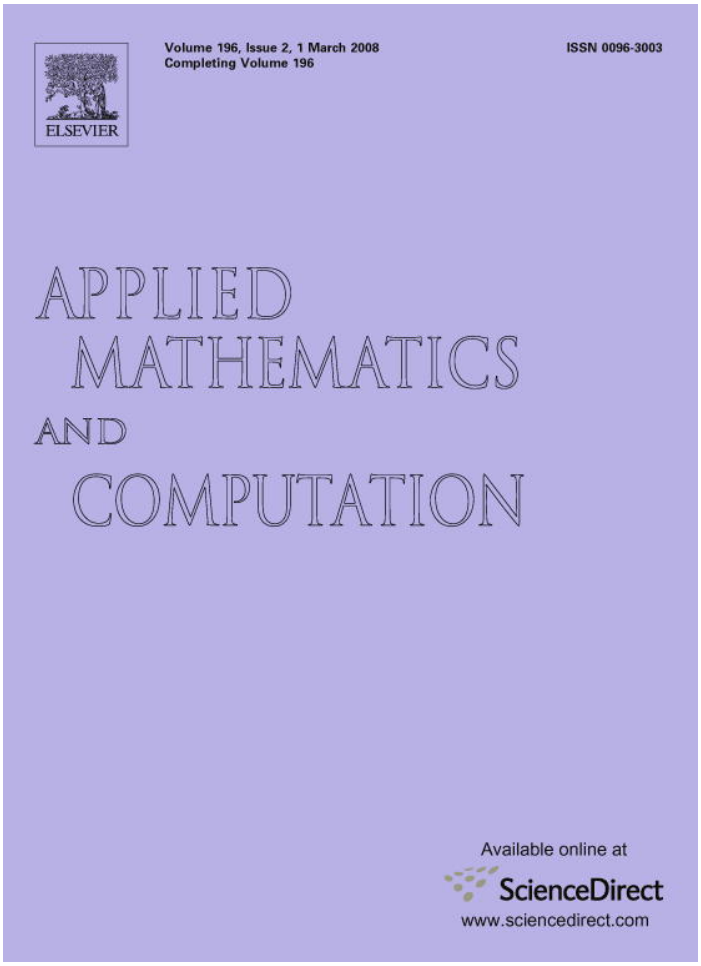

Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 196 (2008) 666–678
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

# A reliable algorithm for computing the topological degree of a mapping in $R^2$

K.H. Ko [a,*], T. Sakkalis [b], N.M. Patrikalakis [c]

[a] Department of Mechatronics, Gwangju Institute of Science and Technology, 1 Oryong-dong, Buk-gu, Gwangju 500-712, Republic of Korea
[b] Mathematics Laboratory, Agriculture University of Athens, Athens 118 55, Greece
[c] Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

**Abstract**

In this paper, we present a method to reliably compute the topological degree of a mapping in the plane, over a simple closed polygon. The method is based on Henrici's argument principle, and computes the degree using the winding number concept in range arithmetic. The algorithm is then applied to the root computation of a univariate polynomial. The proposed algorithms are demonstrated with examples.


*Keywords:* Gauss map; Topological degree; Principle of argument; Interval arithmetic; Affine arithmetic; Univariate polynomials

## 1. Introduction

Let $p(x,y)$ and $q(x,y)$ be real continuous functions. We consider the vector field $\mathbf{F}$ as follows:

$$\mathbf{F}:\mathbf{R}^2\to\mathbf{R}^2,\quad \mathbf{F}(x,y)=(p(x,y),q(x,y)). \tag{1}$$

A zero $\mathbf{z}_0$ of $\mathbf{F}$ is a tuple $(x_0,y_0)$ of real numbers such that $\mathbf{F}(x_0,y_0)=(0,0)$. In this paper we will also regard a pair of real numbers $(x,y)$ as a complex number $\mathbf{z}$ and write $\mathbf{z}=(x,y)=x+\mathrm{i}y$.

Let $A$ be a rectangle in the $xy$ plane defined by $a_1\leqslant x\leqslant b_1$ and $a_2\leqslant y\leqslant b_2$. Assume that no zero of $\mathbf{F}$ lies on the boundary $\partial A$. In that case, we will call such an $A$ compatible with $\mathbf{F}$.

If $\mathbf{F}$ and $A$ are as above, we may define the Gauss map $\mathbf{G}$ as follows:

$$\mathbf{G}:\partial A\to S^1,\quad \mathbf{G}=\frac{\mathbf{F}}{|\mathbf{F}|}, \tag{2}$$

where $S^1$ is the unit circle. Since $|\mathbf{F}|\neq 0$ on $\partial A$, $\mathbf{G}$ is continuous.

* Corresponding author.
*E-mail address:* khko@gist.ac.kr (K.H. Ko).

0096-3003/$ - see front matter 
doi:10.1016/j.amc.2007.07.002

Assume that both $\partial A$ and $S^1$ carry the counterclockwise orientation. Then, the degree $d$ of $\mathbf{G}$ is an integer that indicates how many times $\partial A$ is wrapped around $S^1$ by $\mathbf{G}$, see Fig. 1; for the precise definition see [1]. The value $d$ is also called the topological degree of $\mathbf{G}$, and of $\mathbf{F}$, over $\partial A$. In the above definition, $A$ can be replaced with any simple closed polygon in $\mathbf{R}^2$, where $A$ is homeomorphic to a unit disk. Furthermore, it is well known that the concept of the topological degree can be generalized for a continuous map $\mathbf{F}$ over any simple polytope $P \in \mathbf{R}^n$. In the sequel, unless otherwise stated, for purposes of computation, $\mathbf{F}$ will be a polynomial map and $A$ will be a planar domain homeomorphic to the unit disk whose boundary $\partial A$ is a simple closed curve.

Several different approaches for degree computation have been proposed in the literature. Stenger [2] proposed an algorithm to compute the topological degree of a differentiable mapping $\mathbf{F}$ defined on a connected $n$-dimensional polyhedron, $P$, in $\mathbf{R}^n$ using the sign change of the Jacobian of $\mathbf{F}$ along the boundary of $P$. Stenger's approach is also discussed in Kearfott [3], where an improvement in the computational aspect was made. Stynes [4] simplified the computation process of Stenger's formula [2] by replacing determinant evaluations with a scanning procedure of associated matrices. Stenger's method can compute the correct degree under the assumption that the boundary of $P$ is *sufficiently refined*. The latter, however, limits the applicability of Stenger's method to various problems. Concerning this issue, Stynes [5] discussed the construction of such sufficient refinements of a polygon for Stenger's degree computation method and proposed a class of algorithms for sufficient refinements. However, his approach is based, still, on the knowledge of moduli of continuity and only in the case where such moduli can be estimated, such sufficient refinements can be constructed.

