On an application of symbolic computation and computer graphics to root-finders: the case of multiple roots of unknown multiplicity

Ivan Petković[a,*] , **Beny Neta**[b]

[a] *Faculty of Electronic Engineering, University of Niš, 18000 Niš, Serbia*

[b] *Naval Postgraduate School, Department of Applied Mathematics, Monterey, CA 93943, United States*

Abstract

The contemporary powerful mathematical software enables a new approach to handling and manipulating complex mathematical expressions and other mathematical objects. Particularly, the use of symbolic computation leads to new contribution to constructing and analyzing numerical algorithms for solving very difficult problems in applied mathematics and other scientific disciplines. In this paper we are concerned with the problem of determining multiple zeros when the multiplicity is not known in advance, a task that is seldom considered in literature. By the use of computer algebra system *Mathematica*, we employ symbolic computation through several programs to construct and investigate algorithms which both determine a sought zero and its multiplicity. Applying a recurrent formula for generating iterative methods of higher order for solving nonlinear equations, we construct iterative methods that serve (i) for approximating a multiple zero of a given function f when the order of multiplicity is unknown and, simultaneously, (ii) for finding exact order of multiplicity. In particular, we state useful cubically convergent iterative sequences that find the exact multiplicity in a few iteration steps. Such approach, combined with a rapidly convergent method for multiple zeros, provides the construction of efficient composite algorithms for finding multiple zeros of very high accuracy. The properties of the proposed algorithms are illustrated by several numerical examples and basins of attraction.

AMS Mathematical Subject Classification (2010): 65H05, 65D18, 68W30, 33F05, 65Gxx

Keywords: Nonlinear equations, iterative methods, multiple zeros, symbolic calculation, computer graphics.

1 Introduction

The development of digital computers, circa 1970, has enabled easy and successful computations in many scientific disciplines such as engineering disciplines, physics, chemistry, communication, biology, education, astronomy, geology, banking, business, insurance, health care, social science, as well as many other fields of human activities. In this paper we pay our attention to applied mathematics, in particular, to the construction of algorithms in numerical analysis, and a specific application of computers – symbolic computation. In the course of the development of numerical methods, it turned out in the last quarter of the 20th century that further development of new algorithms of higher efficiency and greater accuracy was not possible due to the lack of fast hardware and advanced software. At the beginning of the 21st century the rapid development of computer power and accessibility, computer multi-precision arithmetics and symbolic computation enabled the construction, testing and analysis of very efficient numerical algorithms, even "confirmation

*Corresponding author
E-mail addresses: ivan.petkovic@elfak.ni.ac.rs (I. Petković), bneta@nps.edu (B. Neta)

analytically derived results" [1]. Simply, symbolic computation replaced lengthy hand derivations and manipulation with computer-based derivation and manipulation.

Symbolic computation, employed in solving mathematical problems, is concerned with software for handling and manipulating mathematical expressions and other mathematical objects. It is important to emphasize that these expressions contain variables in general form such as $f(a, b, c, ...)$, where $a,\ b,\ c$ are not numerical values and f may or may not be given explicitly. Various manipulations (automatic simplification of expressions, differentiation, indefinite integration, polynomial decomposition, polynomial factorization, etc.) treat these variables as symbols (hence the name *symbolic computation*) and provide *exact* computation. It is superfluous to emphasize that, in the case of complex and lengthly expressions, symbolic computation is the only tool for solving given problems; their solution and analysis would not be possible by a classic "paper-and-pencil" fashion. Note that software applications that execute symbolic computation are most frequently called *computer algebra systems* (shorter CAS) although some authors make distinction between "symbolic computation" and "computer algebra," see, e.g., [2]. For more details on symbolic computation and its applications see the book [3].

As mentioned in [4], "newer generation of mathematicians and computer scientists can really take advantage of computer aided research supported by the modern CAS." Among the major general purposes CAS are certainly Mathematica and Maple, although Axiom, GAP, Maxima, Sage and SymPy are very useful within their symbolic functionality. All of these CAS are available of the platforms Windows, Mac OS X and Linux (with the exception of Sage which works in Windows as "virtual machine"). All of these computational packages perform sophisticated mathematical operations.

In this paper we present a class of iterative methods for the determination of multiple zeros of functions when the multiplicity is not known in advance. One should say that there is a vast literature concerned with iterative methods for finding multiple zeros were developed, see. eg., [5]–[13]. However, in most papers one assumes that the order of multiplicity is known. Procedures for the determination of exact value of multiplicity m and sufficiently close initial approximation x_0 were very seldom discussed in the literature. The knowledge of multiplicity m and a good initial approximation x_0 are two very important tasks and should be a composite part of any root-finder. The latter task was considered in some recent papers and books, see, for instance, [14]–[18].

Here we are concerned with generating iterative methods of higher order for solving nonlinear equations. More precisely, we study iterative methods for finding a multiple zero of a given function f when the order of multiplicity is unknown (Section 2). At the same time, we develop iterative methods for approximating multiplicity and prove their cubic convergence (Sections 3 and 4). Such approach, combined with fourth order two-point method for multiple zeros proposed in [7], provides the construction of an efficient algorithm for finding multiple zeros of very high accuracy, which is presented in Section 5. Throughout the text it is assumed that order of multiplicity is a positive integer. The case of fractional order is discussed in Section 7 and demonstrated on a numerical example.

It is worth noting that numerical experiments and the study of computational efficiency of considered root-finding methods based on convergence order and computational costs (see [19, p. 12]) are not often sufficient to give a real estimation of the quality of these methods and, consequently, their proper ranking. For this reason, a powerful tool for comparison and analysis of root-finding algorithms, using basins of attraction, is presented in Section 6 for six examples. This approach gives a much better insight into visualization in approximating function zeros, especially in regard to areas of convergence. The mentioned tools provide considerably better understanding of iterative processes.

