Nonlinear Polynomial Systems: Multiple Roots and their Multiplicities

K. H. Ko
khko@mit.edu

T. Sakkalis
takis@deslab.mit.edu

N. M. Patrikalakis
nmp@mit.edu

Massachusetts Institute of Technology
Cambridge, MA 02139-4307, USA

Abstract

In this paper we present methods for the computation of roots of univariate and bivariate nonlinear polynomial systems as well as the identification of their multiplicity. We first present an algorithm, called the TDB algorithm, which computes the values and the multiplicities of roots of a univariate polynomial. The procedure is based on the concept of the degree of a certain Gauss map, which is deduced from the polynomial itself. In the bivariate case, we use a combination of resultants and our procedure for the univariate case, as the basis for developing an algorithm for locating the roots and computing their multiplicities. Our methods are robust and global in nature. Complexity analysis of the proposed methods is included together with comparison with standard subdivision methods. Examples illustrate our techniques.

Keywords: Cauchy index, Gauss map, univariate and bivariate polynomials, topological degree

1. Introduction

Polynomials are popular in curve and surface representations and many critical problems arising in Computer Aided Geometric Design such as surface interrogation, are reduced to finding the zero set of a system of nonlinear polynomial equations

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \tag{1}$$

where $\mathbf{f} = (f_1, f_2, \cdots, f_n)$ and each f_i is a polynomial of l independent variables $\mathbf{x} = (x_1, x_2, \cdots, x_l)$.

Several root-finding algorithms for multivariate polynomial systems (1) have been used in practice. Newton type methods, which are classified as *local* solution techniques, have been applied to many problems since they are quadratically convergent and produce accurate results. They, however, require good initial approximations of the roots of the systems, and fail to provide full assurance that all roots have been found. These limitations can be overcome by *global* solution techniques, which can be categorized into three different types [10]: (1) algebraic and hybrid methods, (2) homotopy methods and (3) subdivision methods. Among those types, the subdivision methods have been widely used in practice because of their performance and efficiency. The Interval Projected Polyhedral (IPP) algorithm [14, 10] is one example, and it has been successfully applied to various problems. Of particular interest is locating zeros of a univariate polynomial. It is a critical problem in diverse fields such as control theory and much literature has been devoted to it [6].

Most of the root finding algorithms, however, experience difficulties in dealing with roots with high multiplicity such as performance deterioration and lack of robustness in numerical computation. For example, the IPP algorithm, which belongs to the subdivision class of methods, slows down drastically and suffers from proliferation of boxes that are assumed to enclose roots. Moreover, since a root with high multiplicity is unstable with respect to small perturbation, round-off errors during floating point arithmetic may change the topological aspect in such a way that a cluster of roots could be formed around the root.

Solving univariate polynomials with multiple roots is an important but difficult task. Rump [11] collected nine methods to bound multiple roots of polynomials and compared them rigorously. He also proposed a new hybrid algorithm which gives numerically nearly optimal bounds for multiple roots of univariate polynomials. Even though those methods work well in most cases, it is not easy for a user to control the size of the bound of a root in general. Wilf [16] used Sturm sequences to compute all roots of a univariate polynomial, but his approach relies on the division of polynomials to compute Sturm sequences. So, it is not numerically robust unless exact arithmetic or symbolic computation is used.

Literature on root multiplicity of a system of equations

is quite limited. Möller and Stetter [8] used an eigenproblem method to calculate the solutions of systems of polynomial equations and studied the case where the system contains multiple roots in common. Eigenvalues of a matrix obtained from an input algebraic equation system correspond to roots of the system and the algebraic multiplicity of each eigenvalue is equal to the multiplicity of the corresponding root. Also Moritsugu and Kuriyama [9] proposed a practical algorithm to compute roots and their multiplicity of a system of algebraic equations based on the work by Möller and Stetter [8]. However, their approaches require a step to find a *nonderogatory matrix*, a matrix whose *Frobenius normal form* consists of one companion block [9], using random linear combinations of multiplication tables, which has to be performed with trial and error. The same problem was studied by Manocha and Demmel [5] in the form of finding multiple intersections of parametric and algebraic curves. Their approach uses elimination theory and resultants represented in matrix form, reducing the intersection problem to computing the eigenvalues of a matrix. To handle high multiplicity eigenvalues, they proposed a method which can cluster the eigenvalues. This heuristic is based on a fact that rounding errors involved in floating arithmetic will change the topological structure of multiple eigenvalues such that the eigenproblem will end up locating a cluster of eigenvalues and the average over the cluster is a good approximation of a multiple root.

In this paper, we restrict our scope to finding real and complex roots for univariate polynomials, and real roots for bivariate polynomial systems. We provide definitions of root multiplicity for both cases and algorithms to compute roots and their multiplicity. We analyze the behavior of the IPP algorithm around a multiple root and propose a post-processing procedure which sorts out the intervals that contain at least one root under a user specified tolerance, and provides explicit information about root multiplicity robustly.

The paper is structured as follows: in Section 2 definitions for the multiplicity of a root are provided. In Section 3, a root computation procedure based on the notion of local topological degree is proposed with comparison to the IPP algorithm. An algorithm for solving a univariate polynomial equation, called the TDB algorithm is proposed in Section 4 and Section 5 is devoted to the multiple roots of bivariate polynomial systems. Concluding remarks are made in Section 7.

2. Multiplicity of Roots

Let $f(x)$ be a polynomial of a single variable with coefficients in the set of real numbers, $\mathbf{R}$. We denote by $f'(x), f'', \cdots, f^{(m)}(x)$ the first, second, $\cdots$, m-th derivative of $f(x)$, respectively. A number $a \in \mathbf{C}$, where $\mathbf{C}$ is the set of the complex numbers, shall be called a root of $f(x)$ if $f(a) = 0$. We say that a root a of $f(x)$ has multiplicity k, if

$$f(a) = f'(a) = \cdots = f^{(k-1)}(a) = 0, \text{ and } f^{(k)}(a) \neq 0. \tag{2}$$

Now suppose that we are given two polynomials $f(x, y), g(x, y)$ with real coefficients in the two variables x and y. Consider the system

$$\begin{aligned} f(x, y) &= 0, \\ g(x, y) &= 0. \end{aligned} \tag{3}$$

Define

$$\begin{aligned} V_f &= \{(x, y) \in \mathbf{C} \mid f(x, y) = 0\} \quad \text{and} \\ V_g &= \{(x, y) \in \mathbf{C} \mid g(x, y) = 0\}. \end{aligned}$$

Definition 2.1 *A root of the system (3) shall be a pair of numbers $z_0 = (x_0, y_0)$ such that $f(x_0, y_0) = g(x_0, y_0) = 0$. Moreover, z_0 shall be called isolated if there exists an open ball $B(z_0, r)$ centered at z_0 of positive radius r, so that z_0 is the only root of (3) in $B(z_0, r)$.*

We may define the *multiplicity* k of z_0, as a root of the system (3) as follows (*Lemma 2, p. 407 of [12]*): Without loss of generality, we may assume, after a linear coordinate change, that f, g have the form

$$\begin{aligned} f(x, y) &= a_0 y^n + a_1(x) y^{n-1} + \cdots + a_n(x), \\ g(x, y) &= b_0 y^m + b_1(x) y^{m-1} + \cdots + b_m(x), \end{aligned} \tag{4}$$

where a_0, b_0 are non zero constants. We shall call such f and g *regular* in y. Suppose now that z_0 is the *only common point* of V_f and V_g lying above x_0. Consider $h(x) = \mathrm{Res}_y(f, g)$, the resultant of f, g with respect to y. Then,

Definition 2.2 *The multiplicity of $z_0 = (x_0, y_0)$ as a root of (3) is the multiplicity of x_0 as a zero of $h(x)$.*

It is convenient to associate to f and g the vector field

$$F : \mathbf{R}^2 \to \mathbf{R}^2, \qquad F(x, y) = (f(x, y), g(x, y)).$$

In that respect, a common zero $z_0 = (x_0, y_0)$ of f and g is nothing but a zero of F. In addition, we shall call z_0 a real zero if $z_0 \in \mathbf{R}^2$.