Boult and Sikorski [6] discussed the computation of the topological degree of the class $F$ of Lipschitz functions with constant $K$, with $f : C \to \mathbf{R}^2$, where $C$ is the unit square, the infinity norm of $f$ is larger than zero and $K/4\beta \geqslant 1$. Here $\beta$ is the minimum of the infinite norm of $f$ on the boundary of $C$. They showed that $m^* = 4\lfloor K/4\beta \rfloor$ is a lower bound necessary to compute the degree of $f \in F$ and it provides a sufficient refinement of $\partial C$ such that the Stenger's degree computation method [2] yields a correct degree value. The determination of $K$ or $\beta$ is critical for the robustness of their algorithm for the degree computation. Since the constant $K$ with which the Lipschitz condition is satisfied and/or a lower bound for the infinity norm of the map on the boundary $\partial C$ may not be available *a priori* in actual computation, their algorithm may not work robustly as is demonstrated by the examples in their paper showing that their algorithm may fail to compute the correct degree value of $f$ depending on $K$ and $\beta$. Mourrain et al. [7] discussed the problem of constructing sufficient refinements in the topological degree computation and presented a procedure for computing with certainty the degree using Stenger's method, with emphasis on the 2D case. The main focus of their approach is also placed on a method of how to guarantee sufficient refinements for the degree computation based on Stenger's method. Their method, however, requires isolation of the roots of a univariate polynomial, which is the critical

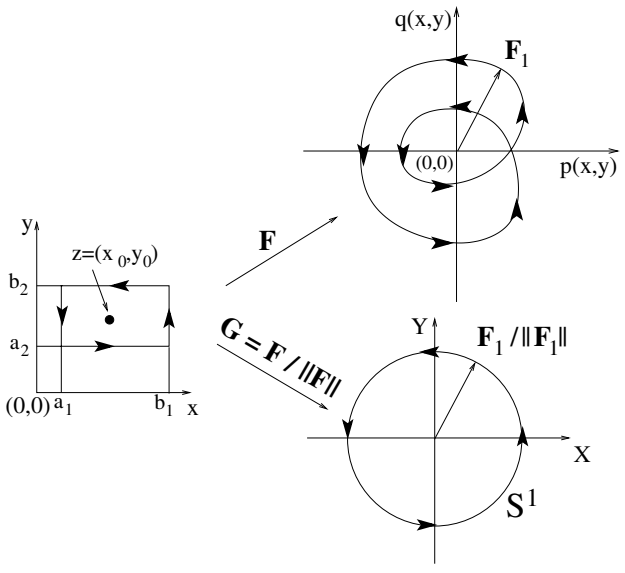


Fig. 1. An illustration of the topological degree of the Gauss map.

step to ensure the correctness of the degree computation. That in itself is, in general, a difficult problem and is not practical in real situations.

Another crucial assumption that Stenger's method makes is that the Jacobian of the map **F** is non-zero at a zero of **F**. This, however, can easily be violated if a root has multiplicity greater than one.

The topological degree, on the other hand, can be calculated without considering sufficient refinements of the boundary and computing the Jacobian of a mapping, which are the basic ingredients of Stenger's method. Sakkalis [1] proposed an algorithm to compute the degree of the Gauss map based on the Cauchy index and the Euclidean algorithm. The degree $d$ of **F** given in Eq. (1), is computed by

$$d = -\frac{1}{2} I_A F, \tag{3}$$

where $I_A F$ is the sum of Cauchy indices along $\partial A$ under the map **F**. He discussed a relation between the Cauchy index, the topological degree and the root counting inside a domain of an analytical function, and provided a solid mathematical procedure for the topological degree computation. The algorithm, however, may not be appropriate for practical purposes since it requires another algorithm to compute roots of Sturm sequences of polynomials along the boundary $\partial A$ for the Cauchy index computation, and that itself is not robust.

Another popular way to compute the topological degree is developed based on the concept of the winding number. Especially, this approach, which is conceptually and computationally simple, has been widely used in computing the poles of a complex polynomial [8].

We denote by

$$\Delta_{\mathbf{z}\in[\mathbf{a},\mathbf{b}]} \arg(\mathbf{F}(\mathbf{z})) \tag{4}$$

the amount of change of the argument of **F** as **z** moves from **a** to **b** continuously along a simple curve with end points **a** and **b**. Assume that we have a domain $A$ and a sequence of discretization points $\mathbf{c}_1, \ldots, \mathbf{c}_K (\mathbf{c}_1 = \mathbf{c}_{K+1})$ on the curve $\partial A$. Then the degree of the Gauss map, $d$, is obtained as follows [1,9]:

$$d = \frac{1}{2\pi} \Delta_{\mathbf{z}\in[\mathbf{c}_1,\mathbf{c}_{K+1}]} \arg(\mathbf{F}(\mathbf{z})). \tag{5}$$

Using the discretization points on $\partial A$, we have

$$d = \frac{1}{2\pi} \sum_{k=1}^{K} \Delta_{\mathbf{z}\in[\mathbf{c}_k,\mathbf{c}_{k+1}]} \arg(\mathbf{F}(\mathbf{z})). \tag{6}$$

Since $\arg(\mathbf{F}(\mathbf{z}))$ is continuous over the interval $[\mathbf{c}_k, \mathbf{c}_{k+1}]$, Eq. (6) can be rewritten as

$$d = \frac{1}{2\pi} \sum_{k=1}^{K} \arg\left(\frac{\mathbf{F}(\mathbf{c}_{k+1})}{\mathbf{F}(\mathbf{c}_k)}\right) \tag{7}$$

under the following condition [10,11,8,12,9]:

$$|[\arg(\mathbf{F}(\mathbf{z}))]|_{\mathbf{z}\in[\mathbf{c}_k,\mathbf{c}_{k+1}]} \leqslant \pi. \tag{8}$$

The robustness of computing (7) is guaranteed under condition (8) since there should exist a unique value of $\frac{\mathbf{F}(\mathbf{c}_{k+1})}{\mathbf{F}(\mathbf{c}_k)}$ with the condition. Based on the argument principle, Henrici [8] provided a numerical algorithm to efficiently compute the winding number without using exact arithmetic.