Properties of the presented iterative methods are demonstrated on numerical examples in Section 7. Our main tools in developing and analyzing these methods are symbolic computation realized through several programs (implemented in computer algebra system *Mathematica*) and dynamic study by plotting basins of attraction for four methods and seven polynomials.

2 Generator of root-finders

Approximating zeros of a given scalar function f belongs to the most important problems appearing not only in applied mathematics but also in many disciplines of physics, finance, engineering branches, and so on. Solution of the mentioned task often requires from the user to combine numerical analysis and computing science, first of all symbolic computation (assuming, of course, the use of necessary computer hardware). During the last three centuries, many one-point methods were stated, such as Newton's, Halley's, Laguerre's, Chebyshev's method. Particular attention is due to the so-called the Traub-Schröder basic sequence (often called Schröder's family of the first kind, see [19], [20]) and the Schröder-König sequence or Schröder's family of the second kind [20]. Both families explicitly depend of f and its $n-1$ derivatives and have the order at most n. The latter of these families will be considered in this section.

Let f be a given function with isolated zero α in some interval. If α is a zero of multiplicity m, then we have the representation $f(x) = (x-\alpha)^m p(x),\ p(\alpha) \neq 0$. An iterative method for approximating a single (simple or multiple zero) α will be written in the form $x_{k+1} = g(x_k)$, assuming that the initial approximation x_0 is known. The applied iterative method will produce the sequence $\{x_k\}$ that converges to the zero α if x_0 is reasonably close to α. Iterative methods for finding multiple zeros usually require the knowledge of order of multiplicity m, otherwise, many of them will converge only *linearly*. In this paper we mainly consider a class of methods for finding multiple zeros which do not require the multiplicity but their order of convergence is higher than 1.

We start with the following assertion proved in [21].

Theorem 1. *Let $x_{k+1} = g_n(x_k) \quad (k = 0, 1, \ldots)$ be an iterative method of order n for finding a simple or multiple zero α of a given function f (sufficiently many times differentiable). Then the iterative method*

$$x_{k+1} = g_{n+1}(x_k) := x_k - \frac{x_k - g_n(x_k)}{1 - \dfrac{1}{n} g_n'(x_k)} \quad (n \geq 2;\ k = 0, 1, \ldots) \tag{1}$$

has the order of convergence $n+1$.

Remark 1. The iterative formula (1) was derived in [22] and later in [21] using a different approach. In the latter paper some useful properties of this formula were presented.

Remark 2. The ability of the iterative formula (1) to generate root-finding methods of an arbitrary order of convergence for both simple and multiple zeros is the main advantage of the generating formula (1). For this very useful property it will be sometimes called *accelerating generator* and denoted by $\mathcal{AG}\,(1)$, the term introduced in [21].

Remark 3. The equivalence of the iterative method (1) and the Schröder-König method was proved in [23] for simple zeros taking Newton's method as the staring method. We recall that the Schröder-König method is defined by

$$x_{k+1} = x_k - \frac{R_{n-2}(x_k)}{R_{n-1}(x_k)} \quad (n \geq 2;\ k = 0, 1, \ldots), \tag{2}$$

where R_k is calculated from the recursive relation

$$R_0 = 1, \quad R_n(x) = \sum_{\lambda=1}^{n} (-1)^{\lambda+1} A_\lambda(x) R_{n-\lambda}(x), \quad \text{where} \ \ A_\lambda(x) = \frac{f^{(\lambda)}(x)}{\lambda! f(x)}. \tag{3}$$

Both iterative formulas (1) and (2) generate the same methods $g_3,\ g_4,\ldots$ for finding simple zeros starting from the Newton method $x_{k+1} = g_2(x_k) = x_k - f(x_k)/f'(x_k)$. However, Theorem 1 does not require any assumption on the order of multiplicity of the zero α which means that the iterative formula (1) can generate methods for finding multiple zeros too, without alternation to its structure. It is sufficient to start with a suitably chosen initial iterative function $g_n(x)$ $(n \geq 2)$ assuming that the order of multiplicity is known. Besides, $\mathcal{AG}\,(1)$ has additional advantage since it can start from any method of order n, not necessary from the Newton method, as in the case of the Schröder-König method (2). This is evident from the fixed structure of the recurrent relation $R_0 = 1,\ R_1 = f'(x)/f(x)$ arising from (3).

The main goal of this paper is to consider some iterative formulas for finding multiple zeros of a given differentiable function when the order of multiplicity is not known in advance, together with efficient method for finding the order of multiplicity. The combination of these two methods leads to a very efficient two-point iterative method for finding multiple zero of the known multiplicity.

First, we apply generating accelerator $\mathcal{AG}\,(1)$ to produce iterative methods for finding simple or multiple zero of a function f. If α is a multiple zero of $f(x)$, then α is obviously a simple zero of the function $u(x) = f(x)/f'(x)$. Indeed, starting from the factorization $f(x) = (x-\alpha)^m p(x),\ p(\alpha) \neq 0$, and assuming that α has the order of multiplicity m, we obtain

$$\frac{f(x)}{f'(x)} = \frac{(x-\alpha)^m p(x)}{m(x-\alpha)^{m-1}p(x) + (x-\alpha)^m p'(x)} = \frac{(x-\alpha)p(x)}{mp(x) + (x-\alpha)p'(x)} = (x-\alpha)q(x), \quad q(\alpha) \neq 0.$$

Applying Newton's method $N(x) = x - \varphi(x)/\varphi'(x)$ to the function $\varphi(x) = u(x)$, we obtain the iterative method

$$g_2(x) = x - \frac{f(x)f'(x)}{f'(x)^2 - f(x)f''(x)} = x - \frac{u(x)}{1 - 2A_2(x)u(x)}, \tag{4}$$

where A_r is defined in (3). The iterative method (4) has the order of convergence two and it was derived for the first time by Schröder in his paper [20].