3. Univariate Case

In this section we will give a procedure, that is based on the computation of the local degree of a vector field arising from $f(x)$, for the computation of the multiplicity of a root of $f(x)$. We start this section by explaining *the Gauss map*, which is used as a basis for the development of root multiplicity computation.

3.1. The Gauss Map

Let $p(x,y), q(x,y)$ be real polynomials without common factors, and let us consider the vector field

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

Let A be a rectangle in the plane defined by $a_1 \le x \le a_2,\ a_3 \le y \le a_4$ so that no zero of F lies on the boundary ∂A, and $p \cdot q$ does not vanish at its vertices; we call such an A compatible with F. Then, we can define the Gauss map

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

where S^1 is the unit circle. Since $\|F\| \neq 0$ on ∂A, G is continuous. Assume that both ∂A and S^1 carry the counterclockwise orientation. Then the degree d of G is an integer that, roughly speaking, tells how many times ∂A is wrapped around S^1 by G, see Figure 1.

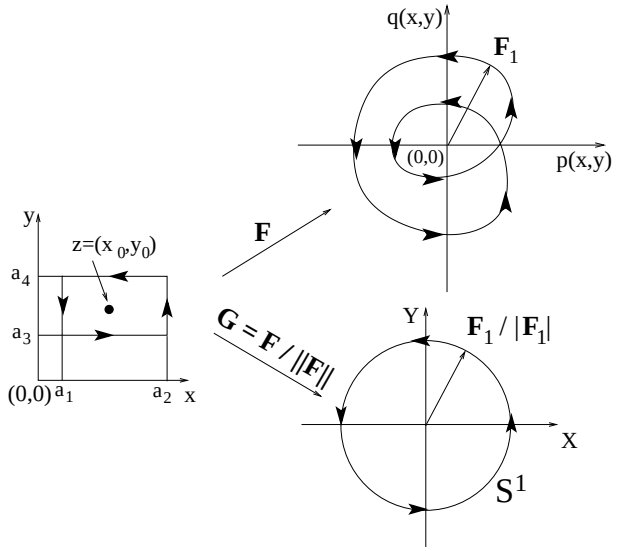

Figure 1. A schematic diagram of the degree of Gauss map

A univariate polynomial $f(x)$ can be converted into a complex polynomial $f(z)$ by replacing x by $z = x + iy$. If we solve the complex polynomial equation $f(z) = 0$ in the complex domain, we can find real and complex roots of $f(x) = 0$ simultaneously. Our procedure for the univariate case is based on the following Proposition, whose proof can be found in [12].

Proposition 3.1 *Let $h(w)$ be a complex polynomial in the variable w. Write $h(w) = R(w) + iI(w)$, and consider $F = (R, I)$, where R, I are the real and imaginary parts of $h(w)$, respectively. Then, if A is compatible with F, d is equal to the number of roots z (along with their multiplicities) of $h(w)$ that lie inside A.*

3.2. Computation of d

In this paragraph we will give two methods of the computation of d. The first is based on the notion of the Cauchy index of a rational function, and the second is a more direct one.

3.2.1. Cauchy Index Method

Definition 3.1 *Let $R(x)$ be a rational function and $[a,b]$ a closed interval, $a < b$, with $R(a) \neq \infty, R(b) \neq \infty$. Denote by N_-^+ (N_+^-) the number of real poles x of R inside (a,b) so that*

$$\lim_{t \to x^-} R(t) = -\infty, \quad \textit{and} \quad \lim_{t \to x^+} R(t) = \infty,$$

$$\left(\lim_{t \to x^-} R(t) = \infty, \quad \textit{and} \quad \lim_{t \to x^+} R(t) = -\infty \right)$$

respectively. Then the **Cauchy index** *$I_a^b R$ of R on $[a,b]$ is defined as*

$$I_a^b R = N_-^+ - N_+^-.$$

By convention, $I_a^b R = -I_b^a R$.

Example 1 *Let $p(x)$ be a real polynomial. Then, $I_a^b \frac{p'}{p}$ is equal to the number of distinct real roots of p that are inside (a,b).*

If $R(x) = r(x)/s(x)$ and $[a,b]$ are as above, then we may compute the Cauchy index $I_a^b R$ by using the Euclidean algorithm [12] and Sturm's theorem [7]. Indeed, if $\deg s \ge \deg r$ we may define a sequence $f_1, f_2, \cdots, f_m$ of polynomials as follows: $f_1 = s, f_2 = r$ and $f_i = q_i f_{i+1} - f_{i+2}$ with $\deg f_{i+2} < \deg f_{i+1}$, and $f_m = GCD(r,s)$. Then,

Theorem 3.1 [Sturm] *Let $f_1, \cdots, f_m$ be as above, and let $V(x)$ be the number of sign changes in the sequence of numbers $f_1(x), f_2(x), \cdots, f_m(x)$, $x \in \mathbf{R}$. Then,*

$$I_a^b R = V(a) - V(b).$$

Finally, we define

$$\begin{aligned} I_A F = & \quad I_{a_1}^{a_2} \frac{q(x,a_3)}{p(x,a_3)} + I_{a_3}^{a_4} \frac{q(a_2,y)}{p(a_2,y)} + I_{a_2}^{a_1} \frac{q(x,a_4)}{p(x,a_4)} \\ & + \quad I_{a_4}^{a_3} \frac{q(a_1,y)}{p(a_1,y)}. \end{aligned}$$

Then, we have

Theorem 3.2 (Proposition 1, [12], p. 540)

$$d = -\frac{1}{2} I_A F.$$

Example 2 *Suppose*

$$f(z) = (z - \frac{1}{2})^5 = 0. \quad (5)$$

Obviously, the only root of $f(x)$ *is* 0.5 *with multiplicity* 5*. We are going to use the above method to compute the multiplicity of the root.*