Condition (8), however, is not easily satisfied in general [8,12,9]. Without the condition being satisfied, Henrici's algorithm gives only a strictly lower bound of the winding number [12]. There have been many efforts to devise an algorithm to verify the validity of this condition. Ying and Katz [9] proposed a method to verify condition (8). They choose a linear function $P(z)$ of the complex variable $z$, given a complex continuous function $f(z)$ over the linear interval $[z_1, z_2]$ and define $R(z) \approx f(z) - P(z)$. If $\min|P(z)| > \max|R(z)|, z \in [z_1, z_2]$, then (8) is guaranteed. However, their approach requires the computation of an upper bound for $|f''|$ along the line segment $[z_1, z_2]$, which involves computing the second derivatives and estimating the upper bound. Davies [11] discussed a way to avoid the direct reference to the derivative of a function to compute the winding number

but failed to address the failsafe way to determine the sufficient number of function evaluations, which is equivalent to failure in satisfying (8). Carpentier and Dos Santos [10] proposed two empirical tests:

$$\left|\arg\left(\frac{\mathbf{F}(z_k)}{\mathbf{F}(z_{k+1})}\right)\right| < \frac{3\pi}{4}, \tag{9}$$

$$\frac{1}{6.1} < \left|\arg\left(\frac{\mathbf{F}(z_k)}{\mathbf{F}(z_{k+1})}\right)\right| < 6.1, \tag{10}$$

that control the discretization density along the boundary $\partial A$. But the choice of the constants is based on experiments and we can find a case where condition (8) does not hold.

In this paper, we propose a method of adaptive discretization of the boundary of a domain to generally guarantee condition (8). For this, we exploit the self-validation of range arithmetic to satisfy (8). We, then, develop an algorithm to reliably compute the degree of the Gauss map using a discretized method based on the concept of argument principle with Eq. (7). This degree computation approach is not affected by the nature of the Jacobian inside the domain, and requires no derivative computation.

In Section 2, we briefly review interval and affine arithmetic and its extension to complex arithmetic. In Section 3, we propose an adaptive discretization method for condition (8) and develop a degree computation algorithm using the proposed method. In Section 4, the proposed algorithm is tested with various numerical examples to demonstrate its robustness and performance. Then in Section 5 the proposed degree computation algorithm is applied to root computation of a univariate polynomial. Open problems and future developments are discussed in Section 6.

## 2. Range arithmetic

Range arithmetic is an arithmetic system which is used for self-validated computation. The result of range arithmetic is an enclosure of the exact value considering uncertainties and errors during evaluation. Examples of range arithmetic include interval arithmetic [13] and affine arithmetic [14]. Both keep track of intervals of ideal values, but the difference between those two arithmetic systems lies in how to control the size of intervals during evaluation.

### 2.1. Interval arithmetic

An interval, $[\zeta]$, is a set of real numbers defined as follows [13]:

$$[\zeta] = [a,b] = \{x | a \leqslant x \leqslant b, a, b \in \mathbf{R}, a \leqslant b\}. \tag{11}$$

Interval arithmetic operations are defined by [13,15]:

$$[a,b] \circ [c,d] = \{x \circ y | x \in [a,b] \text{ and } y \in [c,d]\}, \tag{12}$$

where $\circ$ is one of the basic arithmetic operations for real numbers such as $+$, $-$, $\times$ and $/$. Each arithmetic operation is defined by using the formulae with the endpoints of each interval:

$$\begin{aligned}
&[a,b] + [c,d] = [a+c, b+d],\\
&[a,b] - [c,d] = [a-d, b-c],\\
&[a,b] \times [c,d] = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)],\\
&[a,b]/[c,d] = [\min(a/c,a/d,b/c,b/d), \max(a/c,a/d,b/c,b/d)],
\end{aligned} \tag{13}$$

where the division operation is not defined when $0 \in [c,d]$.

The mathematical definitions of an interval and associated arithmetic assume that real numbers defining intervals are represented with infinite precision. However, when interval arithmetic is implemented on digital computers using floating point arithmetic following the definition of ANSI/IEEE STD 754-1985, *Standard for Binary Floating Point Arithmetic* [16], achieving infinite precision is not possible, leading to violation of the *inclusion property* of interval arithmetic. As a solution to this problem, it is proposed that the IEEE standard

rounding scheme should be controlled carefully during interval operations to generate strict enclosure of an exact result. This rounding scheme is called the *rounded interval arithmetic* [17,13,15].

Rounded interval arithmetic (RIA) performs rounding operations for floating point values conservatively at each arithmetic operation, ensuring the *inclusion* property. For more details, see [17,18,15].

### 2.2. Affine arithmetic

Interval arithmetic is simple to implement and its performance can compete with floating point arithmetic but its significant drawback is overestimation of an interval during evaluation. Such overestimation cannot be controlled within the interval arithmetic framework, resulting in range explosion [14]. As a solution to this problem affine arithmetic is developed, which maintains information of both enclosures and their correlations, providing a mechanism for tighter bounds during evaluation [14].

Affine arithmetic is represented as a first-degree polynomial [14]:

$$\hat{x} = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + \cdots + x_n\epsilon_n, \tag{14}$$

where $x_0$ is the central value of the affine form $\hat{x}$, $\epsilon_i$ an independent component of the total uncertainty of the quantity $x$ with $-1 \leqslant \epsilon_i \leqslant 1$ and $x_i$ the magnitude of that component.

A value in affine form can be easily converted to an interval and *vise versa*. Thus, the final result in affine arithmetic is represented as an interval, which can be readily used for various applications.

### 2.3. Complex range arithmetic

Complex arithmetic can benefit from the concept of range arithmetic, which leads to a special arithmetic system, called the *complex range arithmetic*. A complex range value consists of real and imaginary parts each of which is an enclosure. Depending on the arithmetic systems used for the real and imaginary parts, we have *complex interval arithmetic* and *complex affine arithmetic*, respectively.