The construction of methods of higher order is very hard and tedious work since handling with very complicated expressions is needed. For this reason we will use symbolic computation for their construction, as presented by a program written in CAS *Mathematica*, developed by Wolfram Research company [24]. For simplicity, we omit argument x in $u(x)$ and $A_k(x)$.

PROGRAM 1: METHODS FOR MULTIPLE ZEROS WITH UNKNOWN MULTIPLICITY

```
Clear["Global‘*"];
r=5; t[0]=f[x];
Do[t[k]=D[t[k-1],x],{k,1,r}];
s[2]=t[0]*t[1]/(t[1]^2-t[0]*t[2]); g2[2]=x-s[2];
Do[s[k]=Simplify[(x-g[k-1])/(1-D[g[k-1],x]/(k-1))]; g[k]=x-s[k],{k,3,r}];
f[x]=u*t[1]; Do[t[k]=k!*A[k]*t[1],{k,2,r}];
Do[w[n]=FullSimplify[s[n]/.Table[Derivative[k][f][x]->t[k],{k,2,n}]],{n,2,r}];
Do[g[n]=x-w[n];Print["g(",n ",g[n]],{n,2,r}]
```

$$
\begin{aligned}
g(2) &= x - \frac{u}{1-2uA[2]} \\
g(3) &= x - \frac{u-2A[2]u^2}{1-3A[2]u+3A[3]u^2} \qquad (5)\\
g(4) &= x - \frac{u(1-3u(A[2]-uA[3]))}{1-4A[2]u+2(A[2]^2+2A[3])u^2-4A[4]u^3} \qquad (6)\\
g(5) &= x - \frac{u-4A[2]u^2+2u^3(A[2]^2+2A[3])-4A[4]u^4}{1-5A[2]u+5(A[2]^2+A[3])u^2-5(A[2]A[3]+A[4])u^3+5A[5]u^4} \qquad (7)
\end{aligned}
$$

Remark 4. The iterative function $g_3(x)$ can be obtained by applying Halley's method

$$H(x) = x - \frac{\varphi(x)}{\varphi'(x) - \dfrac{\varphi(x)\varphi''(x)}{2\varphi'(x)}}$$

to the function $u(x) = f(x)/f'(x)$. We refer to (5) as Halley-like method. This method was derived in [21] but it is possible that the iteration function $g_3(x)$ had been stated earlier; the authors have not found a reliable source.

PROGRAM 1 can generate new sequences $g_6(x),\ g_7(x), \ldots$ but we finished with $g_5(x)$ since the next formulas obtained by $\mathcal{AG}\,(1)$ are too cumbersome and inefficient in practice. From the computational cost of view, even the method (7) is expensive. We emphasize that the derivation of the formulas (6) and (7) by classical "pencil-and-paper" method is laborious since very complicated expressions appear.

It was proved in many references that the order of convergence of Schröder's iterative methods (4) is two. According to Theorem 1 it follows that the order of (5), (6) and (7) is three, four and five, respectively.

3 Iterative methods for finding order of multiplicity

Let us introduce the error $\varepsilon_k = x_k - \alpha$ of the approximation x_k to the zero α of f. Later we will need the following well-known assertion:

Theorem 2. (Traub [19, Theorem 2.2]) *Let*

$$x_{k+1} = \varphi(x_k) \quad (k = 0, 1, \ldots) \tag{8}$$

be an iterative method such that $\varphi^{(n)}$ is continuous in the neighborhood of the zero α of a given function f. Then φ is of order n if and only if

$$\varphi(\alpha) = \alpha, \quad \varphi'(\alpha) = \cdots = \varphi^{(n-1)}(\alpha) = 0, \quad \varphi^{(n)}(\alpha) \neq 0. \tag{9}$$

Furthermore,

$$\frac{\varepsilon_{k+1}}{\varepsilon_k^n} \to \frac{\varphi^{(n)}(\alpha)}{n!}. \tag{10}$$

The limit

$$\mathcal{C}(\varphi) = \frac{\varphi^{(n)}(\alpha)}{n!} = \lim_{k\to\infty} \frac{\varphi(x_{k+1}) - \alpha}{(\varphi(x_k) - \alpha)^n}$$

in (10) is called the *asymptotic error constant* (shorter AEC) in Traub's sense for the method φ, see [25]. We need the asymptotic error constant of the method (5) and we will find it by symbolic computation using Theorem 2.

The motivation for the construction of iterative formulas for finding the order of multiplicity of α arises from the formula

$$S_1(x) = x - m\frac{f(x)}{f'(x)} \tag{11}$$

for approximating a multiple zero of the known multiplicity m and the formula (4), which can be written in the form

$$S_2(x) = x - \frac{f(x)}{f'(x)} \cdot \frac{f'(x)^2}{f'(x)^2 - f(x)f''(x)} = x - \frac{f(x)}{f'(x)} \cdot \mu_2(x). \tag{12}$$

Obviously,

$$\mu_2(x) = \frac{f'(x)^2}{f'(x)^2 - f(x)f''(x)} = \frac{1}{u'(x)}. \tag{13}$$

The letter S in (11) and (12) stands for Schröder who derived both formulas in his pioneering paper [20] in 1870. Comparing (11) and (12) we assume that the factor $\mu_2(x)$ could present a formula for approximating the order of multiplicity m. Actually, this is true since $\mu_2(x)$, defined by (13), is the known Lagouanelle's formula [26] for approximating order of multiplicity. Indeed, setting $f(x) = (x-\alpha)^m p(x)$ $(p(\alpha) \neq 0)$, we find

$$\mu_2(x) = \frac{1}{u'(x)} = \frac{(mp(x) + (x-\alpha)p'(x))^2}{mp(x)^2 + (x-\alpha)^2 p'(x)^2 - (x-\alpha)^2 p(x)p''(x)}$$

and, hence,

$$\mu_2(x) = \frac{1}{u'(x)} \;\to\; m \quad \text{when} \quad x \to \alpha.$$

We wish to show that such approach is valid for the iterative formulas arising from higher-order methods generated by $\mathcal{AG}\,(1)$.