Let $z = x + iy$. Then we can rewrite equation (5) as follows:

$$f(z) = (x + iy - \frac{1}{2})^5 = p(x, y) + iq(x, y), \quad (6)$$

where

$$\begin{aligned} p(x,y) &= -\frac{1}{32} + \frac{5x}{16} - \frac{5y^4}{2} - \frac{5x^2}{4} + 5xy^4 \\ &\quad -\frac{15xy^2}{2} + 15x^2y^2 + \frac{5y^2}{4} + \frac{5x^3}{2} - \frac{5x^4}{2} \\ &\quad +x^5 - 10x^3y^2, \\ q(x,y) &= -\frac{5y}{16} - \frac{5y^3}{2} - 10yx^3 + \frac{15yx^2}{2} + y^5 \\ &\quad +5yx^4 - \frac{5xy}{2} - 10y^3x^2 + 10y^3x. \end{aligned}$$

Since the root of equation (5) is known, we can create a rectangular domain A which encloses $(0.5, 0)$ as follows:

$$A = [0.49, 0.51] \times [-0.01, 0.01]. \quad (7)$$

The boundary of the domain A, ∂A, consists of four straight lines, from which we can obtain the values $a_1 = 0.49$, $a_2 = 0.51$, $a_3 = -0.01$ and $a_4 = 0.01$ that are used for the Cauchy index computation. Then, we have four polynomial equations $p(x, a_3) = 0$, $p(x, a_4) = 0$, $p(a_1, y) = 0$ and $p(a_2, y) = 0$. The roots of each equation can be obtained by using Maple or Mathematica. Table 1 summarizes all roots of $p(x, a_3) = 0$. We choose three roots, no. 2, 3 and 4 from Table 1 since they lie within a_1 and a_2. From the algorithm for the Cauchy index calculation, we find that the corresponding Cauchy index, $I_{a_1}^{a_2} R_3$, is -3. Similarly, we can find $I_{a_3}^{a_4} R_2 = -2$, $I_{a_2}^{a_1} R_4 = -3$ and $I_{a_4}^{a_3} R_1 = -2$. Therefore, the sum of all Cauchy index values is

No.	Roots
1	0.4692231646
2	0.4927345747
3	0.5000000000
4	0.5072654253
5	0.5307768354

Table 1. Roots of $p(x, a_3) = 0$

$$I_A F = I_{a_1}^{a_2} R_3 + I_{a_3}^{a_4} R_2 + I_{a_2}^{a_1} R_4 + I_{a_4}^{a_3} R_1 = -10, \quad (8)$$

and the degree of the Gauss map becomes

$$d = -\frac{1}{2} I_A F = 5, \quad (9)$$

which is the known multiplicity of the root $x = 0.5$.

3.2.2. Direct Computation Method The topological degree computation using the Gauss map and the Cauchy index explained in Sections 3.1 and 3.2.1 are mathematically sound. However, the algorithms may not be easily implemented for practical purposes since they require symbolic manipulation to extract real and imaginary parts $p(x, y)$ and $q(x, y)$ as in equation (6). In addition, roots of polynomials along the boundary that are required in the Cauchy index calculation have to be found robustly, which is a problem that we are trying to solve in this paper. Any symbolic computation routines such as Maple and Mathematica could be used in the implementation but the performance of those programs is not promising. Therefore, a different approach has to be considered for the implementation.

Instead of using the Cauchy index to compute the degree of the Gauss map, we opt to compute the degree of the Gauss map directly from the map F defined in Section 3.1. We can calculate the degree of the Gauss map by understanding the behavior of the map F. Obviously, the number of times that the map $F(x, y)$ surrounds the origin $(0, 0)$ as (x, y) is moving along the closed loop ∂A which encloses a zero in the xy domain is equivalent to the degree of the Gauss map. This allows us to compute the degree without the help of the Cauchy index computation. We will denote this algorithm as the *direct computation method*.

We define a map F as given in Proposition 3.1 and sample points on the closed loop ∂A in the xy domain, which are mapped onto vectors through F. How many points should be sampled is a critical issue in this process. The number of sampling points is large enough to capture all small loops of the map F surrounding the origin in the vector field. This problem is similar to computation of zeros of analytic functions using the *principle of the argument*. An estimate of the number of sampling points can be obtained based on the results of [3, 1, 17].

Suppose we have two consecutive sample points m, $m+1$ on ∂A. The corresponding vectors are $\mathbf{F}_m$ and $\mathbf{F}_{m+1}$, respectively as shown in Figure 2. The angle between the vectors is calculated by $\Delta\phi_{m+1} = arg(\mathbf{F}_{m+1}) - arg(\mathbf{F}_m)$, where $arg(\mathbf{F})$ is an angle from a fixed axis. The total sum of $\Delta\phi_{m+1}$

$$\phi_{total} = \sum_{i=0}^{n} \Delta\phi_{i+1} \quad (10)$$

will give the total angle change of a vector of the map F during one loop along ∂A. The rotation number, i.e. the degree of the Gauss map can be calculated by $d = \frac{\phi_{total}}{2\pi}$.

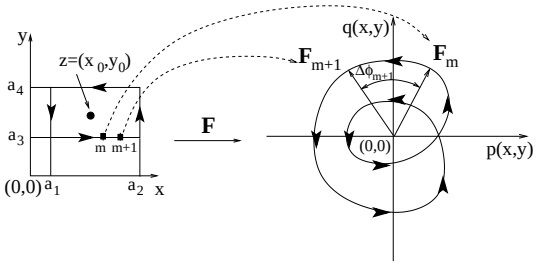

Figure 2. A diagram for the direct computation method

An example polynomial in complex numbers is $f(z)$ in equation (6). At sampled points (x, y) along the boundary ∂A, the polynomial $f(z)$ is evaluated at a complex value $z = x + iy$. Two hundred points are sampled along ∂A and their corresponding points of the map F are plotted in Figure 3. Each side of the rectangle in xy plane is mapped by F

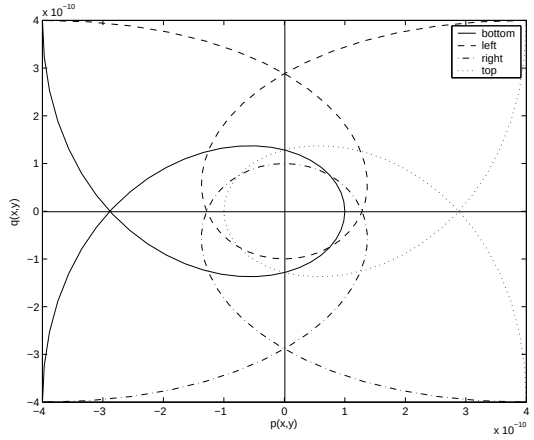

Figure 3. A plot of (p(x,y),q(x,y))

onto a curve which wraps the origin one and a quarter time, namely, the rotation angle in the Gauss map is $2\pi + \frac{\pi}{2}$ as shown in Figure 3. Therefore, the origin is wrapped around five times by the map F, which corresponds to the result of the direct calculation method.

Applications and Examples To understand the behavior of the IPP algorithm [10, 14], let us consider a univariate polynomial defined as follows:

$$y = f(x) = \left(x - \frac{1}{2}\right)^{2m}, \quad m = 1, 2, 3, 4 \ \ (0 \le x \le 1). \tag{11}$$

Two tolerances for the IPP algorithm are chosen to solve equation (11): 10^{-4} and 10^{-7}. These tolerances relate to the maximum size of the intervals not discarded by the IPP algorithm as not containing roots.