Complex interval arithmetic is analyzed in [19]. Given two real range values, $[\zeta]$ and $[\eta]$, the set

$$\mathbf{C} = \{a + \mathrm{i}b | a \in [\zeta], \quad b \in [\eta]\} \tag{15}$$

is defined as the *complex interval*, which forms a rectangle in the complex domain whose sides are parallel to the real and imaginary coordinate axes. A complex interval can also be represented as a circle in the complex domain [19]. Suppose we have two complex intervals, $\mathbf{IA} = [\zeta_1] + \mathrm{i}[\zeta_2]$ and $\mathbf{IB} = [\eta_1] + \mathrm{i}[\eta_2]$. Then, the basic complex interval arithmetic operations are defined as follows [19]:

$$\begin{aligned}
&\mathbf{IA} + \mathbf{IB} = [\zeta_1] + [\eta_1] + \mathrm{i}([\zeta_2] + [\eta_2]),\\
&\mathbf{IA} - \mathbf{IB} = [\zeta_1] - [\eta_1] + \mathrm{i}([\zeta_2] - [\eta_2]),\\
&\mathbf{IA} \times \mathbf{IB} = [\zeta_1][\eta_1] - [\zeta_2][\eta_2] + \mathrm{i}([\zeta_1][\eta_2] + [\zeta_2][\eta_1]),\\
&\mathbf{IA}/\mathbf{IB} = \frac{[\zeta_1][\eta_1] + [\zeta_2][\eta_2]}{[\eta_1]^2 + [\eta_2]^2} + \mathrm{i}\frac{[\zeta_2][\eta_1] + [\zeta_1][\eta_2]}{[\eta_1]^2 + [\eta_2]^2}, \quad 0 \notin [\eta_1]^2 + [\eta_2]^2.
\end{aligned} \tag{16}$$

On the other hand, a complex affine value consists of real and imaginary parts in affine form and the complex affine arithmetic system is defined using affine arithmetic. Each complex affine value can be converted to complex intervals. Therefore, the resulting complex affine value is a rectangle in coordinate axes as in complex interval arithmetic.

In this paper, we focus on the rectangular representation method since it is an intuitive extension of range arithmetic and easy to implement in algorithms proposed in this paper.

## 3. Computation of topological degree

In this section, we present our basic theoretical results, as well as the main algorithms for the degree computation. One of the key ingredients of our method is the argument principle by Henrici [8].

### 3.1. Adaptive discretization for degree computation

#### 3.1.1. Theoretical background

Suppose that we have a map $\mathbf{F} = (p, q)$ and a rectangle $A$ compatible with it. Let $\mathbf{c}_k = (x_k, y_k)$ and $\mathbf{c}_{k+1} = (x_{k+1}, y_{k+1})$ be two points on $\partial A$. Let $\boldsymbol{\gamma}_k$ be the part of $\partial A$ that connects $\mathbf{c}_k$ and $\mathbf{c}_{k+1}$. Assume that $\boldsymbol{\gamma}_k$ carries the counterclockwise orientation and is parametrized by (normalized) arc length $t$ $(0 \leqslant t \leqslant 1)$ with $\boldsymbol{\gamma}_k(0) = \mathbf{c}_k$ and $\boldsymbol{\gamma}_k(1) = \mathbf{c}_{k+1}$. Let $\mathscr{F}^p_{\gamma_k}$ and $\mathscr{F}^q_{\gamma_k}$ be the images of $\gamma_k$ under the maps $p(x, y)$ and $q(x, y)$. Since $p$ and $q$ are continuous and $\gamma_k$ is connected and compact, $\mathscr{F}^p_{\gamma_k}$ and $\mathscr{F}^q_{\gamma_k}$ are intervals, say $\mathscr{F}^p_{\gamma_k} = [a^p_{\gamma_k}, b^p_{\gamma_k}]$ and $\mathscr{F}^q_{\gamma_k} = [a^q_{\gamma_k}, b^q_{\gamma_k}]$. Then we have our first result:

**Theorem 1.** *If* $(0, 0) \notin \mathscr{I} = [a^p_{\gamma_k}, b^p_{\gamma_k}] \times [a^q_{\gamma_k}, b^q_{\gamma_k}]$, *then*

$$|\Delta_{\mathbf{z} \in [\mathbf{c}_k, \mathbf{c}_{k+1}]} \arg(\mathbf{F}(\mathbf{z}))| < \pi. \tag{17}$$

**Proof.** If $(0, 0) \notin \mathscr{I}$, the rectangle $\mathscr{I}$ has to lie across at most two consecutive quadrants in the $pq$ domain. This implies that the angle between any two vectors $\mathbf{F}(\mathbf{c}'_k)$ and $\mathbf{F}(\mathbf{c}'_{k+1})$ in $\mathscr{I}$ with $\mathbf{c}'_k, \mathbf{c}'_{k+1} \in [\mathbf{c}_k, \mathbf{c}_{k+1}]$ should be less than $\pi$. □

#### 3.1.2. Algorithm

For a rectangle $A$ compatible with $\mathbf{F}$ we can choose $\mathbf{c}_k$ and $\mathbf{c}_{k+1}$ such that the condition of Theorem 1 is always satisfied. Based on this, we propose an algorithm for computing the topological degree as follows:

(1) Let $\mathbf{c}_1, \ldots, \mathbf{c}_{K+1} = \mathbf{c}_1$ be a sequence of points on $\partial A$, arranged counterclockwise. Take two consecutive points $\mathbf{c}_k$ and $\mathbf{c}_{k+1}$ and form the curve $\gamma_k$.
(2) Find the sets $\mathscr{F}^p_{\gamma_k}$ and $\mathscr{F}^q_{\gamma_k}$, and construct $\mathscr{I} = [a^p_{\gamma_k}, b^p_{\gamma_k}] \times [a^q_{\gamma_k}, b^q_{\gamma_k}]$.
(3) Check if $(0, 0) \in \mathscr{I}$.
  3.1. If true, update $\mathbf{c}_{k+1} \leftarrow \gamma_k(\omega)$ where $0 < \omega < 1$ and form a new curve $\gamma_k(t)$ with $0 \leqslant t \leqslant 1$. Then go to 2.
  3.2. Otherwise, go to 4.
(4) Compute sum = sum + $\Delta_{\mathbf{z} \in [\mathbf{c}_k, \mathbf{c}_{k+1}]} \arg(\mathbf{F}(\mathbf{z}))$.
(5) $k = k + 1$.
(6) If $k = K + 1$, compute $d = \frac{\text{sum}}{2\pi}$ and terminate. If not, go to 1.