We need the following assertion given in [19, Theorem 2-5].

Theorem 3. *Let $\phi(x)$ be an iterative function of order n, for some set of multiplicities m. Then for these values of m there exists a function $\omega(x)$ such that*

$$\phi(x) = x - u(x)\omega(x), \quad \omega(\alpha) \neq 0.$$

Theorem 3 gives an idea for developing suitable formulas for the approximation of order of multiplicity. In this paper we will concentrate to the formula of the form

$$g_n(x) = x - u(x)\mu_n(x), \tag{14}$$

where $g_n(x)$ $(= \phi(x))$ defines the iterative method of order n generated by $\mathcal{AG}\,(1)$ starting from Newton-like method g_2 given by (4). Obviously,

$$\mu_n(x) = \omega(x) = \frac{x - g_n(x)}{u(x)}.$$

Theorem 4. *$\mu_n(x) \to m$ when $x \to \alpha$.*

Proof. From (14) we have

$$g_n'(x) = 1 - u'(x)\mu_n(x) - u(x)\mu_n'(x), \quad x - g_n(x) = u(x)\mu_n(x). \tag{15}$$

According to (15) $\mathcal{AG}\,(1)$ can be written in the form

$$g_{n+1}(x) = x - \frac{u(x)\mu_n(x)}{1 - \dfrac{1 - u'(x)\mu_n(x) - u(x)\mu_n'(x)}{n}}, \tag{16}$$

or, in the spirit of (14), as

$$g_{n+1}(x) = x - u(x)\mu_{n+1}(x),$$

where

$$\mu_{n+1}(x) = \frac{\mu_n(x)}{1 - \dfrac{1 - u'(x)\mu_n(x) - u(x)\mu_n'(x)}{n}}. \tag{17}$$

Now we use the induction and assume that $\mu_n \to m$ when $x \to \alpha$ for some $n \geq 2$. Let $x \to \alpha$. Then

$$u(x) = \frac{f(x)}{f'(x)} = \frac{(x-\alpha)p(x)}{mp(x) + (x-a)p'(x)} \to 0.$$

Previously we have proved that $1/u'(x) \to m$ (Lagouanelle's formula (13)) so that $u'(x)\mu_n(x) \to 1$ when $x \to \alpha$. Having in mind these facts we have that

$$g_n'(x) = 1 - u'(x)\mu_n(x) - u(x)\mu_n'(x) \to 0 \ \text{ as } \ x \to \alpha$$

and from (17) we find that $\mu_{n+1} \to m$ when $x \to \alpha$. Since the inductive assumption is valid for $n = 2$ (Lagouanelle's formula (13)) we conclude that $\mu_n \to m$ for arbitrary $n \geq 2$. □

According to the iterative formulas (4), (5) and (6) we obtain the following formulas for the approximation of order of multiplicity:

$$\mu_2(x) = \frac{1}{1 - 2A_2(x)u(x)} \quad \text{(Lagouanelle's formula)}, \tag{18}$$

$$\mu_3(x) = \frac{1 - 2A_2(x)u(x)}{1 - 3A_2(x)u(x) + 3A_3(x)u(x)^2}, \tag{19}$$

$$\mu_4(x) = \frac{1 - 3A_2(x)u(x) + 3A_3(x)u(x)^2}{1 - 4A_2(x)u(x) + 2\Big[2A_3(x) + A_2(x)^2\Big]u(x)^2 - 4A_4(x)u(x)^3}. \tag{20}$$

In this paper we are concentrating to the iterative formula (5) of Halley-like type for finding zero approximations and the formulas (18) and (19) as parts of coupled algorithms for finding multiple zeros. Obviously, the formula (20) is not convenient since additional calculation of $f^{(4)}$ is required and we will restrict ourselves to the formulas (18) and (19).

Our root-finder consists from two parts (I) and (II). Part (I) deals with a couple of sequences for finding (i) approximations to the multiple zeros (5) and (ii) approximations to the multiplicity defined by (18) or (19) (in this order). After the determination of both approximations to the zero approximation x_k and the order of multiplicity m with sufficiently high precision, the process of refinement continues with Part (II) consisting of an efficient two-point method dealing with the known multiplicity to improve additionally the accuracy of the zero approximation.

Before establishing the described algorithms we need the asymptotical error constant of the Halley-like iterative method (5). We will use the following development of a function f about the zero α of multiplicity m

$$f(x) = \frac{f^{(m)}(\alpha)}{m!}\Big(1 + C_1\varepsilon + C_2\varepsilon^2 + C_3\varepsilon^3 + \cdots\Big), \quad C_k = \frac{m!}{(m+k)!}\frac{f^{(m+k)}(\alpha)}{f^{(m)}(\alpha)} \ \ (k = 1, 2, \ldots),$$

with $\varepsilon = x - \alpha$.

Theorem 5. *Let x_0 be sufficiently close to a simple or multiple zero of a function f. Then the iterative method $g_3(x_k)$ defined by* (5) *converges cubically and*

$$\frac{g_3(x_k)-\alpha}{(x_k-\alpha)^3} \to \frac{2C_2 - C_1^2}{m}. \tag{21}$$

Proof. We use symbolic computation in the program *Mathematica* since expressions appearing in the convergence analysis are pretty cumbersome and lengthy.

