Q \subset R$, `Prepare` for determining which box Q to send for processing, and `Accept` for integrating the result of the computation of $\mathcal{P}(Q)$ for a previously sent box Q into the approximation of A being constructed. Although the function `Initialize` is empty in this algorithm, in the applications it can set some configuration options for the computation of the function $\mathcal{P}$. The implementation of the first of these functions depends on the actual property P and a numerical lower estimate $\mathcal{P}$ used to verify it, and can be summarized as follows.

Algorithm 1 (the function `Process`)**.**

input: $Q \in \mathcal{B}(R)$, $q > 0$;
output: 0 or 1;
code:
    return $\big(Q, q, \mathcal{P}(Q)\big)$;

The number $q > 0$ in the input corresponds to the subdivision order, and the input data is returned together with the computed valued of $\mathcal{P}(Q)$ in order to enable its identification in Algorithm 3.

4.3. **Preparing data for processing.** Since both functions `Prepare` and `Accept` are run by the same process (the coordinator) and their actions in some sense complement each other, it is natural that they must share several variables. In particular, several finite lists of representable rectangular sets under analysis are stored in the variables listed below. The elements of $\mathcal{S}_q(R)$ are called *boxes*, and the elements of $\mathcal{E}_q(R)$ are called *probes*. Words justifying the symbol used are indicated in italics.

- $\mathcal{A}$ — boxes for which it has been successfully verified that they are contained in A by computing $\mathcal{P}(Q) = 1$;
- $\mathcal{F}$ — boxes for which the above computation *failed*, that is, $\mathcal{P}(Q) = 0$ was computed;
- $\mathcal{W}$ — boxes *waiting* for the above verification for which the value $\mathcal{P}(Q)$ is yet unknown, but it is likely to be 1;
- $\mathcal{G}$ — *good* probes, that is, probes for which $\mathcal{P}(Q) = 1$ has been determined;
- $\mathcal{N}$ — *negative* probes, that is, probes for which $\mathcal{P}(Q) = 0$ has been computed;
- $\mathcal{T}$ — probes which are waiting for *testing*;
- $\mathcal{C}$ — boxes and probes *currently* being processed (by some worker).

Each of these sets is actually stored in the algorithms as a series of sets indexed by the subdivision level denoted as a subscript, e.g., $\mathcal{A}_q \subset \mathcal{S}_q(R)$ and $\mathcal{N}_q \subset \mathcal{E}_q(R)$. All these sets are initially empty, except where initialized otherwise.

The function for preparing data to send to a worker is quite straightforward. It tries to find a probe or a box which waits for processing, and which belongs to the set $\mathcal{T}_q$ or $\mathcal{W}_q$ with the lowest possible index q, with priority given to probes. It returns $\emptyset$ if no more data needs processing at the current stage of computations.

Algorithm 2 (the function `Prepare`)**.**
input: none;
output: either $\emptyset$ or a pair (Q, q) with $Q \in \mathcal{S}_q(R) \cup \mathcal{E}_q(R)$;
code:
    $q_{\mathcal{T}} := \inf\{q > 0 : \mathcal{T}_q \setminus \mathcal{C}_q \neq \emptyset\}$;
    $q_{\mathcal{W}} := \inf\{q > 0 : \mathcal{W}_q \setminus \mathcal{C}_q \neq \emptyset\}$;
    if $q_{\mathcal{T}} < \infty$ **and** $q_{\mathcal{T}} \leq q_{\mathcal{W}}$ **then**
        $q := q_{\mathcal{T}}$; $Q :=$ any element of $\mathcal{T}_q \setminus \mathcal{C}_q$;
    else if $q_{\mathcal{W}} < \infty$ **then**
        $q := q_{\mathcal{W}}$; $Q :=$ any element of $\mathcal{W}_q \setminus \mathcal{C}_q$;
    else return $\emptyset$;
    $\mathcal{C}_q := \mathcal{C}_q \cup \{Q\}$;
    return (Q, q);

4.4. **Acquiring processed data.** The function which acquires data processed by workers is more complicated. The general idea is to make sure that probes are tested first. If this verification fails then the probe is abandoned together with the area around it. If it succeeds, however, then all the boxes which have the probe at one of their corners are enqueued for further testing, provided that they are not yet contained in $A_{\approx}$. If the verification for some box fails then the box is subdivided

and smaller boxes around good probes located at its corners are tested next. If it succeeds, however, then the entire box is added to $\mathcal{A}$, its corners are automatically assumed to be good probes, and all the adjacent boxes which contain these probes are taken for subsequent verification. The details are provided in the algorithm

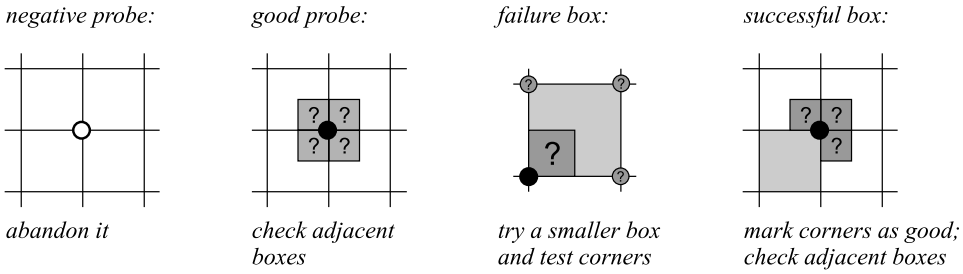

FIGURE 2. Rough idea of treating results computed for probes and boxes in Algorithm 4.

below, and the rough idea behind treating the four cases is illustrated in Figure 2. Some effort is undertaken to avoid checking unnecessary probes or testing boxes for which the result is known a priori because of the assumed monotonicity of $\mathcal{P}$. Moreover, as it follows form the definition of $\mathcal{S}_q(R)$, if the size of a subdivided box is so small that degenerate boxes appear then they are not analyzed any further, because the maximal feasible resolution is considered to have been reached in that case. Note that one may prefer to stop the subdivision process even earlier, before some subdivision order $q_\infty > 0$ has been reached. This can be easily done by redefining $\mathcal{S}_{q_\infty}(R) := \emptyset$ and $\mathcal{E}_{q_\infty}(R) := \emptyset$, as proposed in Algorithm 4. This feature is also included in the software implementation [10] of this algorithm, in which the maximal allowed subdivision level q_∞ must be specified explicitly.