The number of roots before *consolidation* for each m is summarized in Table 2, based on IEEE double (64 bit) precision rounded interval arithmetic. The consolidation is a process of taking the union of intersecting intervals, and merging them into a single interval. Due to the high degree of contact of the graph $y = f(x)$ of (11) with the x axis at the root, the IPP algorithm yields a series of intervals reported as roots as shown in Table 2. The consolidation may generate one interval which encloses roots but its size can be much larger than the input tolerances so that the consolidated interval may not be useful in practice. Moreover, the intervals produced by the IPP algorithm cannot be guaranteed to contain a root because the algorithm proceeds by discarding the intervals that do not for sure enclose the root. This means that some intervals do not have a root even though they are not discarded by the IPP algorithm. This phenomenon deteriorates when the degree of contact increases as indicated in Table 2, which can be explained as follows: The graph $y = f(x)$ is transformed to a polynomial in Bernstein basis and provided as input to the IPP algorithm. The IPP algorithm uses the control points of the polynomial to discard the regions which do not contain roots with the convex hull property of Bernstein polynomials. Near a root with high multiplicity, the polynomial is very close to the x axis, even though it does not touch it. However, the control points of the polynomial could cross the axis due to the limited precision of a computer (i.e. double precision) and rounded interval arithmetic. When this happens, the IPP algorithm cannot distinguish between two distinct states: either the polynomial itself really crosses the axis or only the control polygon crosses the axis. Therefore, the IPP algorithm will mark as a root the interval where the control polygon crosses the axis even though the real polynomial does not. Hence, each individual interval needs to be verified to make sure it contains at least one root, and the intervals which do not enclose a root should be discarded.

m	10^{-4}	10^{-7}
1	1	1
2	1	2048
3	48	49151
4	222	196607

Table 2. Number of roots of (11) before consolidation under double precision for two input tolerances

Such problems inherent to the IPP algorithm can be handled by using the procedure in Section 3. The choice of a domain A which encloses a root can be arbitrary. Since the IPP algorithm produces intervals, we can use them to con-

m	IPP Tol. 10^{-4}		IPP Tol. 10^{-7}	
	Intervals	Red. (%)	Intervals	Red. (%)
1	1	0	1	0
2	1	0	1	99.9512
3	2	97.9167	2	99.998
4	2	99.5495	2	99.9995

Table 3. Number of remaining intervals and reduction ratios for the roots of (11)

struct a rectangular domain A. The height of the domain may be chosen equal to the size of each interval for simplicity. Then, the direct calculation method is applied to each rectangular domain. If the direct computation method produces zero, which means that there is no zero inside the box, then the corresponding interval is discarded. A value more than one indicates that the box encloses at least one zero.

Table 3 shows the number of intervals left after the direct computation method has been applied to each case shown in Table 2. A workstation with a 1.6GHz CPU and 512MB RAM under Linux was used in this test. Two intervals are sorted out as the root of the polynomial of high degree cases $m = 3, 4$. Both intervals enclose a root $x = 0.5$ and intersect with each other. We cannot choose one of them as a root. Instead, we can consolidate them to one interval (i.e. take their union), whose size is a little larger than the input tolerance.

Accuracy and Complexity Analysis The time complexity of the direct computation method is linearly proportional to the number of sample points along the boundary ∂A, n_s, and the number of input intervals produced by the IPP algorithm, n_I. Therefore, the time complexity of the method becomes $O(n_s n_I)$. However, the reduction of the number of intervals containing roots is significant. Hence it is expected that the processing time for subsequent procedures is reduced accordingly.

The topological degree computation algorithm guarantees that at least one root exists inside an interval. But it is not obvious how many isolated roots are in that interval. There could be either one isolated root with a multiplicity d or there are d isolated roots in the interval which are very closed to each other, or a combination of roots of both types. The distinction of the three cases cannot be made unless additional precision is used for evaluation.

3.3. Summary

In this section, we introduced a procedure to compute the multiplicity of a root of a univariate polynomial using the Gauss map and the Cauchy index, and developed a practical method called, the direct computation method, for multiplicity evaluation. In the next section we will propose an algorithm for solving a univariate polynomial equation using the direct computation method.

4. An Algorithm for Solving a Univariate Polynomial Equation : Topological Degree Bisection (TDB) Algorithm

In this section, a modified algorithm is proposed, which is more simple and efficient for finding all roots of a univariate polynomial than the IPP algorithm, and extracts all real and complex roots of the polynomial as well as their multiplicities. This algorithm is motivated by the work of Wilf [16]. In his work, Sturm sequences [7] are computed in calculating the number of zeros of a polynomial inside a domain. But the use of Sturm sequences is prone to numerical errors in floating point arithmetic. The proposed algorithm avoids Sturm sequences in the degree computation and is based on a quadtree decomposition technique with the direct computation method proposed in Section 3.2.2. Unlike other subdivision methods such as the IPP algorithm, the proposed method can generate boxes which are guaranteed to contain roots and produce multiplicity information of each root.

4.1. TDB Algorithm

The input of this algorithm is prepared as follows: We replace the variable of a univariate polynomial with a complex variable $z = x + iy$. Then we take a rectangle S in the complex domain which encloses all real and complex roots of the polynomial. This rectangle can be determined by using the coefficients of the polynomial and we verify that the number of roots in the rectangle (including multiplicities) is equal to the degree of the polynomial.

A pseudo-code of this algorithm is presented in Table 4. Let us consider a rectangle $S = [a_1, b_1] \times [a_2, b_2]$ in $\mathbf{R}^2$, where $[a_i, b_i]$ is a closed interval in $\mathbf{R}$ and denote TOL as the user-defined tolerance.

The input of the TDB algorithm is a rectangle S and a univariate polynomial. The rectangle is the domain where a problem is defined. The output is a list of rectangles which contain roots. Here, the input polynomial is assumed to contain *isolated roots* only. In line 1, the total number of roots in S is computed by using the direct computation algorithm. This number should be equal to the degree of the input polynomial since both real and complex roots are counted. If the computed number of roots and the degree of the polynomial are different, we increase the sampling rate for the direct computation method until both numbers match. If there is no root, then the routine terminates. Lines 3 and 4 check if the length of each side of the rectangle S is less than TOL. If so, then S is reported as a rectangle containing

```
Iteration(S)
1 :   deg = degree(S);
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
          report S; return;
5 :   subdivide(S,s1,s2,s3,s4);
6 :   deg1 = degree(s1);deg2 = degree(s2);
      deg3 = degree(s3);deg4 = degree(s4);
7 :   total_degree = deg1 + deg2 + deg3 + deg4;
8 :   while(total_degree < deg)
9 :       adjust(s1,s2,s3,s4,TOL);
10 :      deg1=degree(s1);deg2=degree(s2);
          deg3=degree(s3);deg4=degree(s4);
11 :      total_degree = deg1 + deg2 + deg3 + deg4;
12 :  end
13 :  Iteration(s1); Iteration(s2);
      Iteration(s3); Iteration(s4);
end
```