Let $\varphi(x)$ is the iteration function defined in Theorem 2 and let $\psi(x) = \varphi(x) - \alpha$. Then the condition (9) reduces to

$$\psi(\alpha) = 0,\ \psi'(\alpha) = 0,\ \psi''(\alpha) = 0, \ldots, \psi^{(n-1)}(\alpha) = 0,\ \psi^{(n)}(\alpha) \neq 0.$$

We introduce the notation `fx`$= f(x)$, `fx1`$= f'(x)$, `e` $= \varepsilon = x - \alpha$; `fma` $= f^{(m)}(\alpha)$; `e1` $= \hat{\varepsilon} = \psi(x) = g_3(x) - \alpha$ and let

$$\texttt{e1} = \psi(x) = H_0 + H_1\varepsilon + H_2\varepsilon^2 + H_3\varepsilon^3 + \cdots$$

be Taylor's series of `e1` about the point 0. Now we use the following program.

PROGRAM 2: CONVERGENCE RATE AND AEC OF HALLEY-LIKE METHOD

```
Clear["Global`*"];
fx = fma/m! e^m (1 + C1e + C2e^2 + C3e^3);
f1x = D[fx, e] // Simplify;
u = Simplify[Series[fx/f1x, {e, 0, 4}], Assumptions -> Element[m, Integers]];
u1 = D[u, e]; u2 = D[u1, e];
e1 = Series[e - u/(u1 - (u u2)/(2 u1)), {e, 0, 4}];
H0 = Coefficient[e1, e, 0] // Simplify
Out[H0] = 0
H1 = Simplify[Coefficient[e1, e, 1], Assumptions -> Element[m, Integers]]
Out[H1] = 0
H2 = Simplify[Coefficient[e1, e, 2], Assumptions -> Element[m, Integers]]
Out[H2] = 0
H3 = Simplify[Coefficient[e1, e, 3], Assumptions -> Element[m, Integers]]
Print["H3=",H3]
```

$$H3 = \frac{2C2 - C1^2}{m}$$

According to the outcomes given above we conclude that

$$g_3(x) - \alpha = \hat{\varepsilon} = H_3\varepsilon^3 + O(\varepsilon^4), \quad H_3 = \frac{2C_2 - C_1^2}{m}.$$

and, introducing the iteration index k,

$$\lim_{k\to\infty} \frac{g_3(x_k)-\alpha}{(x_k-\alpha)^3} = H_3 = \frac{2C_2 - C_1^2}{m}. \tag{22}$$

From (22) there follows that the Halley-like method $x_{k+1} = g_3(x_k)$, given by (5), is of third order. Besides, the asymptotic error constant is

$$H_3 = \frac{2C_2 - C_1^2}{m} = \frac{1}{m(m+1)}\left(\frac{2}{m+2}\frac{f^{(m+2)}(\alpha)}{f^{(m)}(\alpha)} - \frac{1}{m+1}\left(\frac{f^{(m+1)}(\alpha)}{f^{(m)}(\alpha)}\right)^2\right). \quad \Box \tag{23}$$

Note that from the last relation we have

$$\varepsilon_{k+1} = x_{k+1} - \alpha = g_3(x_k) - \alpha = H_3\varepsilon_k^3 + O(\varepsilon_k^4). \tag{24}$$

Using slight modifications of PROGRAM 2 it is easy to prove that the order of convergence of the iterative methods (6) and (7) is four and five, respectively.

4 Coupled algorithms for multiple zeros

Now we state two algorithms that are constituted by two sequences.

Algorithm 1:

$$\begin{cases} x_{k+1} = g_3(x_k) = x_k - \dfrac{u(x_k)\Big[1 - 2A_2(x_k)u(x_k)\Big]}{1 - 3A_2(x_k)u(x_k) + 3A_3(x_k)u(x_k)^2} \\ \mu_2(x_{k+1}) = \dfrac{1}{1 - 2u(x_{k+1})A_2(x_{k+1})} \end{cases} \tag{25}$$

Algorithm 2:

$$\begin{cases} x_{k+1} = g_3(x_k) = x_k - \dfrac{u(x_k)\Big[1 - 2A_2(x_k)u(x_k)\Big]}{1 - 3A_2(x_k)u(x_k) + 3A_3(x_k)u(x_k)^2} \\ \mu_3(x_{k+1}) = \dfrac{1 - 2A_2(x_{k+1})u(x_{k+1})}{1 - 3A_2(x_{k+1})u(x_{k+1}) + 3A_3(x_{k+1})u(x_{k+1})^2} \end{cases} \tag{26}$$

Remark 5. The second step in both Algorithm 1 and Algorithm 2 serves for finding the order of multiplicity using improved approximations calculated in the first step. At the first sight, it seems that the second step could work with functions f, f', f'' (for Algorithms 1) and f, f', f'', $f^{(4)}$ (for Algorithm 2), already evaluated in the first step at the point x_k. However, note that the values calculated at x_{k+1} (as in (25) and (26)) are *reused* in the next iteration for the first step in Algorithms 1 and 2. The exception is the last iteration when the stopping criterion concerning the first step is satisfied, which eliminates the need for the second step (see the criterion (2) in the flow chart in Figure 1). A number of numerical examples have shown that the values of the function f and its derivatives at the point x_k do not provide sufficiently accurate approximation to m_k (see the condition (1) in the flow chart in Figure 1), which requires additional iteration(s). In other words, the increased computational cost due to the mentioned unutilized values in the last iteration is the price that has to be paid in order to decrease the total number of iterations in PART (I) of the algorithm presented in Figure 1.

One of the main goals of this paper is to determine the convergence speed of sequences $\mu_2(x_k)$, $\mu_3(x_k)$ and $\mu_4(x_k)$ defined by (18), (19) and (20), respectively. Since the argument of these sequences is x_{k+1}, we deal in PROGRAM 3 with `e1=` $x_{k+1} - \alpha$. Also, we use the notation `a =` α; `e =` $x - \alpha$; `mi4 =` $\mu_4(x)$. We give the program for $\mu_4(x)$, the remaining two sequences $\mu_2(x_k)$ and $\mu_3(x_k)$ can be analyzed by small modifications of PROGRAM 3.