Algorithm 3 (the function `Accept`)**.**
input: $Q \in \mathcal{B}(R)$, $q > 0$, $r \in \{0, 1\}$;
output: none;
code:

$\mathcal{C}_q := \mathcal{C}_q \setminus \{Q\}$; $\mathcal{T}_q := \mathcal{T}_q \setminus \{Q\}$; $\mathcal{W}_q := \mathcal{W}_q \setminus \{Q\}$;
(case 1 – a negative probe:)
if $r = 0$ **and** $Q \in \mathcal{E}_q(R)$ **then**
  $\mathcal{N}_q := \mathcal{N}_q \cup \{Q\}$;
  for each $U \in \mathcal{W}_q \setminus \mathcal{C}_q$ such that $Q \in U$ **do**
    `Accept` $(U, q, 0)$;
(case 2 – a good probe:)
else if $r = 1$ **and** $Q \in \mathcal{E}_q(R)$ **and** $Q \notin \mathcal{G}_q$ **then**
  $\mathcal{G}_q := \mathcal{G}_q \cup \{Q\}$; $\mathcal{G}_{q+1} := \mathcal{G}_{q+1} \cup \{Q\}$;
  for each $U \in \mathcal{S}_{q+1}(R)$ such that $Q \in U$ **do**
    if not ($U \subset W$ for some $W \in \mathcal{A}_{q'} \cup \mathcal{F}_{q'}$
      with $q' \le q + 1$) **then**
      $\mathcal{W}_{q+1} := \mathcal{W}_{q+1} \cup \{U\}$;
(case 3 – a box for which the verification failed:)
else if $r = 0$ **and** $Q \in \mathcal{S}_q(R)$ **then**
  $\mathcal{F}_q := \mathcal{F}_q \cup \{Q\}$;
  for each $V \in \mathcal{E}_q(R)$ such that $V \in Q$ **do**
    if $V \in \mathcal{G}_q$ **and** $\mathcal{S}_{q+1}(R) \neq \emptyset$ **then**
      let $U \in \mathcal{S}_{q+1}(R)$ such that $V \in U \subset Q$;
      if $U \notin \mathcal{A}_{q+1} \cup \mathcal{F}_{q+1}$ **then**

$\mathcal{W}_{q+1} := \mathcal{W}_{q+1} \cup \{U\}$;
$\mathcal{G}_{q+1} := \mathcal{G}_{q+1} \cup \{V\}$;
if $V \notin \mathcal{G}_q \cup \mathcal{N}_q$ **then**
$\mathcal{T}_q := \mathcal{T}_q \cup \{V\}$;
(case 4 – a box which is proved to be contained in A:)
else if $r = 1$ **and** $Q \in \mathcal{S}_q(R)$ **then**
if not ($Q \subset U$ for some $U \in \mathcal{A}_{q'}$ with $q' \le q$) **then**
$\mathcal{A}_q := \mathcal{A}_q \cup \{Q\}$;
for each $q' > q$ such that $\mathcal{W}_{q'} \neq \emptyset$ **do**
for each $U \in \mathcal{W}_{q'} \setminus \mathcal{C}_{q'}$ such that $U \subset Q$ **do**
$\mathcal{W}_{q'} := \mathcal{W}_{q'} \setminus \{U\}$;
for each $V \in \mathcal{E}_q(R)$ such that $V \in Q$ **do**
`Accept` $(V, q, 1)$;

4.5. **The main algorithm.** The three functions defined in the previous subsections are combined in the following algorithm which computes a rigorous lower bound $A_\approx$ for the set A.

Algorithm 4 (the computation of $A_\approx$)**.**
input: a representable rectangular set R; a property $\mathcal{P}$ on $\mathcal{B}(R)$;
two integers $q_\infty > q_0 > 0$;
output: a representable subset of R;
code:
set $\mathcal{T}_{q_0} := \mathcal{E}_{q_0}(R) \cap \mathrm{int}R$;
redefine $\mathcal{S}_{q_\infty}(R) := \emptyset$ and $\mathcal{E}_{q_\infty}(R) := \emptyset$;
run the parallel computations framework with the procedures
`Process`, `Prepare` and `Accept` defined by Algorithms 1, 2
and 3, respectively, for $\mathcal{P}$, R, and the initial $\mathcal{T}$ as above;
return $A_\approx := \bigcup \mathcal{A}$;

Before we analyze the complexity of this algorithm in Section 5 and we prove its asymptotic properties in Section 6, we would like to make an observation that this algorithm always executes in finite time, and thus has the termination property. This holds true because each probe and each box that appears in the algorithm is processed at most once, and the number of those boxes and probes does not exceed the number of elements of all the sets $\mathcal{E}_q(R)$ and $\mathcal{S}_q(R)$ for $q = q_0, \ldots, q_\infty - 1$, which are all finite.

We would like to end this section with a remark indicating one of the benefits of the incremental strategy of approximations exercised in Algorithm 4. Namely, if one decides to refine a previously computed approximation of A by increasing the target resolution q_∞ to $q_\infty^+ > q_\infty$ then Algorithm 4 can be re-run with the previously computed (and stored) values of $\mathcal{P}$ on boxes in $\mathcal{S}_q(R)$ and probes in $\mathcal{E}_q(R)$ for $q = q_0, \ldots, q_\infty - 1$. Then the function `Process` called with boxes in $\mathcal{S}_q(R)$ and probes in $\mathcal{E}_q(R)$, for $q = q_0, \ldots, q_\infty - 1$, immediately returns the tabulated result in no time, and the computations effectively start at the subdivision level q_∞. This feature has been implemented in the software [10] accompanying this paper, and proves itself to be very efficient. Note that such an easy way of refining a previously computed result is not available in the "naive" approach based on testing all the cubes in $\mathcal{S}_{q_\infty - 1}$, in which the entire complement in R of previously obtained approximation of A must be processed from scratch in order to achieve the same result.

5. Time complexity of the algorithm

The time complexity of an algorithm is typically measured in terms of the number of elementary operations in the algorithm's run as a function of the size s of the input. The notation $O\big(f(s)\big)$ is used to indicate the asymptotic upper bound for the worst case in the following sense. It is said that the algorithm has the *(pessimistic) time complexity of* $O\big(f(s)\big)$ if there exist constants $c, s_0 > 0$ such that the number of elementary operations in the run of the algorithm on any input of size $s > s_0$ does not exceed $cf(s)$. See [7, §3.1] for a more detailed explanation of this notation and for a comprehensive introduction to the subject of time complexity of algorithms in general.

Since the evaluation of $\mathcal{P}(Q)$ is usually the bottleneck in the applications targeted at in this paper and the other combinatorial operations in Algorithm 4 are very simple, it is natural to measure the execution time of this algorithm in terms of the number of calls to the procedure `Process`. However, it is more difficult to determine the right measurement method for the size of the input, because various factors may play significant role in the computations. Therefore, for the sake of simplicity, let us assume that a property P on a rectangular representable set $R \subset \mathbf{R}^n$ has been fixed together with its lower bound $\mathcal{P}$ and the initial order q_0 of subdivisions of R. Then the time of running Algorithm 4 substantially depends on the number q_∞ which determines at what subdivision depth the computations are stopped. As a consequence, it seems to be most reasonable to measure the time complexity of Algorithm 4 as a function of q_∞.