**Theorem 2.** *The above algorithm computes the topological degree of a map correctly.*

**Proof.** For every pair of consecutive points $\mathbf{c}_k$ and $\mathbf{c}_{k+1}$ with $k = 1, \ldots, K$, if $(0, 0) \notin \mathscr{I}$, then by Theorem 1 and the argument principle, the algorithm computes the correct degree. If $(0, 0) \in \mathscr{I}$, then since $\mathbf{c}_k$ is not a zero, we can always find a new point in the neighborhood of $\mathbf{c}_k$, which makes $(0, 0) \notin \mathscr{I}$ at line 3.1. Therefore, the algorithm computes the correct degree. □

### 3.2. Implementation and analysis

For implementation of the algorithm, range arithmetic can be used instead of computing the exact bounds directly since the exact enclosure computation during arithmetic operations is inefficient and may not be possible. However, range arithmetic in general produces more conservative enclosure than the theoretical one due to its inclusion property mentioned in Section 2. Therefore, there may exist more cases where line 3 of the algorithm in Section 3.1.2 becomes true, resulting in denser sampling of the boundary.

As it is pointed out in Section 2.2, the size of an interval can increase rapidly if interval arithmetic is used to evaluate a high degree polynomial. Such overestimation enlarges the size of the rectangle for the test of condition (8), leading to rapid increase of the number of samples. Affine arithmetic, however, can control the size of the range during evaluation and maintain a low number of samples for degree computation.

The number of sampling points also depends on the characteristic of the map **F** and the reduction ratio $\omega$. In general, if a zero of a map is close to the boundary of a rectangle $A$, then the number of discretization points for $\partial A$ increases. This increase is also coupled with the reduction ratio. If the reduction ratio is not small enough, then, the number of evaluations of the map may increase.

## 4. Examples

In this section, the proposed degree computation algorithm is tested with various examples. The algorithm is implemented in interval and affine arithmetic and both versions are compared for each example. They are implemented using the C++ programming language. For interval and affine arithmetic *PROFIL/BIAS* [20,21] and *Libaffa* libraries [22] are used, respectively. The tests are performed on a linux system with a 3.2 GHz CPU and 2 GB RAM. All codes are compiled with the g++ v4.0.3 compiler. In the tests, a value of $\omega = 0.5$ is used for the reduction ratio.

### 4.1. Univariate polynomials

Given a univariate polynomial, we may derive the map **F** as follows [1]:

**Proposition 3.** *Let $f(z)$ be a complex polynomial in the variable $z = x + \mathrm{i}y$. Write $f(z) = p(x,y) + \mathrm{i}q(x,y)$, where $p, q$ are the real and imaginary parts of $f(z)$, and consider $\mathbf{F} = (p,q)$. Then, if $A$ is compatible with* **F**, *the topological degree $d$ of* **F** *is equal to the number of roots $z$ (along with their multiplicities) of $f(z)$ that lie inside $A$.*

Then, we compute the topological degree of the map **F** with respect to the initial domain given as input.

**Test 1:** The test function is a polynomial of degree 22 as follows:

$$f(z) = (z^2 + z + 1)^2 (z-1)^4 (z^3 + z^2 + z + 1)^3 (z-2)(z-4)^4. \tag{18}$$

The input domain of $[-5,5] \times [-5,5]$ is provided to the algorithm. The number of discretization points is 947 for interval arithmetic and 120 for affine arithmetic with computation time of 0.09 and 0.22 s, respectively. The computed degree is 22.

**Test 2:** In this test we compare the affine and interval arithmetic versions of the degree computation algorithm with Wilkinson's polynomial of increasing degrees from 10 to 40:

$$f(t) = \prod_{i=0}^{n} \left( t - \frac{i}{n} \right), \tag{19}$$

where $n = 10, 15, \ldots, 40$. The domain $[-2,2] \times [-2,2]$ is provided as input to the program. The roots are real and equally distributed on the interval $[0,1]$ and the number of the roots is equal to the total degree of the polynomial. The elapsed time and numbers of discretizations for each degree are summarized in Table 1. Affine arithmetic requires more steps than interval arithmetic in bound computation. However, the estimation of tight error bounds in affine arithmetic offsets its performance inferiority in our case because of the less number of evaluations resulted from the tight enclosure, which is obviously observed in Figs. 2 and 3. Fig. 2 shows how the execution time changes as the degree of the polynomial increases for interval and affine arithmetic and Fig. 3 shows the relations between the number of discretization points and the input degrees. Here in both figures, the vertical axes are given in log scale and the dotted lines are the results of affine arithmetic.

**Test 3:** An example from [9] is taken for test as follows:

$$f(z) = 1.0 - (z^{16} - 1)^2. \tag{20}$$

The input domain of $[-2,2] \times [-2,2]$ is provided as input and the computed degree is 32, the computation time 0.11 and 0.21 s for interval and affine versions and the numbers of discretized points are 2048 and 148, respectively.

Table 1
Elapsed times and number of discretization points

| Degree | Interval | | Affine | |
|---|---|---|---|---|
| | Time | Number | Time | Number |
| 10 | 0 | 159 | 0.04 | 50 |
| 15 | 0.05 | 716 | 0.1 | 74 |
| 20 | 0.5 | 3670 | 0.22 | 102 |
| 25 | 3.54 | 17,052 | 0.4 | 130 |
| 30 | 25.84 | 93,599 | 0.64 | 156 |
| 35 | 176.68 | 447,320 | 0.93 | 178 |
| 40 | 1388.6 | 2,505,688 | 1.38 | 212 |

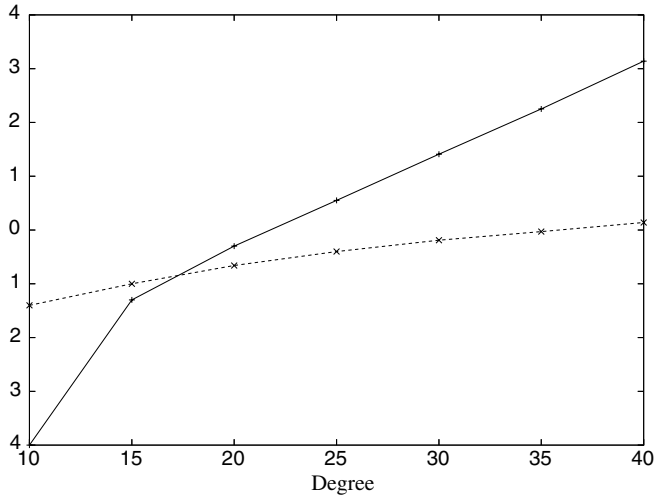


Fig. 2. Plot of degree vs. time in log scale.