PROGRAM 3: CONVERGENCE RATE OF METHODS FOR FINDING ORDER OF MULTIPLICITY

```
Clear["Global'*"]
f[e1] = fma/m!*e1^m(1 + C1*e1 + C2*e1^2 + C3*e1^3);
g = f[e1]; g1 = D[g, e1]; g2 = D[g1, e1]; g3 = D[g2, e1]; g4 = D[g3, e1];
u = g/g1 // Simplify; A2 = g2/(2g1) // Simplify;
A3 = g3/(6g1) // Simplify; A4 = g4/(24g1) // Simplify;
mi4 = Series[Simplify[1 - 3 A2*u + 3 A3*u^2], {e, 0, 3}]*Series[1/Simplify
      [1 - 4 A2*u + 2 u^2 (A2^2 + 2 A3) - 4 A4*u^3], {e, 0, 3}] // Simplify;
r4 = mi4 - m; Print["r4=",r4]
```

$r4 = C1e1 + (2C2 - C1^2)e1^2 + O(e1^3).$

From the expression given by r4 and (24) we observe that

$$\eta^{(4)}_{k+1} := \mu_4(x_{k+1}) - m = C_1 H_3 \varepsilon_k^3 + O(\varepsilon_k^6) = \mathcal{K}\varepsilon_k^3 + O(\varepsilon_k^6), \quad \mathcal{K} = C_1 H_3, \tag{27}$$

where H_3 is given by (23). Now we will prove that the iterative sequence $\{\mu_4(x_k)\}$ of approximations to the order of multiplicity m, defined by (19), is also cubically convergent as the root-finding method (5) that appears in the first step of Algorithms 1 and 2.

In view of (24) we have $\varepsilon_k = O(e_{k-1}^3)$, and taking into account (27) we obtain

$$\begin{aligned} M(\mu_4): &= \lim_{k\to\infty} \frac{\mu_4(x_{k+1}) - m}{(\mu_4(x_k) - m)^3} = \lim_{k\to\infty} \frac{\mathcal{K}\varepsilon_k^3 + O(e_k^6)}{\Big(\mathcal{K}\varepsilon_{k-1}^3 + O(e_{k-1}^6)\Big)^3} \\ &= \lim_{k\to\infty} \frac{\mathcal{K}\Big(\mathcal{K}\varepsilon_{k-1}^3\Big)^3 + O(e_{k-1}^{12})}{\Big(\mathcal{K}\varepsilon_{k-1}^3\Big)^3 + O(e_{k-1}^{12})\Big)} = \mathcal{K}. \end{aligned}$$

According to the last expressions and (21) and (23), the asymptotic error constant of the sequence $\{\mu_4(x_k)\}$ is equal to

$$\begin{aligned} AEC(\mu_4) &= M(\mu_4) = \mathcal{K} = C_1 H_3 = \frac{C_1(2C_2 - C_1^2)}{m} \\ &= \frac{1}{m(m+1)^2}\frac{f^{(m+1)}(\alpha)}{f^{(m)}(\alpha)}\Bigg(\frac{2}{m+2}\frac{f^{(m+2)}(\alpha)}{f^{(m)}(\alpha)} - \frac{1}{m+1}\Big(\frac{f^{(m+1)}(\alpha)}{f^{(m)}(\alpha)}\Big)^2\Bigg). \end{aligned}$$

In the same way, using a small modification of PROGRAM 3, we find

$$\eta^{(3)}_{k+1} := \mu_3(x_{k+1}) - m = C_1\varepsilon_{k+1} + (2C_2 - C_1^2)\varepsilon_{k+1}^2 + O(\varepsilon_{k+1}^3) = C_1 H_3 \varepsilon_k^3 + O(\varepsilon_k^6). \tag{28}$$

In a similar way as above we obtain

$$M(\mu_3) = \lim_{k\to\infty}\frac{\mu_3(x_{k+1}) - m}{(\mu_3(x_k) - m)^3} = C_1 H_3$$

and

$$AEC(\mu_3) = M(\mu_3) = C_1 H_3 = \frac{C_1(2C_2 - C_1^2)}{m} = AEC(\mu_4).$$

Using the same procedure and simplified PROGRAM 3 we find

$$\eta_{k+1}^{(2)} := \mu_2(x_{k+1}) - m = 2C_1\varepsilon_{k+1} + O(\varepsilon_{k+1}^3) = 2C_1H_3\varepsilon_k^3 + O(\varepsilon_k^9). \tag{29}$$

Hence

$$M(\mu_2) = \lim_{k\to\infty} \frac{\mu_2(x_{k+1}) - m}{(\mu_2(x_k) - m)^3} = 2C_1H_3$$

and

$$AEC(\mu_2) = M(\mu_2) = 2C_1H_3 = \frac{2C_1(2C_2 - C_1^2)}{m} = 2\,AEC(\mu_4).$$

According to the last results we conclude that the sequences $\{\mu_2(x_k)\}$ and $\{\mu_3(x_k)\}$ of approximations to the order of multiplicity m, defined by (18) and (19), respectively, have also the order three.