In order to avoid complications in the formulation of Theorem 5 below, we will silently assume that whenever Algorithm 4 is run with some arguments $R, \mathcal{P}, q_0, q_\infty$, the set $\mathcal{R}$ is rich enough so that $S_{q_\infty - 1} \neq \emptyset$. Note that this assumption also implies that $\mathcal{P}$ is defined on a gradually larger set of representable boxes as q_∞ is increased. In practice, this assumption can be instantly satisfied if a software implementation of representable numbers with adjustable precision is used and $\mathcal{P}$ is evaluated in interval arithmetic using a formula for P.

Since the running time of this algorithm strongly depends on the inaccuracies in the evaluation of $\mathcal{P}$, for the purpose of obtaining realistic estimates we shall assume that $\mathcal{P}$ is accurate on boxes which are small enough and not too close to the boundary of the support A of P. We say that $\mathcal{P}$ is *accurate below subdivision level* q_1 if for all $Q \in \mathcal{S}_q$ with $q > q_1$ one has $\mathcal{P}(Q) = 1$ if $Q \subset A$ and all the neighbors of Q in $\mathcal{S}_q$ are also contained in A. In particular, we have $\mathcal{P}(Q) = 1$ for every probe $Q \in \operatorname{int} A$.

Note that since the number of boxes considered at subsequent subdivision levels grows exponentially, an exponential complexity with respect to q_∞ should be expected. We prove the following

Theorem 5. *Let P be a property on a representable rectangular set $R \subset \mathbf{R}^n$ whose support A is convex. Let $\mathcal{P}$ be a lower bound for P on R which is accurate below some subdivision level q_1. Let $q_0 > 0$. Then the pessimistic time complexity of Algorithm 4 called with R, $\mathcal{P}$, q_0 and q_∞, measured in terms of the number of calls to* `Process` *as a function of q_∞, is $O(2^{(n-1)q_\infty})$.*

Proof. Let us estimate the number of processed probes and boxes in the run of Algorithm 4. Initially, almost $2^{q_0 n}$ probes are tested. At each subsequent subdivision level $q = q_0 + 1, \dots, q_1$, no more than 2^{qn} boxes are tested, and at most that many probes as well. This totals to at most

$$C_0 := 2^{q_0 n} + 2 \sum_{q=q_0+1}^{q_1} 2^{qn} \leq 2^{(q_1+1)n+1}$$

calls to the procedure `Process`.

Denote by G_q the convex hull of $\mathcal{G}_q$ for $q \geq q_1$. Note that $G_q \subset A$ because A is convex. Since $\mathcal{P}$ is accurate below subdivision level q_1, the procedure `Process` will succeed on every $Q \in \mathcal{W}_{q_1+1}$ contained in G_{q_1} whose all neighbors are also contained in A. There are at most

$$C_1 := 2^{(q_1+1)n}$$

such boxes. Their sub-boxes as well as probes contained in them are not tested anymore.

Now consider $q > q_1 + 1$. Since all the boxes in $\mathcal{S}_{q-1}$ that are contained in G_{q-1} together with all their neighbors are already proven to be contained in A, the only probes in $\mathcal{E}_q$ and boxes in $\mathcal{S}_q$ which need to be tested are located along the boundary of G_{q-1}. Since G_{q-1} is convex and contained in R, a rough upper estimate for the number of those probes and boxes can be obtained by the maximal $(n-1)$-dimensional "area" of the belt along the boundary of G_{q-1}, measured by the means of boxes in $\mathcal{S}_q$ $(2n \cdot 2^{(n-1)q})$, times the number of possible neighbors of each probe (2^n), times the number of box layers that the boundary of G_{q-1} can intersect (2) plus one extra layer to include those boxes that have a neighbor sticking out of A $(+1)$. This gives a total of $(2+1) \cdot 2n \cdot 2^{(n-1)q} \cdot 2^n$ probes and boxes that can be potentially tested at the subdivision level q. Taking into consideration these estimates for $q = q_1 + 2, \ldots, q_\infty - 1$, we obtain the following upper bound on the number of calls to the function `Process`:

$$6n2^n \sum_{q=q_1+2}^{q_\infty-1} 2^{(n-1)q} \leq 6n2^n 2^{(n-1)q_\infty} = C_2 2^{(n-1)q_\infty},$$

where C_2 is a constant independent of q_∞. After having joined this result with the previous estimates, we obtain an upper bound on the number of calls to `Process` in the entire run of Algorithm 4 as $C_0 + C_1 + C_2 2^{(n-1)q_\infty}$, which is $O\big(2^{(n-1)q_\infty}\big)$. This completes the proof. □

We would like to point out the fact that the time complexity of Algorithm 4 is substantially better than that of the brute force approach based on scanning all the boxes in $\mathcal{S}_{q_\infty-1}$. The complexity of the latter is $O\big(2^{nq_\infty}\big)$, which is also exponential with respect to q_∞, but with a twice larger base. Therefore, the gain attained by using the strategy proposed in this paper is exponential.

6. Convergence of the method

In this section we prove that under certain assumptions Algorithm 4 introduced in Section 4 converges, that is, the computed approximation $A_\approx$ of A can fill A arbitrarily well as the amount of computations and their accuracy increase.

Typically, in the analysis of convergence of a numerical method it is assumed that the calculations are done with infinite precision, and thus the impact of numerical errors which arise in real computations is often neglected. Instead, in this paper we conduct a substantially more realistic analysis of convergence of Algorithm 4, which takes into account the finiteness of the set of representable numbers, and therefore leads to conclusions which are meaningful in real applications, not only in theory. Although this approach is somewhat more complicated than the "idealistic" version, we are firmly convinced that it is worth the effort.

In what follows we prove that the higher the precision of calculations is applied in terms of quality of both representable numbers $\mathcal{R}$ and a lower bound $\mathcal{P}$ for P, and the deeper the subdivision levels q_0 and q_∞ are taken, the better the approximation of the support of P is obtained. We begin with introducing some terminology and

notation necessary to formulate Theorem 6 about convergence of our method, which will then be followed by its detailed proof.

For convenience, we are going to use the "maximum" metric d in $\mathbf{R}^n$ throughout this section: $d(x,y) = \max\{|x_i - y_i| : i = 1,\dots,n\}$. To shorten the notation, any product of n intervals $I_1 \times \cdots \times I_n \subset \mathbf{R}^n$ will be called a *box*. We will use its *diameter* in the metric d to measure how large it is, and its *inner size* defined as the minimum of the lengths of the intervals $I_1,\dots,I_n$ to know a lower bound for how much it covers.