Table 4. TDB algorithm

a root. If not, the rectangle S is subdivided into four rectangles s_1, s_2, s_3 and s_4 at the mid points of each side of S. The numbers of roots in s_i $(i = 1, 2, 3, 4)$ are computed in line 6. Theoretically, the sum of the numbers of roots in s_i $(i = 1, 2, 3, 4)$ should be equal to the number of roots in S. However, it frequently happens that a root lies on a side of a subdivided rectangle, which violates the assumption made in the Gauss map computation discussed in Section 3.2. Therefore, in such a case, the sum of the numbers of roots over each s_i $(i = 1, 2, 3, 4)$ is different from that over S. If this happens, then lines 7 through 11 are performed. In line 8, each rectangle s_i is adjusted such that the size (width and height) of s_i is increased by $0.01 \times TOL$. After this adjustment, each new rectangle s_i overlaps with its adjacent rectangle. Then, the total number of roots over all s_i is computed as in lines 9 and 10. This process is repeated until the total number of roots over the s_i is equal to or larger than the number of roots in S. When the iteration stops, each rectangle s_i is provided as an argument to the routine itself recursively. Due to the adjustment process, the algorithm often generates different rectangles which contain the same root. So, all rectangles are checked such that any two rectangles which overlap are merged into a single rectangle which encloses them.

4.2. Examples

In this section, the proposed algorithm is tested with two concrete examples. The workstation identified in Section 3 is used and the algorithm is implemented in C++ and compiled with GNU g++.

Roots	d
[0.0499999995,0.0500000001]+i[-5.820766091e-10,5.770766091e-10]	1
[0.0999999995,0.1000000001]+i[-5e-12,5.770766091e-10]	1
[0.1499999996,0.1500000001]+i[-5.820766091e-10,5.770766091e-10]	1
[0.1999999996,0.2000000002]+i[-5e-12,5.770766091e-10]	1
[0.2499999996,0.2500000002]+i[-5.820766091e-10,5.770766091e-10]	1
[0.2999999997,0.3000000003]+i[-5e-12,5.770766091e-10]	1
[0.3499999997,0.3500000003]+i[-5.820766091e-10,5.770766091e-10]	1
[0.3999999998,0.4000000004]+i[-5e-12,5.770766091e-10]	1
[0.4499999998,0.4500000004]+i[-5.820766091e-10,5.770766091e-10]	1
[0.4999999999,0.5000000005]+i[-5e-12,5.770766091e-10]	1
[0.5499999999,0.5500000005]+i[-5.820766091e-10,5.770766091e-10]	1
[0.6000000000,0.6000000006]+i[-5e-12,5.770766091e-10]	1
[0.6499999994,0.6500000001]+i[-5.820766091e-10,5.770766091e-10]	1
[0.6999999995,0.7000000001]+i[-5e-12,5.770766091e-10]	1
[0.7499999995,0.7500000001]+i[-5.820766091e-10,5.770766091e-10]	1
[0.7999999996,0.8000000002]+i[-5e-12,5.770766091e-10]	1
[0.8499999996,0.8500000002]+i[-5.820766091e-10,5.770766091e-10]	1
[0.8999999997,0.9000000003]+i[-5e-12,5.770766091e-10]	1
[0.9499999997,0.9500000003]+i[-5.820766091e-10,5.770766091e-10]	1
[0.9999999998,1.0000000001]+i[-5e-12,5.770766091e-10]	1

Table 5. Roots of equation (12)

A first example is Wilkinson's polynomial in which twenty real roots are equally distributed on $[0, 1]$:

$$p(t) = \prod_{i=1}^{20} \left(t - \frac{i}{20} \right) = 0. \tag{12}$$

The condition numbers of many of the roots of equation (12) are very high so that finding all roots is difficult [10]. This equation is provided to the proposed algorithm with a tolerance of 10^{-9} and 2000 sampling points for the direct computation method. It takes 103 seconds for computation. All roots are summarized in Table 5 together with their multiplicities d. All numbers are rounded at the 11th digit below the decimal point for display. The maximum width of the real part of the intervals representing the roots is 5.87×10^{-10}. The maximum width of the imaginary parts is 1.16×10^{-9} and all imaginary parts contain zero.

The algorithm is also tested with a polynomial with real and complex roots with multiplicities d as follows:

$$p(t) = (t^2+t+1)^2(t-1)^4(t^3+t^2+t+1)^3(t-2)(t-4)^4 = 0. \tag{13}$$

By inspection, the exact roots and multiplicities d are given in Table 6. A tolerance of 10^{-9} and 2800 sample points are used in the TDB algorithm. The resulting roots and multiplicities are summarized in Table 7. All real numbers are rounded at the fourth digit below the decimal point for display. The execution time of the computation is 60 seconds. Comparing with Table 6, we see that the obtained roots from the TDB algorithm enclose the exact roots and the multiplicities are the same.

4.3. Analysis and Comparison

It is impossible to know *a priori* how many rectangles will be tested by the direct computation method in gen-

Roots	d
i	3
1	4
4	4
$-\frac{1}{2}+i\frac{\sqrt{3}}{2}$	2
-1	3
$-\frac{1}{2}-i\frac{\sqrt{3}}{2}$	2
-i	3
2	1

Table 6. Exact roots and multiplicities of (13)

Roots	d
[-5.956e-10,5.956e-10]+i[1,1]	3
[1,1]+i[-5.956e-10,5.956e-10]	4
[4,4]+i[-5.939e-10,5.939e-10]	4
[-0.5,-0.5]+i[0.866,0.866]	2
[-1,-1]+i[-5.956e-10,5.956e-10]	3
[-0.5,-0.5]+i[-0.866,-0.866]	2
[-5.956e-10,5.956e-10]+i[-1,-1]	3
[2,2]+i[-5.94e-10,0]	1

Table 7. Roots of equation (13)

eral. But we can analyze the algorithm for the worst case. This algorithm is based on the quadtree decomposition technique. So it forms a tree with four children at each node during execution. Let us assume that the algorithm has stopped at a depth d_e and subdivision has happened at every node up to depth $d_e - 1$. Then, the total number of nodes will be $\sum_{j=0}^{d_e-1} 4^j$. For each node, the algorithm has complexity $O(mn_p)$, where m is the degree of an input polynomial and n_p is the number of sample points used by the direct computation method. Hence, the time complexity of the worst case becomes $O(4^{d_e} mn_p)$.

The complexity of the IPP algorithm for a univariate polynomial of degree m is $O(m^2)$ per step, whereas the TDB algorithm has complexity $O(mn_p)$ per step. So we cannot compare both methods directly since depending on n_p, the complexity of the TDB algorithm could be significantly different. When an input univariate polynomial contains only simple roots, then the IPP algorithm outperforms the proposed TDB algorithm. But when the input polynomial has a root with high multiplicity, the performance of the IPP algorithm progressively deteriorates with increasing multiplicity as opposed to the TDB algorithm whose performance remains proportional to the degree of the input polynomial. Let us take equation (11) for this comparison. The same input tolerance of 10^{-7} is used for this test, and 2000 sample points are used for the TDB algorithm. The time comparison results are consistent with the above analysis as shown in Table 8.