Summarizing the above consideration we can state the following assertion:

Theorem 6. *Let $\{x_k\}$ be the sequence of approximations to the zero α of multiplicity m, produced by the Halley-like method* (5)*. Then the iterative sequences $\{\mu_2(x_k)\}$* (18)*, $\{\mu_3(x_k)\}$* (19) *and $\{\mu_4(x_k)\}$* (20) *of approximations to the order of multiplicity m of the multiple zero α of a given function f are cubically convergent.*

Remark 6. The order of convergence of the sequences $\{\mu_3(x_k)\}$, $\{\mu_4(x_k)\}$ etc. does not increase since all errors $|\mu_k(x_k) - m|$ $(k \geq 2)$ are of the order of $|x_k - \alpha| = |\varepsilon_k|$. More general, the order of convergence of the sequence $\{\mu_k(x_k)\}$ $(k \geq 2)$ is equal to the order of iterative method that produces approximations x_k, used in the next step for the calculation of multiplicity. For example, using the iterative function $g_2(x_k)$ of the second order we obtain than all sequences $\{\mu_2(x_k)\}$, $\{\mu_3(x_k)\}$ etc. are of second order. In fact, the use of higher derivatives in $\{\mu_\lambda(x_k)\}$ gives approximations to the order of multiplicity with the main part which is very close to the exact multiplicity m plus additional "parasite terms" of order $O(\varepsilon^\nu)$ $(\nu > 1)$, which are negligible. A simple analysis of the proof given in Appendix also leads to the same conclusion.

5 Algorithm for finding multiple zeros of great accuracy

Now we state an efficient composite algorithm for finding multiple zero. For simplicity, we will write m_k instead of $\mu_\lambda(x_k)$ $(\lambda \geq 2)$ in what follows. The flow chart of this algorithm is shown in Figure 1 and it is consisted from the parts (I) and (II):

F i g u r e 1

Figure 1: Flow chart of Algorithm 2

(I): Starting from a sufficiently good initial approximation x_0 to the zero α of a given function f, Algorithm 1 or 2 (defined by the couple of iterative sequences (25) and (26)) is applied iteratively until the termination criterion given by the inequality $\delta := |f(x_k)|^{1/m} < \tau$ is met, where the multiplicity m is rounded to the nearest integer by the command ROUND(m_k) in the flow chart on Fig. 1. In fact, having in mind that $|x_k - \alpha| = O\big(|f(x_k)|^{1/m}\big)$, by the termination constant δ we control the wanted accuracy of the approximation x_k in the first part of algorithm putting in practice, say 10^{-5}. The improvement of this accuracy is carried out by the two-step method (30) presented in PART (II) of the algorithm. The flow chart in Figure 1 is the same for both Algorithm 1 and Algorithm 2; the approximation of order of multiplicity m_k is calculated by (18) or (19). Note that two IF criteria provides both the exact value of the multiplicity m (condition IF(1)) and sufficient accuracy of the approximation x_k (condition IF(2)).

(II): In this part we use sufficiently good approximation x_k to the zero α and the knowledge of the multiplicity m found in PART (I). To improve this approximation to a great accuracy, we apply recently stated two-point method for finding a multiple zero given by Li et al. [7]:

$$\begin{cases} y_k = x_k - \frac{2m}{m+2} \cdot u(x_k), \\ x_{k+1} = x_k - \dfrac{\frac{m(m-2)}{2}\left(\frac{m+2}{m}\right)^m z_k - \frac{m^2}{2}}{1-\left(\frac{m+2}{m}\right)^m z_k} \cdot u(x_k), \quad u(x_k) = \dfrac{f(x_k)}{f'(x_k)},\ z_k = \dfrac{f'(y_k)}{f'(x_k)}. \end{cases} \tag{30}$$

For more details on multipoint root-finders see the monograph [8].

Although the first two-point fourth order method for simple zeros, consuming only three function evaluations, was derived in 1960 by Ostrowski [27], a half of century was needed until the construction of two-point methods of fourth order for multiple zeros, see [6], [7], [10], [11] and other related papers. The only reason for this delay was the lack of symbolic computation; its application has provided that very complicated expressions can be handled. Note that the method (30) was generalized in [6] and [11].

As mentioned, the two-point method (30) possesses great computational efficiency since it has the order four requiring only three function evaluations: $f(x_k)$, $f'(x_k)$, $f'(y_k)$. Due to its very fast convergence, it is sufficient to apply only one iteration for solving most practical problems. If we wish a very accurate approximation (rarely required in practice), we can apply the method (30) iteratively, usually two or three iterations, which is emphasized by dash lines in Figure 1. The numerical examples presented in Section 7 have been executed with only one iteration of the method (30).

6 Basins of attraction of the presented methods

The improvement of computer graphics have provided a new methodology for visual study of convergence behavior of root-finding methods as a function of the various starting points. This approach is based on the notion of *basins of attraction*. Let $\alpha_1, \alpha_2, \ldots, \alpha_r \in S$ be simple or multiple zeros of a given sufficiently many times differentiable function f in some complex domain $S \subseteq \mathbb{C}$. If an iterative method is defined by $x_{k+1} = g(x_k)$, then the basin of attraction of the zero α_i is the set

$$\mathcal{B}_{f,g}(\alpha_i) = \{\zeta \in S \,|\, \textit{the iteration } x_{k+1} = g(x_k) \textit{ with } x_0 = \zeta \textit{ converges to } \alpha_i\}.$$

The basin of attraction is used to compare methods (4)–(7) by taking a square containing all the zeros and using many points (usually equally distributed) as initial points to see which zero the method converges to. In the simple case of two zeros at ± 1, one would like to have the square divided by a straight vertical line through the origin with all points to the left converge to the zero at -1 and all the point to the right converge to $+1$.

In our work we have taken 7 examples with various number of zeros and a variety of multiplicities. We will show how each of the four methods performs in each case by plotting the basins of attraction and collecting data about the number of function evaluations per point on average used by each method for each example. We also collect the CPU time in seconds required to run each method on the 360 000 equally spaces points in the 6 by 6 square centered at the origin. We allow a maximum of 40 iterations from every initial point. If the method did not converge we paint the point black.

Each basin will have a different color and the shading is darker when the number of iterations is higher.