Given $\varepsilon,\kappa > 0$, we say that a set $A \subset \mathbf{R}^n$ is (ε,κ)-*thick* if for every positive $\rho \le \varepsilon$, every point $x \in A$ is within the distance of at most ρ/κ from some box of inner size ρ contained in A. We say that a property P is *continuous* (with the constants ε,κ) if its support is (ε,κ)-thick. Intuitively, a set A is thick if its every point is close to a considerable part of its interior; in particular, A does not have any isolated points (or small connected components), whiskers with empty interior, or even thin cusps. Note that the pair (ε,κ) may not be determined uniquely for A, and κ can be usually increased if ε is taken smaller. For example, the unitary disk in the plane is (ε,κ)-thick with any $\varepsilon \le \sqrt{2}$ and $\kappa \le \big(4 + 2\sqrt{4-\varepsilon^2}\big)/\varepsilon$, while a rectangle is (ε,κ)-thick with any $\varepsilon > 0$ not exceeding its inner size and any $\kappa > 0$. If A is (ε,κ)-thick for some $\varepsilon > 0$ and an arbitrarily large $\kappa > 0$, like in the case of the rectangle, then we say that A is (ε,∞)-*thick*.

Let A denote the support of a given property P on R. Given $\lambda,\eta > 0$, we say that a lower estimate $\mathcal{P}$ for P on R is (λ,η)-*accurate* if for every representable rectangular set $Q \subset R$ whose diameter is smaller than λ and whose distance from ∂A is larger than η, the property $\mathcal{P}(Q)$ is satisfied if and only if $Q \subset A$. Intuitively, if the diameter of the boxes on which $\mathcal{P}$ is evaluated is small enough and these boxes are far enough form the boundary of A, then the results provided by $\mathcal{P}$ are meaningful not only if $\mathcal{P}(Q) = 1$, but also if $\mathcal{P}(Q) = 0$. Note that if $\mathcal{P}$ is computed using a formula for P evaluated in interval arithmetic then it is reasonable to expect that $\mathcal{P}$ is (λ,η)-accurate for some $\lambda,\eta > 0$. This is because the computation of $\mathcal{P}(Q)$ usually has some "safety margins," so the errors coming from overestimates in the computation of $\mathcal{P}(Q)$ on small boxes (whose diameter is limited by λ) are typically small enough to provide the accurate result if the box Q is at some distance (bounded from below by η) from ∂A.

If I is a representable interval then the *inaccuracy of* $\mathcal{R}$ *in* I is defined as the maximal distance of points in all $\mathcal{E}_q(I)$ for $q \ge 1$ from the actual values which follow from the formula (1). In practice, if μ_1 is the maximal distance between consecutive representable numbers in I and the relative error due to rounding the results of arithmetic operations does not exceed μ_2 then the inaccuracy μ of $\mathcal{R}$ in $I = [a,b]$ satisfies the inequality $\mu \le 2(b-a)\mu_2 + \mu_1$. For a representable rectangular set $B = I_1 \times \cdots \times I_n \subset \mathbf{R}^n$, the *inaccuracy of* $\mathcal{R}$ *in* B is defined as the maximum of the inaccuracies of $\mathcal{R}$ in each I_j for $j = 1,\dots,n$. Generally speaking, this inaccuracy reflects the finiteness of the set of floating-point numbers that can be represented using a chosen encoding method. A non-zero inaccuracy is caused by the need to round results of arithmetic operations to nearby representable numbers. The number μ provides an upper bound for the shift caused by rounding.

As a measurement of how well one set approximates another, we use the *Hausdorff distance* with respect to the metric d in $\mathbf{R}^n$, defined for $A,B \subset \mathbf{R}^n$ as

$$d_H(A,B) := \max\Big\{ \sup_{a\in A} \inf_{b\in B} d(a,b), \sup_{b\in B} \inf_{a\in A} d(a,b) \Big\}.$$

The following theorem says that given any continuous property P on R, Algorithm 4 returns an arbitrarily close lower bound $A_\approx$ for A, provided that the precision of the computations is high enough in terms of the set of representable

numbers $\mathcal{R}$, a lower estimate $\mathcal{P}$ for P on R, and the initial (q_0) as well as the final (q_∞) depth of subdivisions, all of these parameters given explicitly. In order to simplify the reasoning, we assume that the box R is a hypercube which has the same representable intervals for its edges.

Theorem 6. *Let $I \subset \mathbf{R}$ be a representable interval of length $r > 0$. Let P be a continuous property on $R := I^n$ with the constants $\varepsilon, \kappa > 0$. Denote its support by A. Let $\mathcal{P}$ be a lower estimate for P on R. Let $A_\approx$ denote the set returned by Algorithm 4 applied to R, $\mathcal{P}$, $q_0 \geq \log_2(r/\varepsilon) + 2$, and some $q_\infty > q_0$. Set $\varepsilon_0 := 2^{-q_0+2}r$. Let $\delta < \varepsilon_0$ be an arbitrarily small positive number.*

If $\mathcal{P}$ is (λ, η)-accurate for some $\lambda > 0$ and $\eta < \delta/4$, if $q_\infty > \max\{\log_2(r/\delta) + 4, \log_2(1/\varepsilon_0) + 4, \log_2(r/\lambda) + 2\}$, and if the inaccuracy μ of $\mathcal{R}$ in R is smaller than $\min\{2^{-q_\infty}r, \delta/(4q_\infty)\}$, then $A_\approx \subset A$ and $d_H(A, A_\approx) \leq \varepsilon_0/\kappa + \delta$.

Note that if the support A of P is (ε, ∞)-thick then the Hausdorff distance between A and $A_\approx$ in this theorem is in fact bounded by δ alone, which implies that the set $A_\approx$ gets gradually closer to A with the progress of calculations (or with increasing q_∞) started at some admissible subdivision level q_0. However, if A is not (ε, ∞)-thick then the final proximity between $A_\approx$ and A also depends on the density of the initially tested probes, indicated by q_0.

Before we prove this theorem, let us make some observations about Algorithm 4 applied to R, $\mathcal{P}$, q_0 and q_∞ under the assumptions of Theorem 6. To shorten the notation, define $A^q_\approx$ as part of the computed approximation $A_\approx$ of A built of boxes in all $\mathcal{A}_{q'}$ up to the subdivision level q, that is, $A^q_\approx := \bigcup_{q'=q_0}^{q} \bigcup \mathcal{A}_{q'}$. The following lemma establishes what happens to boxes in the vicinity of good probes.