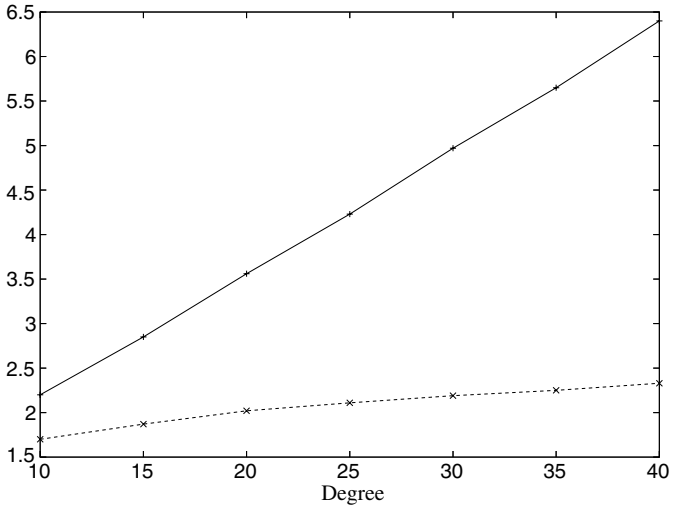


Fig. 3. Plot of degree vs. number of discretization points in log scale.

### 4.2. Bivariate polynomials

**Test 1:** Two examples are taken from [6] as follows:

$$p(x,y) = x^2 + y^2 - 0.5, \quad q(x,y) = 2xy - 0.5, \tag{21}$$

$$p(x,y) = x^3 - 3xy^2, q(x,y) = y^3 - 3x^2y. \tag{22}$$

The results of the tests are summarized in Table 2. The execution times are negligible and are not included in Table 2.

From these tests, we verify that affine arithmetic is far superior in relation to interval arithmetic in controlling the size of intervals, which results in a smaller number of discretization points in the degree computation. This reduced number of discretizations compensates the inherent performance penalty of affine arithmetic, leading to better overall performance in the degree computation. This effect becomes more obvious if a polynomial of high degree is considered.

## 5. Root computation of a univariate polynomial equation

In this section, we discuss how the proposed topological degree computation algorithm can be used to compute the roots of a univariate polynomial equation.

The main concept of our root computation algorithm is taken from the authors' previous work [23]. However, the crucial difference lies in the degree computation method used in the algorithm proposed in Section 3.

### 5.1. Modification of degree computation algorithm

We add a few lines to the proposed degree computation algorithm to handle the case that the subdivided input domain may not be compatible with a map of our interest. The modified version is given as follows:

(1) Let $\mathbf{c}_1, \ldots, \mathbf{c}_{K+1} = \mathbf{c}_1$ be a sequence of points on $\partial A$, arranged counterclockwise. Take two consecutive points $\mathbf{c}_k$ and $\mathbf{c}_{k+1}$ and form the curve $\gamma_k$.
(2) Find the sets $\mathscr{F}^p_{\gamma_k}$ and $\mathscr{F}^q_{\gamma_k}$, and construct $\mathscr{I} = [a^p_{\gamma_k}, b^p_{\gamma_k}] \times [a^q_{\gamma_k}, b^q_{\gamma_k}]$.
(3) Check if $(0,0) \in \mathscr{I}$.
  3.1. If true, check $|\mathbf{c}_{k+1} - \mathbf{c}_k| = 0$.
    3.1.1. If true, $\mathbf{c}_k$ (and $\mathbf{c}_{k+1}$) is a zero. Report and return.
    3.1.2. If not, update $\mathbf{c}_{k+1} \leftarrow \gamma_k(\omega)$ where $0 < \omega < 1$ and form a new curve $\gamma_k(t)$ with $0 \leqslant t \leqslant 1$. Then go to 2.
  3.2. Otherwise, go to 4.
(4) Compute sum = sum + $\Delta_{\mathbf{z} \in [\mathbf{c}_k, \mathbf{c}_{k+1}]} \arg(\mathbf{F}(\mathbf{z}))$.
(5) $k = k + 1$.
(6) If $k = K + 1$, compute $d = \frac{\text{sum}}{2\pi}$ and return. If not, go to 1.

In this algorithm we only have to discuss lines 3.1 through 3.2. The rest remains the same as the one in Section 3.1.2.

**Theorem 4.** *From the above algorithm, if* $(0,0) \in \mathscr{I}$ *and* $|\mathbf{c}_{k+1} - \mathbf{c}_k| = 0$, *then* $\mathbf{c}_k$ *is a zero.*

Table 2
Results of degree computation

| Equation | Domain | Degree | Evaluations (IA) | Evaluations (AA) |
|---|---|---|---|---|
| (21) | $[0,1]^2$ | 0 | 8 | 8 |
| (22) | $[-1,1]^2$ | −3 | 28 | 12 |

**Proof.** Suppose that $\mathbf{c}_{k+1}^{(n)}$ is an updated discretized point, $\gamma_k^{(n)}(t)$ the curve formed with $\mathbf{c}_k$, and $\mathbf{c}_{k+1}^{(n)}$ and $\mathscr{I}^{(n)}$ the $\mathscr{I}$ obtained from line 3 after line 3.1.2 is performed $n$ times. If $(0,0) \in \mathscr{I}^{(n)}$, then a zero may be included in $[\mathbf{c}_k, \mathbf{c}_{k+1}^{(n)}]$. If $\mathbf{c}_k$ is not a zero, then we can always find a point $\mathbf{c}_{k+1}^{(m)}$ $(m > n)$ in the neighborhood of $\mathbf{c}_k$ such that $(0,0) \notin \mathscr{I}^{(m)}$. However, if we have $(0,0) \in \mathscr{I}^{(n)}$ for $n \to \infty$, we get $\mathbf{c}_{k+1}^{(n)} = \gamma_k^{(n-1)}(\omega)$, which is equivalent to $\gamma_k^{(0)}(\omega^{n-1})$. Since $0 < \omega < 1$, $\gamma_k^{(0)}(\omega^{n-1}) \to \gamma_k^{(0)}(0)$, which is equal to $\mathbf{c}_k$. Therefore, $\mathbf{c}_{k+1}^{(n)}$ converges to $\mathbf{c}_k$ and $\mathscr{I}^{(n)}$ to the image of $\mathbf{F}(\mathbf{c}_k)$. This implies that $\mathbf{c}_k$ should be a zero.