Degree	IPP	TDB Algorithm
6	8.76	1.34
8	52.32	1.70
10	193.9	2.06

Table 8. Elapsed time comparison (seconds)

The IPP algorithm has several drawbacks. First, in order to use the IPP algorithm, the input equation has to be transformed such that roots should exist within $[0, 1]$. Moreover, the input polynomial has to be represented in the Bernstein form. This conversion itself is an unstable process [2] and special care must be taken not to lose accuracy of coefficients during the conversion, i.e. exact arithmetic is normally needed in this preliminary formulation step. Second, no information on root multiplicity can be obtained from the IPP algorithm. Last, the IPP algorithm cannot assure that all intervals really contain roots. All of these difficulties can be efficiently handled by the TDB algorithm.

5. Bivariate Case

In this section we will give a procedure for the computation of roots, along with their multiplicities, of a bivariate polynomial system.

5.1. A Suitable Change of Coordinates

Let $f(x, y), g(x, y)$ be two real polynomials without common factors, of degrees n, m, respectively. We would, first, like to bring $f(x, y)$ and $g(x, y)$ in the form (4). To achieve that form, we shall define new coordinates $(\bar{x}, \bar{y})$ through the linear isomorphism:

$$x = \bar{x} + k\bar{y}, \qquad y = \bar{y} \tag{14}$$

Let then $\bar{f}(\bar{x}, \bar{y}) = f(\bar{x} + k\bar{y}, \bar{y})$ and $\bar{g} = g(\bar{x} + k\bar{y}, \bar{y})$. The following result is similar to Lemma 2, p. 407 of [13]:

Proposition 5.1 *We can choose a rational number k so that $\bar{f}$ and $\bar{g}$ satisfy the following two conditions:*

CR *$\bar{f}$ and $\bar{g}$ are regular in y,*

CU *whenever two points $(\bar{x}_0, \bar{y}_0)$ and $(\bar{x}_1, \bar{y}_1)$ satisfy $\bar{f} = \bar{g} = 0$, then $\bar{y}_0 = \bar{y}_1$.*

Proof Let $\phi_n(x, y), \psi_m(x, y)$ be the homogeneous terms of f, g of degrees n, m, respectively. Let also $z_i = (x_i, y_i)$ be the roots of $f = g = 0$. Choose a rational number k so that:

$$(i)\ \phi_n(k, 1)\cdot\psi_m(k, 1) \neq 0 \quad \text{and} \quad (ii)\ k \neq \frac{x_i - x_j}{y_r - y_s} \tag{15}$$

for all $i \neq j$ and $r \neq s$. Now the $\bar{y}^n$ in $\bar{f}$ is then $\bar{\phi_n}(0, \bar{y}) = \phi_n(0 + k\bar{y}, \bar{y}) = \phi_n(k, 1)\,\bar{y}^n$. That shows that $\bar{f}$ is regular in y. Similarly, $\bar{g}$ is also regular in y. Also, if we assume that two points $(\bar{x}_0, \bar{y}_0)$ and $(\bar{x}_1, \bar{y}_1)$ with $\bar{y}_0 \neq \bar{y}_1$ satisfy $\bar{f} = \bar{g} = 0$, then the corresponding untransformed points given by $(x', y') = (\bar{x}_0, \bar{y}_0)$ and $(x'', y'') = (\bar{x}_1, \bar{y}_1)$ clearly satisfy $f = g = 0$ and

$$k = \frac{x' - x''}{y' - y''}$$

which contradicts (15ii). ■

Even though Proposition 5.1 gives us an infinite choice for the right k, it is impractical to attempt to apply requirement (15ii) directly in choosing k. There is, however, a computational way of resolving this difficulty. The reader is referred to Remark 3, p. 409 of [13] for all the details involved. From now on, in view of Proposition 5.1, we will assume that the given polynomials $f(x, y)$ and $g(x, y)$ satisfy conditions **CR** and **CU**.

5.1.1. Example Let us assume that we have two real polynomial equations as follows [10]:

$$\begin{aligned} f(x,y) &= \frac{x^2}{4} + y^2 - 1 = 0, \\ g(x,y) &= (x-1)^2 + y^2 - 1 = 0, \end{aligned} \qquad (16)$$

whose zero sets are illustrated in Figure 4. The exact roots

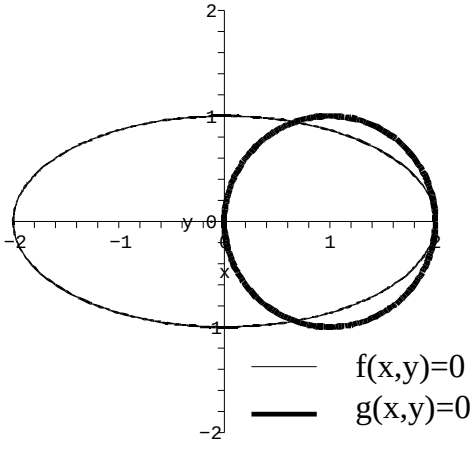

Figure 4. A circle and ellipse, equations (16)

of (16) are:

$$z_1 = \left(\frac{2}{3}, \frac{2}{3}\sqrt{2}\right), z_2 = \left(\frac{2}{3}, -\frac{2}{3}\sqrt{2}\right), z_3 = (2, 0)\,. \quad (17)$$

Roots z_1 and z_2 have multiplicity one and z_3 has multiplicity two. Obviously we can see that equations (16) do not satisfy the **CU** condition. Now, using the linear isomorphism (14), we can rewrite (16) as follows:

$$\begin{aligned} \bar{f}(\bar{x}, \bar{y}) &= \bar{y}^2 + \frac{2k\bar{x}}{k^2+4}\bar{y} + \frac{\bar{x}^2 - 4}{k^2+4} = 0, \\ \bar{g}(\bar{x}, \bar{y}) &= \bar{y}^2 + \frac{2k(\bar{x}-1)}{1+k^2}\bar{y} + \frac{\bar{x}(x-2)}{k^2+1} = 0. \end{aligned} \quad (18)$$

From the subresultant computation [13], we find that

$$\begin{aligned} s_{00} &= 0, \\ s_{11} &= \frac{1}{(1+k^2)(4+k^2)}\left[6k\bar{x} - 2k(k^2+4)\right], \end{aligned}$$

where s_{jj} denotes the j-th subresultant of f and g. We choose $k = 1$ such that $s_{11} \not\equiv 0$. Then equations (18) become

$$\begin{aligned} \bar{f}(\bar{x}, \bar{y}) &= \frac{5}{4}\bar{y}^2 + \frac{1}{2}\bar{x}\bar{y} + \frac{\bar{x}^2}{4} - 1 = 0, \\ \bar{g}(\bar{x}, \bar{y}) &= 2\bar{y}^2 + 2(\bar{x}-1)\bar{y} + \bar{x}^2 - 2\bar{x} = 0, \end{aligned} \quad (19)$$

illustrated in Figure 5. The exact roots of (19) are:

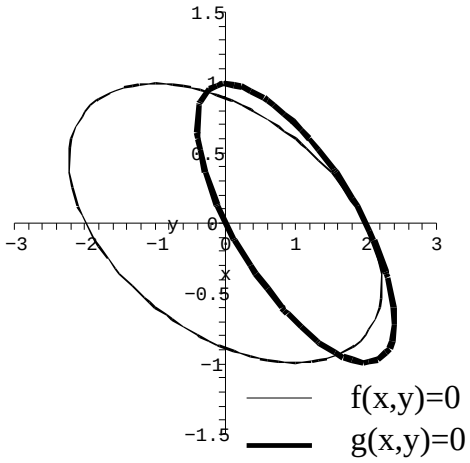

Figure 5. Drawings of (19)

$$\left(\frac{2}{3}(1+\sqrt{2}), -\frac{2}{3}\sqrt{2}\right), \left(\frac{2}{3}(1-\sqrt{2}), \frac{2}{3}\sqrt{2}\right), (2, 0)\,.$$

Therefore, equations (19) satisfy the **CR** and **CU** conditions.