Lemma 7. *If $Q \in \mathcal{G}_q$ for some $q \in \{q_0, \ldots, q_\infty - 2\}$ then for every box $U \in \mathcal{S}_q$ such that $Q \in U$ the following holds: Either $U \subset A^q_\approx$, or the box $W \in \mathcal{S}_{q+1}$ such that $Q \in W \subset U$ is added to $\mathcal{W}_{q+1}$ in the run of Algorithm 4. In the latter case, $Q \in \mathcal{G}_{q+1}$.*

Proof. We prove this lemma by induction with respect to q. To prove the lemma for $q = q_0$, let us notice that the probes in $\mathcal{E}_{q_0}$ can only be added to $\mathcal{G}_{q_0}$ by the call to `Accept` $(Q, q_0, 1)$ while processing the initial set of probes. Then Q is added to $\mathcal{G}_{q_0+1}$ and all the boxes in $\mathcal{S}_{q_0+1}$ containing Q are added to $\mathcal{W}_{q_0+1}$.

Let us now proceed with the inductive step and assume that the lemma holds true for some $q \in \{q_0, \ldots, q_\infty - 3\}$. Consider a probe $Q \in \mathcal{G}_{q+1}$. It was added to $\mathcal{G}_{q+1}$ in one of the following situations: (1) Q was accepted as a good probe in $\mathcal{E}_q$ by the call to `Accept` $(Q, q, 1)$; (2) a failure box in $\mathcal{W}_q$ containing Q was accepted and $Q \in \mathcal{G}_q$; (3) Q is a corner of a box $V \in \mathcal{W}_{q+1}$ for which the verification was successful.

In the first two cases, we can use the inductive assumption on Q, because $Q \in \mathcal{G}_q$. Consider an arbitrary box $U_{q+1} \in \mathcal{S}_{q+1}$ such that $Q \in U_{q+1}$. Let U_q be the box in $\mathcal{S}_q$ such that $U_{q+1} \subset U_q$. Note that Q is a corner of U_q, so by the inductive assumption, either U_q is contained in $A^q_\approx$, and then $U_{q+1} \subset A^{q+1}_\approx$, or U_{q+1} is added to $\mathcal{W}_{q+1}$ in the algorithm's run. In the latter case, there are two possibilities. Either $U_{q+1} \in \mathcal{W}_{q+1}$ is processed and the result is positive, in which case $U_{q+1} \in \mathcal{A}_{q+1}$, and thus $U_{q+1} \subset A^{q+1}_\approx$, or U_{q+1} is determined to be a failure box in one of the following ways: either by computing $\mathcal{P}(U_{q+1}) = 0$, or by finding a negative probe at one of its corners. In both of these negative cases, a call to `Accept` $(U_{q+1}, q+1, 0)$ is made, and the box $U_{q+2} \subset U_{q+1}$ which contains the good probe $Q \in \mathcal{G}_{q+1}$ is added to $\mathcal{W}_{q+2}$, with Q being added to $\mathcal{G}_{q+2}$ at the same time. This proves the inductive step in the cases (1) and (2).

In the case (3), a call to `Accept` $(Q, q+1, 1)$ is made. As a result, Q is added to both $\mathcal{G}_{q+1}$ and $\mathcal{G}_{q+2}$, and every box $U \in \mathcal{S}_{q+2}$ which is not contained in the part

of $A_{\approx}^{q+2}$ constructed so far is added to $\mathcal{W}_{q+2}$. This proves the inductive step in the case (3).

Applying the induction principle completes the proof. □

The core of the proof of Theorem 6 is contained in the following lemma, in which we prove that any point in any sufficiently large box contained in A (but not too close to ∂A) is closely approached with points in $A_{\approx}$, or at least with good probes.

Lemma 8. *Under the assumptions of Theorem 6, if $B \subset A$ is a box of inner size $\varepsilon_0/2$ such that $d(B, \partial A) > \eta$ and if $x \in B$, then for each $q \in \{q_0, \dots, q_\infty - 2\}$ there exists $x_q \in \mathcal{E}_q \cap B$ such that $d(x_q, x) < 2^{-q} r + 2\mu(q+1-q_0)$, and either $x_q \in \operatorname{int} A_{\approx}^q$, or $x_q \in \mathcal{G}_q \cap \mathcal{G}_{q+1}$ and all the boxes in $\mathcal{S}_{q+1}$ which have x_q but are not contained in $A_{\approx}^q$ are added to $\mathcal{W}_{q+1}$ in the run of Algorithm 4.*

Proof. Let us prove this lemma by induction with respect to q. Consider $q = q_0$. Since the inaccuracy of $\mathcal{R}$ in R is μ and by the assumption on q_0, the maximal distance between the initially tested probes in $\mathcal{E}_{q_0}$ does not exceed $2^{-q_0} r + 2\mu < 2^{-q_0} r + 2 \cdot 2^{-q_\infty} r \le \varepsilon_0/4 + \varepsilon_0/4 = \varepsilon_0/2$. Therefore, $B \cap \mathcal{E}_{q_0} \neq \emptyset$. Let x_{q_0} denote the closest point in $\mathcal{E}_{q_0} \cap B$ to x. Note that obviously $d(x, x_{q_0}) < 2^{-q_0} r + 2\mu$. Since $x_{q_0} \in B \subset A$ and $d(x_{q_0}, \partial A) > \eta$, it follows from the fact that $\mathcal{P}$ is (λ, η)-accurate that the function `Process` called with (x_{q_0}, q_0) returns 1, and thus $x_{q_0} \in \mathcal{G}_{q_0}$. Obviously, in that case x_{q_0} is also added to $\mathcal{G}_{q_0+1}$, and all the boxes in $\mathcal{S}_{q_0+1}$ which contain x_{q_0} are added to $\mathcal{W}_{q_0+1}$.

Let us now prove the inductive step. Assume that the statement in the lemma holds true for some $q \in \{q_0, \dots, q_\infty - 3\}$. Let $x_q \in \mathcal{E}_q$ be the point satisfying this statement for this q. Let $\mathcal{B}_{q+1} \subset \mathcal{S}_{q+1}$ be the set of those boxes which contain x_q as one of their corners. We are going to show which point to choose for x_{q+1} so that the statement is satisfied for $q+1$ with x_{q+1}.