In practice, it is required to use a tolerance such that $|\mathbf{c}_{k+1} - \mathbf{c}_k| < \lambda$, $\lambda > 0$ in line 3.1 instead. The tolerance may be chosen to be a machine tolerance for implementation in floating point arithmetic. □

### 5.2. Root computation algorithm

Let us consider a rectangle $A = [a_1, b_1] \times [a_2, b_2]$ in $\mathbf{R}^2$, where $[a_i, b_i]$ is a closed interval in $\mathbf{R}$. Then the root finding algorithm is described as follows, see also Table 3:

In line 1, the total number of roots in $A$ is computed by the function, degree($A$) which computes the topological degree of a given map $\mathbf{F}$ for a region $A$. If there is no root, then the routine terminates in line 2. Lines 3 and 4 check if the size of $A$ is less than the user defined tolerance, TOL, which needs to be provided by the user. If so, then $A$ is added to the output list marked as containing roots. If not, the rectangle $A$ is subdivided into four rectangles $A_1, A_2, A_3$ and $A_4$ at the mid points of each side of $A$. The numbers of roots in each $A_i$ $(i = 1, 2, 3, 4)$ are computed in line 7.

Theoretically, the sum of the numbers of roots in $A_i$ $(i = 1, 2, 3, 4)$ should be equal to the number of roots in $A$. However, it frequently happens that a root lies on the boundary of $A_i$, which makes $\mathbf{A}_i$ not compatible with a map. In such a case, the degree computation function, degree() reports that a root lies on the boundary. If this happens, then lines 9 through 22 are performed. In line 10, each rectangle $A_i$ is adjusted such that the size (width and height) of $A_i$ is increased by a certain amount. In this algorithm we use an amount of TOL $\times$ 0.123. After this adjustment, each new rectangle $A_i$ may overlap with its adjacent rectangle. When such overlap

Table 3
Modified TDB algorithm

```
Iteration(A)
 1: deg = degree(A);
 2: if (deg == 0)then return;
 3: size_x = |b1 − a1|; size_y = |b2 − a2|
 4: if (size_x < TOL AND size_y < TOL)then Add A to the output list; return;
 5: end
 6: subdivide(A, A1, A2, A3, A4);
 7: deg1 = degree(A1);deg2 = degree(A2);deg3 = degree(A3);deg4 = degree(A4);
 8: total_degree = deg1 + deg2 + deg3 + deg4;
 9: while(a root lies on the boundary)
10:    adjust(A1, A2, A3, A4, TOL);
11:    deg1 = degree(A1);deg2 = degree(A2); deg3 = degree(A3);deg4 = degree(A4);
12:    total_degree=deg1+deg2+deg3+deg4;
13:    if(numberIteration>NITER)
14:       if(a root lies on the boundary) then
15:          Add S to the output list; return;
16:       end
17:       adjust(A);
18:       subdivide(A, A1, A2, A3, A4);
19:       numberIteration = 0;
20:    end
21:    numberIteration = numberIteration + 1;
22: end
23:  Iteration(A1); Iteration(A2);
    Iteration(A3); Iteration(A4);
end
```

occurs, there may be a case where a certain root is counted multiple times if it is located in the overlapping region. But this does not invalidate the algorithm's logic since the purpose of lines 9 through 22 is to obtain regions $A_1$–$A_4$, each of which is compatible with the input map. The total number of roots over all $A_i$ is computed as in lines 11 and 12. This process is repeated until no zero lies on the boundary of $A_i$.

We introduce another termination condition in line 13 which checks the number of failures of degree computation for an adjusted region. This condition is considered from the practical point of view. Since floating point arithmetic has limited precision, it cannot properly represent values near a root of high multiplicity. So in order to handle such a case, if the degree computation routine fails more than NITER times, we stop the loop and add the current region to the output list without subdivision.

When the iteration stops, each rectangle $A_i$ is provided as an argument to the routine itself recursively. After all the recursive calls are terminated, the output boxes are post-processed. Due to the adjustment process, the algorithm often generates different rectangles which contain the same root. So, all rectangles need to be checked such that any rectangles which overlap are merged into a single one. Then the algorithm computes the number of roots inside each merged box and reports the final number of roots in each domain.

### 5.3. Examples

In this section, we test the proposed root computation algorithm with some of the examples used in Section 4. Tests are performed on a Linux machine with a 3.2 GHz CPU and 2 GB RAM.

**Test 1:**

$$f(z) = (z^2 + z + 1)^2(z - 1)^4(z^3 + z^2 + z + 1)^3(z - 2)(z - 4)^4 = 0. \tag{23}$$

The input domain of $[-5, 5] \times [-5, 5]$ is provided to the algorithm. The time taken to compute the roots of the polynomial $f(z)$ is 27.71 s for a tolerance for termination of $10^{-7}$ using the reduction ratio of 0.5. The computed roots along with their multiplicities are given in Table 4:

Table 4
Roots of Eq. (23)

| Roots | Multiplicity |
|---|---|
| [−7.216432323e−08, 7.215728886e−08] + i[0.999999939, 1.000000014] | 3 |
| [0.999999939, 1.000000014] + i[−7.21596332e−08, 7.216197888e−08] | 4 |
| [1.999999955, 2.000000029] + i[−7.21596332e−08, 7.216197888e−08] | 1 |
| [3.999999987, 4.000000061] + i[−7.21596332e−08, 7.216197888e−08] | 4 |
| [−0.5000000429, −0.4999999684] + i[0.8660253868, 0.8660254613] | 2 |
| [−1.000000014, −0.999999939] + i[−7.21596332e−08, 7.216197888e−08] | 3 |
| [−0.5000000429, −0.4999999684] + i[−0.8660254613, −0.8660253868] | 2 |
| [−7.216432323e−08, 7.215728886e−08] + i[−1.000000014, −0.999999939] | 3 |