5.2. An Algorithm for Solving a Bivariate Polynomial System

Let $f(x, y), g(x, y)$ be as above and consider $h(x) = \mathrm{Res}_y(f, g)$. Let $x_1, x_2, \cdots, x_r$ be the (distinct) roots of

$h(x)$. Then, we claim that above each such x_i, there is one, and only one, y_i so that the pair $z_i = (x_i, y_i)$ is a root of system (3). Moreover, z_i is a real root if and only if $x_i \in \mathbf{R}$. Indeed, since f, g are regular in y, the lifting property of resultants says that for each $h(x_i) = 0$, there exists a $y_i \in \mathbf{C}$, so that $f(x_i, y_i) = g(x_i, y_i) = 0$; this y_i is unique from condition **CU**. Moreover, if $x_i \in \mathbf{R}$, y_i has to be a real number as well, since the coefficients of f and g are real.

We summarize the above in the following

Remark 5.1 *Let $f(x, y), g(x, y)$ be as in Proposition 5.1, and let $h(x) = Res_y(f, g)$. Then the roots of the system (3) are in an one to one correspondence with the roots of $h(x)$. Moreover, $z_i = (x_i, y_i)$ is a real root of (3) if and only if x_i is a real root of $h(x)$.*

Now suppose that a rectangle $S = [a_1, a_2] \times [a_3, a_4]$ in $\mathbf{R}^2$, contains all real roots of (3). Using our procedure for the univariate case, we assume that we have a partition $a_1 = t_1 < t_2 < t_3 < \cdots < t_k = a_2$ of $[a_1, a_2]$ so that in each subinterval $[t_i, t_{i+1}]$ there exists precisely one (real) root r_i of $h(x)$. Similarly, we also compute the resultant relative to x, $l(y) = Res_x(f, g)$ and using the same procedure we can find a partition $a_3 = s_1 < s_2 < s_3 < \cdots < s_c = a_4$ of $[a_3, a_4]$ such that there exist roots v_j of $l(y) = 0$ in each subinterval $[s_j, s_{j+1}]$. Then we can find a set of kc boxes, T

$$T = \{\alpha_{ij} | \alpha_{ij} = [t_i, t_{i+1}] \times [s_j, s_{j+1}]\}.$$

Remark 5.2 *Let us assume that we have a system of equations (3) and $\alpha_{ij} = [t_i, t_{i+1}] \times [s_j, s_{j+1}]$ encloses a real root of (3). Then, the following must be true*

$$0 \in f([t_i, t_{i+1}], [s_j, s_{j+1}]) \times g([t_i, t_{i+1}], [s_j, s_{j+1}]),$$

where f and g are evaluated in interval arithmetic.

Using Remark 5.2, we can sort out the boxes a_{ij} which enclose roots and associate the corresponding multiplicity.

5.2.1. Examples We take the system of equations (19) as our first example. The resultants of (19) relative to y and x are

$$\begin{aligned} h(x) &= \frac{1}{16}(9x^2 - 12x - 14)(x - 2)^2, \\ l(y) &= \frac{1}{16}y^2(9y^2 - 8). \end{aligned} \quad (20)$$

Using the TDB algorithm with a tolerance 10^{-7}, we can find the roots of $h(x) = 0$ and $l(y) = 0$ in Tables 9 and 10, respectively. Now, we can compute nine boxes from Tables 9 and 10. Using Remark 5.2, we have the following roots and multiplicities, d as in Table 11.

A next example is a system of bivariate polynomials given as follows [15]:

$$\begin{aligned} f(x, y) &= x^3 - 3x^2 + 5x - 4 + y^3 \\ &\quad -3y^2 + 5y - 2xy = 0, \\ g(x, y) &= 2x^3 - 2x^2 + x - 4 - 4x^2y + 2xy \\ &\quad +9y + 3xy^2 - 8y^2 + y^3 = 0, \end{aligned} \quad (21)$$

illustrated in Figure 6. The resultants $h(x)$ and $l(y)$ are given as

$$\begin{aligned} h(x) &= 56x^9 - 704x^8 + 3880x^7 - 12304x^6 \\ &\quad +24744x^5 - 32736x^4 + 28504x^3 \\ &\quad -15760x^2 + 5024x - 704. \end{aligned} \quad (22)$$

$$\begin{aligned} l(y) &= -56y^9 + 608y^8 - 2824y^7 + 7312y^6 \\ &\quad -11496y^5 + 11136y^4 - 6328y^3 \\ &\quad +1744y^2 - 32y - 64. \end{aligned} \quad (23)$$

We can find roots of $h(x) = 0$ and $l(y) = 0$ using the TDB algorithm with a tolerance 10^{-7}, and apply Remark 5.2 to have roots and multiplicities, d, of (21) as in Table 12.

Roots	d
[1.99999995,2.00000003]	2
[-0.27614243,-0.27614236]	1
[1.60947569,1.60947576]	1

Table 9. Roots and multiplicities of $h(x) = 0$

Roots	d
[-7.4506e-08,7.4506e-08]	2
[-0.94280906,-0.94280899]	1
[0.94280899,0.94280906]	1

Table 10. Roots and multiplicities of $l(y) = 0$

5.3. Implementation

The algorithm proposed in Section 5 requires an expression of a resultant of two input polynomial equations. This resultant computation must be performed in exact arithmetic to maintain all topological and numerical properties of roots of the polynomial system. We can use a resultant computation algorithm [4] or any type of computer algebra system such as *Maple*. Also, since we know that a resultant is equivalent to the determinant of the Sylvester matrix

Root (x,y)	d
[1.99999995, 2.00000003]x[-7.4506e-8,7.4506e-8]	2
[-0.27614243, -0.27614236]x[0.94280899, 0.94280906]	1
[1.60947569, 1.60947576]x[-0.94280906, -0.94280899]	1

Table 11. Roots and multiplicities of (19)

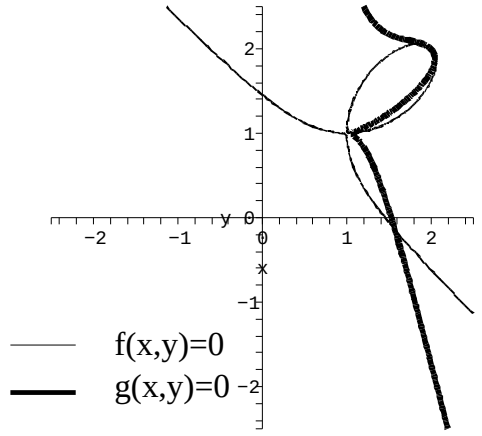

Figure 6. Drawings of (21)

Root (x,y)	d
[0.999999978, 1.00000001]x[0.99999994, 1.00000001]	5
[1.57142855, 1.57142859]x[-0.142857209, -0.142857134]	1
[1.99999999, 2.00000003]x[1.99999996, 2.00000003]	3

Table 12. Roots and multiplicities of (21)

[7] of two bivariate polynomials, we can treat the determinant of the matrix as a function which is provided as input to the TDB algorithm.