First consider the case in which $x \in \bigcup \mathcal{B}_{q+1}$. Then $d(x_q, x) \le 2^{-(q+1)} r + 2\mu < 2^{-(q+1)} r + 2\mu(q + 2 - q_0)$, and we can take $x_{q+1} := x_q$. If $x_q \in \operatorname{int} A_{\approx}^q$ then $x_{q+1} = x_q \in \operatorname{int} A_{\approx}^{q+1}$ because $A_{\approx}^q \subset A_{\approx}^{q+1}$. Otherwise, by the inductive assumption, $x_{q+1} = x_q \in \mathcal{G}_q \cap \mathcal{G}_{q+1}$ and by Lemma 7 applied to $Q \in \mathcal{G}_{q+1}$, either all the boxes in $\mathcal{B}_{q+1}$ are contained in $A_{\approx}^{q+1}$, in which case $x_{q+1} \in \operatorname{int} A_{\approx}^{q+1}$, or $x_{q+1} \in \mathcal{G}_{q+2}$ and all the boxes in $\mathcal{S}_{q+2}$ which contain x_{q+1} but are not contained in $A_{\approx}^{q+1}$ are added to $\mathcal{W}_{q+2}$ in the run of Algorithm 4. This completes the proof of the inductive step in the case in which $x \in \bigcup \mathcal{B}_{q+1}$.

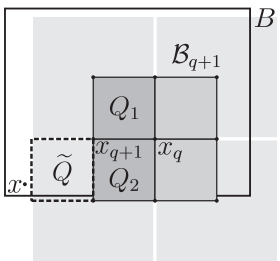

FIGURE 3. Objects that appear in the geometric reasoning in the inductive step in the proof of Lemma 8.

Now consider the complementary case (see Figure 3). Let $\mathcal{C}_{q+1}$ denote the set of all the vertices of the boxes in $\mathcal{B}_{q+1}$. Let x_{q+1} be an element of $\mathcal{C}_{q+1} \cap B$ that minimizes the distance from x. If $x_{q+1} = x_q$ (which can happen in rare situations, because of uneven rounding to representable numbers) then the reasoning in the previous case applies. Otherwise $x_{q+1} \neq x_q$. If x and x_{q+1} belong to some common

box $\widetilde{Q} \in \mathcal{S}_{q+1}$ then $d(x, x_{q+1}) < 2^{-(q+1)}r + 2\mu \leq 2^{-(q+1)}r + 2\mu(q+2-q_0)$. Otherwise, x is closer to x_{q+1} than to x_q at least by the inner size of one box, that is, $d(x, x_{q+1}) < d(x, x_q) - \left(2^{-(q+1)}r - 2\mu\right) \leq \left(2^{-q}r + 2\mu(q+1-q_0)\right) - \left(2^{-(q+1)}r - 2\mu\right) = 2^{-(q+1)}r + 2\mu(q+2-q_0)$. This proves the statement in the aspect of the distance between x and x_{q+1}. Let us now prove the remainder of this statement.

If $x_q \in \operatorname{int} A^q_{\approx}$ then $\bigcup \mathcal{B}_{q+1} \subset \operatorname{int} A^q_{\approx}$ because $A^q_{\approx}$ is built of boxes with respect to a coarser grid than those in $\mathcal{B}_{q+1}$, and thus $x_{q+1} \in \operatorname{int} A^{q+1}_{\approx}$. Otherwise, $x_q \in \mathcal{G}_q \cap \mathcal{G}_{q+1}$ and all the boxes in $\mathcal{B}$ which are not contained in $A^q_{\approx}$ are added to $\mathcal{W}_{q+1}$, by the inductive assumption. Consider all the boxes $Q_1, \ldots, Q_k \in \mathcal{B}_{q+1}$ which contain both x_q and x_{q+1}. If all of them are contained in $A^q_{\approx}$ then $x_{q+1} \in \operatorname{int} A^q_{\approx}$, because $A^q_{\approx}$ is built of boxes with respect to a coarser grid than those in $\mathcal{S}_{q+1}$. Otherwise at least one of those boxes, say, Q_i is added to $\mathcal{W}_{q+1}$ in the run of Algorithm 4. Two possibilities happen later: Either $\mathcal{P}(Q_i)$ is computed and the result is successful, in which case x_{q+1} is added to $\mathcal{G}_{q+1}$ (unless it was added there before), or Q_i is determined to be a failure box either by computing $\mathcal{P}(Q_i) = 0$ or by detecting a negative probe at one of its corners (different from x_{q+1}), in which case x_{q+1} is added to $\mathcal{T}_{q+1}$ (unless it was added to $\mathcal{G}_{q+1}$ before). Note that the verification of P on x_{q+1} results in computing $\mathcal{P}(x_{q+1}) = 1$, which follows from the (λ, η)-accuracy of $\mathcal{P}$, because $x_{q+1} \in B$ and thus $d(x_{q+1}, \partial A) > \eta$. In this way, in both cases $x_{q+1} \in \mathcal{G}_{q+1}$, and by Lemma 7, either all the boxes in $\mathcal{S}_{q+1}$ to which x_{q+1} belongs are contained in $A^{q+1}_{\approx}$, and thus $x_{q+1} \in \operatorname{int} A^{q+1}_{\approx}$, or $x_{q+1} \in \mathcal{G}_{q+2}$ and all the boxes in $\mathcal{S}_{q+2}$ which contain x_{q+1} but are not contained in $\mathcal{A}^{q+1}_{\approx}$ are added to $\mathcal{W}_{q+2}$. This ends the proof of the second case in the inductive step, and also of the entire inductive step.

Applying the induction principle completes the proof. □

We are now ready to prove Theorem 6.

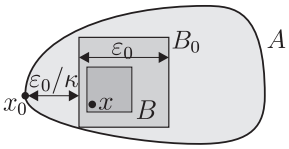

FIGURE 4. Geometric idea behind the proof of Theorem 6: We show that for every $x_0 \in A$ there exists an x at the distance of $\varepsilon_0/\kappa + \delta/2$ from x in a box $B \subset A$ of inner size $\varepsilon_0/2$ at the distance at least η from ∂A, and $d(x, A_{\approx})$ does not exceed $\delta/2$.

Proof of Theorem 6. Let us begin with the easy observation that $A_{\approx} \subset A$ because in Algorithm 4 a box Q is only added to $\mathcal{A}$ if $\mathcal{P}(Q) = 1$. Moreover, $\mathcal{S}_q \neq \emptyset$ for all $q < q_\infty$ because the inaccuracy μ of $\mathcal{R}$ in R is assumed to be below $2^{-q_\infty}r$, so the length of each edge of each box in each $\mathcal{S}_q$ for $q < q_\infty$ is at least $2^{-(q_\infty - 1)}r - 2\mu > 2^{-(q_\infty-1)}r - 2 \cdot 2^{-q_\infty}r = 0$.