Table 5
Roots of Eq. (24)

| Roots | Multiplicity |
|---|---|
| [0.03333330972, 0.03333337694] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.06666660814, 0.06666667536] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.09999997377, 0.100000041] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.1333332722, 0.1333333394] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.1666666378, 0.166666705] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| ⋮ | |
| [0.8999999424, 0.9000000096] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.933333308, 0.9333333752] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.9666666064, 0.9666666736] + i[−4.038214675e−08, 2.700090417e−08] | 1 |
| [0.999999972, 1.000000039] + i[−4.038214675e−08, 2.700090417e−08] | 1 |

**Test 2:** Wilkinson's polynomial of degree 30,

$$f(t) = \prod_{i=0}^{30} \left( t - \frac{i}{30} \right) = 0, \tag{24}$$

is tested with an input domain $[-2.0, 2.0] \times [-2.0, 2.0]$. Its roots are real and equally distributed on the interval $[0, 1]$. The tolerance for termination is $10^{-7}$ with the reduction ratio of 0.5. The time taken to compute the roots of the equation is 66.43 s. The computed roots along with their multiplicities are given in Table 5.

## 6. Conclusions

A reliable topological degree computation algorithm is proposed in this paper. Using the inclusion property of range arithmetic, the proposed method adaptively chooses discretization points along the boundary of a domain satisfying condition (8) to compute the topological degree of a mapping in 2D space using the argument principle. Our method does not require any pre-process or estimation of explicit global information such as derivative computation or root isolation which may not be easily obtained practically, and always generates a correct topological degree value.

The algorithm is implemented in affine arithmetic. This is, in general, more expensive than interval arithmetic but offers a better range control which, in turn, leads to a tighter enclosure of the exact value during evaluation. Due to the tighter bound of affine arithmetic, the degree computation algorithm in affine arithmetic outperforms the one implemented in interval arithmetic when a map of high degree is considered in the calculation.

Two issues involved in our algorithm merit further consideration. First, the choice of the reduction ratio in the algorithm is a critical factor in determining the number of discretization points for the degree computation. As long as the reduction ratio is less than one, then the degree computation algorithm always yields a correct degree value. However, the algorithm discretizes differently depending on the ratio, leading to the different number of discretization points. Therefore, the adaptive choice of the reduction value considering the input mapping can decrease the number of discretizations and we can improve the overall performance of the algorithm. The second issue relates to the rejection strategy. Our algorithm rejects the case that the origin is contained even though the real argument difference is less than $\pi$ for simplicity. However, it is obvious that if such a case is included in the computation, the performance can be improved. These two topics are recommended for future investigation.

## References

[1] T. Sakkalis, The Euclidean algorithm and the degree of the Gauss map, SIAM J. Comput. 19 (3) (1990) 538–543.
[2] F. Stenger, Computing the topological degree of a mapping in $R^n$, Numer. Math. 25 (1975) 23–38.
[3] R.B. Kearfott, An efficient degree-computation method for a generalized method of bisection, Numer. Math. 32 (1979) 109–127.
[4] M. Stynes, A simplification of Stenger's topological degree formula, Numer. Math. 33 (1979) 147–156.
[5] M. Stynes, On the construction of sufficient refinements for computation of topological degree, Numer. Math. 37 (1981) 453–462.
[6] T. Boult, K. Sikorski, An optimal complexity algorithm for computing the topological degree in two dimensions, SIAM J. Sci. Stat. Comput. 10 (4) (1989) 686–698.
[7] B. Mourrain, M.N. Vrahatis, J.C. Yakoubsohn, On the complexity of isolating real roots and computing with certainty the topological degree, J. Complex. 18 (2002) 612–640.
[8] P. Henrici, Applied and Computational Complex Analysis, John Wiley, New York, 1974.
[9] X. Ying, I.N. Katz, A reliable argument principle algorithm to find the number of zeros of an analytic function in a bounded domain, Numer. Math. 53 (1988) 143–163.
[10] M.C. Carpentier, A.F. Dos Santos, Solution of equations involving analytic functions, J. Comput. Phys. 45 (1982) 210–220.
[11] B. Davies, Locating the zeros of an analytic function, J. Comput. Phys. 66 (1986) 36–49.
[12] P. Kravanja, M.V. Barel, A derivative-free algorithm for computing zeros of analytic functions, Computing 63 (1999) 69–91.
[13] R.E. Moore, Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1966.
[14] J. Stolfi, L.H. Figueiredo, An introduction to affine arithmetic, TEMA Tend. Mat. Appl. Comput. 4 (3) (2003) 297–312.
[15] N.M. Patrikalakis, T. Maekawa, Shape Interrogation for Computer Aided Design and Manufacturing, Springer-Verlag, Heidelberg, 2002.

[16] ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating–Point Arithmetic, IEEE, New York, reprinted in ACM SIGPLAN Notices, 22(2) (1985) 9–25, February 1987.
[17] S.L. Abrams, W. Cho, C.-Y. Hu, T. Maekawa, N.M. Patrikalakis, E.C. Sherbrooke, X. Ye, Efficient and reliable methods for rounded-interval arithmetic, Comput. Aided Des. 30 (8) (1998) 657–665.
[18] C.M. Hoffmann, Robustness in geometric computations, J. Comput. Inform. Sci. Eng. 1 (2) (2001) 105–204.
[19] M.S. Petković, L.D. Petković, Complex Interval Arithmetic and Its Applications, Wiley-VCH, Berlin, 1998.
[20] O. Knuppel, Bias-basic interval arithmetic subroutines, Technical Report 93.3, Technical University of Hamburg – Harburg, Harburg, Germany, 1993.
[21] O. Knuppel, Profil-programmers runtime optimized fast interval library, Technical Report 93.4, Technical University of Hamburg-Harburg, Harburg, Germany, 1993.
[22] Libaffa – c++ affine arithmetic library for gnu/linux, <http://www.nongnu.org/libaffa/>, 2005.
[23] K.H. Ko, T. Sakkalis, N.M. Patrikalakis, Resolution of multiple roots of nonlinear polynomial systems, International Journal on Shape Modeling 11 (1) (2005) 121–147.