5.4. Application

Similarly to the univariate polynomial case in Section 3.2.2, a significant improvement over the IPP algorithm can be also achieved by using the procedure for the bivariate polynomial system. Namely, we can eliminate spurious roots and provide the multiplicity information of each root for a bivariate polynomial system. Assume that we have two real polynomials $f(x,y) = g(x,y) = 0$, which satisfy Proposition 5.1. We apply the IPP algorithm to compute the roots $z_i = ([x_i],[y_i])$, $(i = 1,\cdots,s)$ of the system. Then, we compute the resultant of f and g with respect to y, $h(x) = Res_y(f,g)$. The next step is to compute all roots $[b_1],\cdots,[b_r]$ of $h(x) = 0$ and their multiplicities $d_1,\cdots,d_r$ by using the TDB algorithm in Section 4. After finding the roots of $h(x) = 0$ we choose $[b_1]$ and select z_j whose x component overlaps with $[b_1]$. We repeat this process for all $[b]$'s. Then, the selected z's are consolidated and reported as roots of the bivariate polynomial system with their multiplicity.

This procedure can be illustrated with an example. Let

Root (x)	d
[0.59999996, 0.60000002]	2
[0.50236893, 0.50236899]	1
[0.03096437, 0.03096443]	1

Table 13. Roots and multiplicities of $h(x) = 0$

us assume that we have a system of bivariate polynomials as follows:

$$\begin{aligned} f(x,y) &= \frac{5}{4}y^2 + \frac{1}{2}xy - \frac{13}{10}y + \frac{1}{4}x^2 \\ &\quad -\frac{3}{10}x + \frac{111}{400} = 0, \\ g(x,y) &= 2y^2 + 2xy - \frac{27}{10}y + x^2 \\ &\quad -\frac{17}{10}x + \frac{91}{100} = 0, \end{aligned} \tag{24}$$

illustrated in Figure 7. The resultant of (24) with respect to

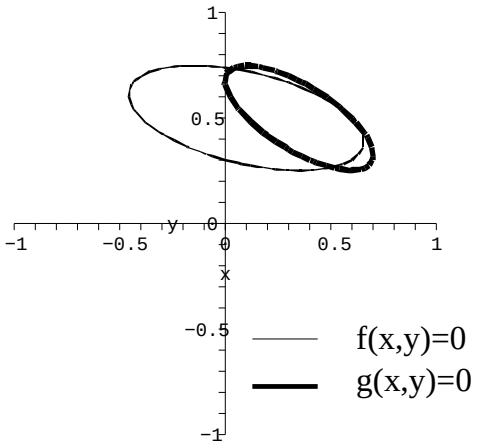

Figure 7. Drawings of (24)

y is

$$h(x) = \frac{9}{16}x^4 - \frac{39}{40}x^3 + \frac{457}{800}x^2 - \frac{237}{2000}x + \frac{63}{20000}. \tag{25}$$

The roots and multiplicities d of (25) are obtained as in Table 13 by using the TDB algorithm in Section 4 with a tolerance 10^{-7}. The IPP algorithm using the same tolerance 10^{-7} produces 4138 boxes that have not been rejected as not containing roots. Using the roots in Table 13, we can reduce the number to eight roots and after consolidation of the eight roots, we have the roots in Table 14 along with their multiplicities.

Root (x,y)	d
[0.59999987, 0.60000012],[0.49999987,0.50000015]	2
[0.50236888, 0.50236909],[0.26429769,0.26429779]	1
[0.03096428, 0.03096447],[0.73570220,0.73570230]	1

Table 14. Roots and multiplicities of (24)

6. Conclusions

In this paper we presented a method that relies on the notion of local degree for the computation of single, as well as, multiple roots of univariate and bivariate polynomial systems. Even though most of the ideas presented here were in existence for some time, it is the first time that are used in this context.

It is apparent from the presentation that multiple roots are difficult to deal with. Indeed, most of the existent root finding algorithms experience difficulties in dealing with such roots.

Our method, however, is successful in isolating and computing multiple roots of univariate and bivariate polynomial systems. This method provides the basis for further research needed in addressing the general problem of multiple roots of nonlinear polynomial systems in n variables.

Acknowledgment

The authors thank T. J. Peters for his comments. This work was supported in part by National Science Foundation (grants DMS-0138098 and CCR-0231511).

References

[1] M. C. Carpentier and A. F. Dos Santos. Solution of equations involving analytic functions. *Journal of Computational Physics*, 45:210–220, 1982.

[2] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, June 1988.

[3] P. Henrici. *Applied and Computational Complex Analysis*. John Wiley, New York, 1974.

[4] D. Manocha and J. F. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15:99–122, 1993.

[5] D. Manocha and J. W. Demmel. Algorithms for intersecting parametric and algebraic curves II: Multiple intersections. *Graphical Model and Image Processing*, 57(2):81–100, 1995.

[6] J. M. McNamee. A bibliography on roots of polynomials. *Journal of Computational and Applied Mathematics*, 47(3):391–394, 1993.

[7] M. Mignotte. *Mathematics for Computer Algebra*. Springer-Verlag, New York, 1992.

[8] H. M. Möller and H. J. Stetter. Multivariate polynomial equations with multiple zeros solved by matrix eigenproblems. *Numerische Mathematik*, 70(3):311–329, 1995.

[9] S. Moritsugu and K. Kuriyama. On multiple zeros of systems of algebraic equations. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pages 23–30, Vancouver, BC, Canada, 1999. ACM Press.

[10] N. M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer-Verlag, Heidelberg, February 2002.

[11] S. M. Rump. Ten methods to bound multiple roots of polynomials. *Journal of Computational and Applied Mathematics*, 156(2):403–432, 2003.

[12] T. Sakkalis. The Euclidean algorithm and the degree of the Gauss map. *SIAM Journal on Computing*, 19(3):538–543, 1990.

[13] T. Sakkalis and R. Farouki. Singular points of algebraic curves. *Journal of Symbolic Computation*, 9:405–421, 1990.

[14] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10(5):379–405, October 1993.

[15] R. J. Walker. *Algebraic Curves*. Springer-Verlag, New York, 1978.

[16] H. S. Wilf. A global bisection algorithm for computing the zeros of polynomials in the complex plane. *Journal of the Association for Computing Machinery*, 25(3):415–420, July 1978.

[17] X. Ying and I. N. Katz. A reliable argument principle algorithm to find the number of zeros of an analytic function in a bounded domain. *Numerische Mathematik*, 53:143–163, 1988.