Let $x_0 \in A$. (See Figure 4 for a geometric illustration of this part of the proof.) By the (ε, κ)-continuity of P, there exists a box B_0 of inner size ε_0 such that $B_0 \subset A$ and $d(x_0, B_0) \leq \varepsilon_0/\kappa$ (because $\varepsilon_0 < \varepsilon$). It is a simple geometric argument to see that there exists another box $B \subset B_0$ of inner size $\varepsilon_0/2$ such that $d(B, \partial A) \geq \delta/2 > \eta$ and $d(x_0, B) \leq \varepsilon_0/\kappa + \delta/2$. Take any $x \in B$ such that $d(x_0, x) \leq \varepsilon_0/\kappa + \delta/2$. We will show that there exists $x_{\approx} \in A_{\approx}$ such that $d(x, x_{\approx}) < \delta/2$. Since x is chosen arbitrarily, this will imply that every element of A is at the distance smaller than δ from some element of $A_{\approx}$, and thus $d_H(A, A_{\approx}) \leq \varepsilon_0/\kappa + \delta$.

Let us apply Lemma 8 to x and B, and consider $x_{q_\infty-2}$ whose existence follows from that lemma. We will show that $x_{q_\infty-2} \in A_{\approx}$. If $x_{q_\infty-2} \in A_{\approx}^{q_\infty-2}$ then this conclusion is trivial. Otherwise, $x_{q_\infty-2} \in \mathcal{G}_{q_\infty-1}$ and all the boxes in $\mathcal{S}_{q_\infty-1}$ which contain $x_{q_\infty-2}$ are added to $\mathcal{W}_{q_\infty-1}$ in Algorithm 4, because none of these boxes is contained in $A_{\approx}^{q_\infty-2}$. Note that at least one of these boxes, denoted further by Q, is fully contained in B, because the inner size of B is $\varepsilon_0/2$, and the diameter of these boxes is at most $2^{-(q_\infty-1)}r + 2\mu < 2^{-(q_\infty-1)} + 2 \cdot 2^{-q_\infty} r = 2^{-(q_\infty-2)}r < \varepsilon_0/4$. Since the diameter of Q does not exceed $2^{-(q_\infty-2)}r < \lambda$ and $d(Q, \partial A) \geq d(B, \partial A) > \eta$, it follows from the (λ, η)-accuracy of $\mathcal{P}$ that $\mathcal{P}(Q) = 1$, and therefore $Q \in \mathcal{A}_{q_\infty-1}$. Thus $x_{q_\infty-1} \in Q \subset A_{\approx}$.

To complete the proof, take $x_{\approx} := x_{q_\infty-2}$ and note that $d(x, x_{q_\infty-2}) < 2^{-(q_\infty-2)}r + 2\mu(q_\infty + 1 - q_0) < \delta/4 + \delta/4 = \delta/2$. □

We would like to remark that, although we do not prove it here, in the actual applications the constructed set $A_{\approx}$ usually fills A quite well, and $A \setminus A_{\approx}$ seems to be contained in a relatively narrow belt along ∂A.

7. Examples and applications

In this section we show some examples and applications of the parallelization method introduced in this paper.

We first consider a very elementary demonstration which illustrates the idea of representing an area of interest in terms of boxes with gradually decreasing grid size, as well as the idea of a lower approximation of this area. Then we analyze a nontrivial application of the method introduced in this paper. In this application, a rigorous lower bound is obtained for the set of those parameters of an overcompensatory nonlinear Leslie population model for which there exist at least two attractors.

7.1. Approximations of a ring.

In this subsection we discuss a simple demonstration of the method introduced in this paper. We use the parallelization framework to compute an approximation of the ring

$$A := \big\{(x, y) \in R : 500 \leq x^2 + y^2 \leq 3400\big\},$$

where $R = [-64, 64] \times [-64, 64] \subset \mathbf{R}^2$. The function `Process` simply calculates $x^2 + y^2$ for all the vertices of Q and returns 1 iff all the values are within the prescribed range. A few subsequent approximations of this ring computed with the increasing upper bound q_∞ on the subdivision depth are illustrated in Figure 5.

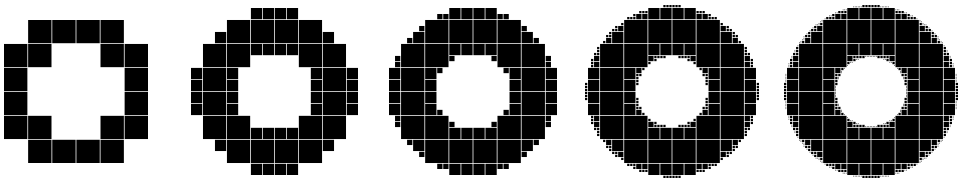

Figure 5. Subsequent lower approximations of the ring A in R computed for $q_\infty = 4, \ldots, 8$, respectively. Space is added between adjacent boxes in order to illustrate the structure of each set.

As one can see, the ring is gradually better approximated. Its interior is subsequently filled by adding boxes of decreasing size to the previously computed approximation. In comparison to the "naive" approach of computing $\mathcal{P}(Q)$ for all

$Q \in \mathcal{S}_{q_\infty - 1}(R)$, which would result in the same set $A_{\approx}$, the gain in the computational time is profound and is listed in the table below. The number of probes and boxes is indicated on which $\mathcal{P}$ is evaluated in each run of the algorithm.

q_∞	# probes	# boxes	$\#\mathcal{S}_{q_\infty - 1}(R)$	time savings
4	9	32	64	36%
5	26	108	256	48%
6	99	236	1,024	67%
7	324	452	4,096	81%
8	592	1,000	16,384	90%

7.2. **Application to a population model.** As a sample serious application of the parallelization method introduced in this paper, we compute a lower bound for the region A in the parameter space $R \subset \mathbf{R}^2$ for which a given dynamical system exhibits two attractors.

Consider the overcompensatory Leslie population model in $\mathbf{R}^2$ analyzed numerically in [11], which is generated by the function

$$f(x_1, x_2; \theta_1, \theta_2, \lambda, p) = \big((\theta_1 x_1 + \theta_2 x_2)\, \mathrm{e}^{-\lambda(x_1+x_2)}, p x_1\big), \tag{2}$$

where $\theta_1, \theta_2, \lambda, p$ are some parameters. It has been shown in [11] that $\theta_1, \theta_2 \in [10, 35]$, $\lambda = 0.1$, and $p = 0.7$ are meaningful values of the parameters, and thus we will restrict our attention to $(\theta_1, \theta_2) \in R := [10, 35]^2 \subset \mathbf{R}^2$ and λ, p as above. Although our method is dimension-independent and equally well we would be able to consider also p changing in some range, for instance $p \in [0.5, 0.9]$ (as suggested in [2]), or to discuss a higher-dimensional Leslie model (as given in [11]), we prefer to stay with dimension 2 for the sake of ease and clarity of visual presentation of the results.

Using interval arithmetic and algorithms introduced in [3] (see also [8]), it is possible to analyze the global structure of the dynamics for entire boxes of parameters $B \subset R$ by computing upper bounds for a Morse decomposition. The idea is to isolate the recurrent dynamics and determine the gradient-like behavior of the remaining trajectories (see [6] for details).

In this method, an a priori fixed rectangular area in the phase space that is assumed (or known) to contain the dynamics of interest is subdivided into a finite number of boxes with respect to some fixed uniform rectangular grid. Then an outer bound of each box is computed in interval arithmetic [9] using the formula for the generator of the dynamical system. The resulting set is then covered with boxes in the phase space. In this way, a multivalued mapping between boxes is created, which is represented by means of a directed graph. Strongly connected path components in this graph correspond to isolating neighborhoods of Morse sets [3], [8], and paths between them in the graph provide a bound for possible connecting orbits in the underlying dynamical system.

In [2], comprehensive analysis of the dynamics using this technique is conducted for a fixed subdivision $\mathcal{S}_q(R)$ of the set R. This method can provide lower bounds for sets of parameters for which the dynamics has some specific properties. In particular, the existence of multiple isolating neighborhoods that are mapped to itself translates to the existence of multiple attraction basins, and thus the existence of at least two attractors in the sense of Conley [6]. In this paper, we will focus on finding a much more accurate rigorous lower bound A for the set of all the parameters in R for which this happens.

We have chosen this particular property of existence of multiple attractors because of its importance from the biological point of view. Namely, the existence of more than one attractor in the system contradicts the commonly held belief that

the initial conditions of a population are not important for its asymptotic behavior, because it will eventually stabilize at the equilibrium. In this case, however, the system exhibits at least two different equilibrium states (corresponding to the attractors), so the initial conditions actually do matter and cannot be ignored.

Although most likely the property which we analyze in this subsection is not continuous (as defined in Section 6), our method can be applied to construct *some* lower bound for the set on which this property is satisfied, and to make gradually better approximations of this set substantially more effectively than by direct application of the technique introduced in [2].

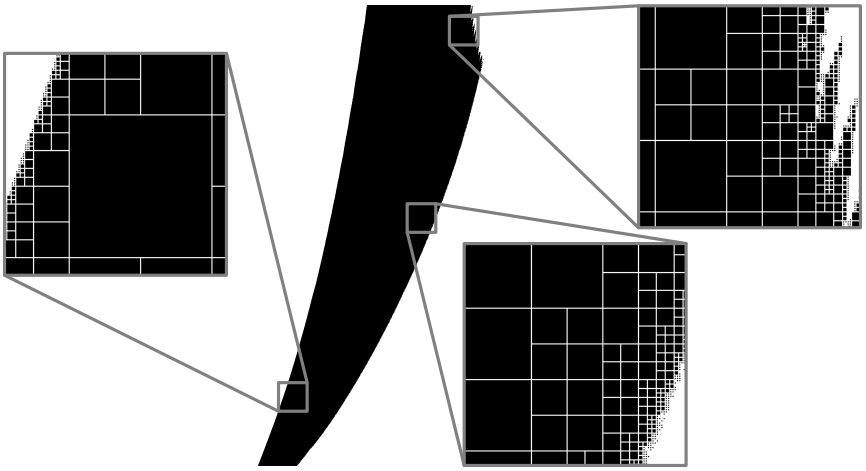

FIGURE 6. A rigorous lower bound for the set of parameters $(\theta_1, \theta_2) \in [10, 35]^2$ for which the Leslie model (2) exhibits multiple attractors, with three close-ups revealing its structure.

Figure 6 shows the set constructed for $q_\infty = 12$ and a few close-ups in which extra space between adjacent boxes was added in order to show the structure of the set $A_\approx$. The time savings depending on various tested values of q_∞ show similar pattern to the ones observed for the example analyzed in the previous subsection, with more than 99% savings at $q_\infty = 12$.

Note that at some close-ups one can notice a few boxes $Q \subset A$ which do not satisfy the property $\mathcal{P}$, but all their sub-boxes do satisfy $\mathcal{P}$, and such boxes are only located at a distance from the boundary of the computed area which is small in comparison to the size of these boxes. This confirms empirically the fact that overestimates arising from checking the property $\mathcal{P}$ with a formula for P using interval arithmetic are most harmful close to ∂A. Moreover, we speculate that the complicated shape of the top right-hand part of the computed set $A_\approx$ is also due to the overestimates which are more harmful at some areas than at others. We are inclined to believe that the complicated structure that can be seen in this area is spurious, and we expect that A actually has smooth shape that can be deduced by extrapolation from the remaining part of A. Our belief is justified by the fact that many holes and gaps in the constructed set are gradually filled in as smaller and smaller boxes are processed. One can see this at the close-up in the top right-hand corner of Figure 6.

The source code of the program that was used to conduct the computations described above is available at [10], together with the results of these computations. This program uses the CAPD software library [4] for an implementation of the

interval arithmetic, and the CHomP software library [5] for the parallel computations framework as well as for the data structures for representing the constructed set $A_{\approx}$.

Acknowledgments

The author would like to express his gratitude to Prof. Konstantin Mischaikow for the inspiration to undertake this work, and for the suggestion of the title for this paper. Also a thorough review and constructive remarks made by one of the reviewers are gratefully acknowledged.

References

[1] Z. Arai, On hyperbolic plateaus of the Hénon map, Experiment. Math. 16 (2007), 181–188.
[2] Z. Arai, W. Kalies, H. Kokubu, K. Mischaikow, H. Oka and P. Pilarczyk, Databases for the global dynamics of multi-parameter systems, SIAM J. Appl. Dyn. Syst., accepted.
[3] H. Ban and W. Kalies, A computational approach to Conley's decomposition theorem, Journal of Computational and Nonlinear Dynamics 1 (2006), 312–319.
[4] Computer Assisted Proofs in Dynamics, http://capd.ii.uj.edu.pl/.
[5] Computational Homology Project, http://chomp.rutgers.edu/.
[6] C. Conley, Isolated Invariant Sets and the Morse Index, CBMS Regional Conference Series in Math., no. 38, Amer. Math. Soc., Providence, RI, 1978.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, second edition, 2001.
[8] W. D. Kalies, K. Mischaikow and R. C. A. M. VanderVorst, An algorithmic approach to chain recurrence, Found. Comput. Math. 5 (2005), 409–449.
[9] R. E. Moore, Interval Analysis, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966.
[10] P. Pilarczyk, Parallelization method for a continuous property. Software and examples, http://www.pawelpilarczyk.com/parallel/.
[11] I. Ugarcovici, H. Weiss, Chaotic dynamics of a nonlinear density dependent population model, Nonlinearity 17 (2004), 1689–1711.

Paweł Pilarczyk, Universidade do Minho, Centro de Matemática, Campus de Gualtar, 4710-057 Braga, Portugal
URL: http://www.pawelpilarczyk.com/