Example 1. In the first example we have taken the polynomial

$$p_1(z) = (z^2 - 1)^2$$

with roots at ± 1 each with multiplicity 2. The basins of attraction are given in Figure 2 where the leftmost is for method g_2 and the rightmost is for g_5. It is clear that all performed very well and the boundary of the basins is a vertical straight line. This is, by the way, is not always true. There are method for multiple zeros that require the knowledge of multiplicity and will not have straight line as boundary, for example, one of Dong's methods [29]. To have a more quantitative comparison, we have collected the data in Table 1. Method g_3 uses the least number of function evaluations per point on average (15.50) and g_2 uses the most number (17.48). Based on CPU time, we have the same conclusion.

Example 2. The second example is different from the first in the fact that the multiplicity is 3,

$$p_2(z) = (z^2 - 1)^3.$$

We will not show the basins, but the conclusions are the same.

Example 3. Here we increased the multiplicity to 4,

$$p_3(z) = (z^2 - 1)^4.$$

Again the conclusions are independent of the multiplicity.

Example 4. The next example is a polynomial with the three roots of unity each with multiplicity 3, i.e.,

$$p_4(z) = (z^3 - 1)^3.$$

The basins are displayed in Figure 3. Now the boundary of the three basins is no longer a straight line. There is only one method, known to the authors, for which the boundaries are straight lines and it is Euler-Cauchy. This method requires the knowledge of the multiplicity.

The basins for each root are divided into two disjoint areas. The best method is g_5 which shows the largest basin for each root. The number of function evaluation per point is the lowest for g_3 and the highest for g_2. The fastest method is g_3 and the slowest is g_5. Both of these two methods have only one black point.

Example 5. The polynomial has four root at ± 1 and $\pm i$, each with multiplicity 4, that is,

$$p_5(z) = (z^4 - 1)^4.$$

The basins are displayed in Figure 4. Again the basin for each root is made up of several disjointed areas. The best method is again g_5. In terms of the number of function evaluations per point, we have g_4 as best and g_2 as worst. The fastest method is g_3 followed closely by g_4 and the slowest is g_2. The number of black point is the smallest for g_3 and g_4 (1201) and the largest for g_2. Based on the data we can say that g_3 is best, even though the largest contiguous basins are for g_5.

Example 6. Here we have roots of multiplicity 4 at 0 and ± 1, i.e.,

$$p_6(z) = (z^3 - z)^4.$$

The basins are displayed in Figure 5. This is a very hard problem for these methods. The only one without black points is g_5. The worst is g_2. In terms of CPU time g_4 is faster than g_5 but it uses more function evaluations than g_5.

Example 7. The last example is

$$p_7(z) = (z^5 - 1)^5.$$

The basins are displayed in Figure 6. The conclusions are similar: the method g_5 has the largest contiguous basin for each root. The method g_4 is faster than g_5 but slower than g_3. The number of black points is the smallest for g_5 followed by g_4. The largest number of black points is 8562 for g_2. This method is also the slowest and uses the highest number of function evaluations.

T a b l e 1

Table 1: CPU time, average numbers of iterations and number of function evaluations

Concluding remarks on basins of attraction: In order to decide on potentially best method overall, we have averaged the data across 7 examples. The method g_3 is the fastest (611.33 seconds) followed by g_4 (621.74 seconds) and g_5 (700.94 seconds). Method g_5 has the lowest number of black points and the largest contiguous basins. In terms of the number of function evaluations per point, g_4 is best with 18.26 followed by g_5 with 18.78. It is assumed that the above conclusions hold for the tested polynomials p_1–p_7; in general, it is hard to rank iterative methods regarding their quality, even when they possess the same order of convergence and approximately same computational efficiency.

F i g u r e 2

Figure 2: g_2 (left), g_3 (second from left), g_4 (third from left) and g_5 (right) for the zeros of the polynomial $(z^2 - 1)^2$.

F i g u r e 3

Figure 3: g_2 (left), g_3 (second from left), g_4 (third from left) and g_5 (right) for the zeros of the polynomial $(z^3 - 1)^3$.

F i g u r e 4

Figure 4: g_2 (left), g_3 (second from left), g_4 (third from left) and g_5 (right) for the zeros of the polynomial $(z^4 - 1)^4$.

F i g u r e 5

Figure 5: g_2 (left), g_3 (second from left), g_4 (third from left) and g_5 (right) for the zeros of the polynomial $(z^3 - z)^4$.

F i g u r e 6

Figure 6: g_2 (left), g_3 (second from left), g_4 (third from left) and g_5 (right) for the zeros of the polynomial $(z^5 - 1)^5$.

7 Numerical examples

In this section we present results obtained by the combination of Algorithm 1/Algorithm 2 and the two-point method (30) to six examples. All computation were performed by CAS *Mathematica* using multi-precision arithmetic. We choose the termination constant $\delta = 10^{-5}$; it turned out that three iterations were sufficient to meet the termination criterion for all six examples. The outcome approximation x_k and the exact order of multiplicity m serve as the initial values for one iteration of the two-point method (30) in order to improve the accuracy of the wanted multiple zero. We used the six test functions, including $f_2(x)$ taken from [5] and f_4 from [28].

$f(x)$	m	x_0	α
$f_1(x) = \left(e^{x^2+6x-16} - 1\right)^2\left((x-1)^3 - 1\right)^2$	4	1.7	2
$f_2(x) = \left(xe^{x^2} - \sin^2 x + 3\cos x + 5\right)^4$	4	-0.7	$-1.207647827130918927009\ldots$
$f_3(x) = \left(x\sin x - 2\sin^2(x/\sqrt{2})\right)\left(x^8 + x^4 + 100\right)$	6^*	-1.2	0^*
$f_4(x) = x^{10} - 20x^9 + 175x^8 - 882x^7 + 2835x^6 - 6072x^5$ $+8777x^4 - 8458x^3 + 5204x^2 - 1848x + 288$	3^*	-40	2^*
$f_5(x) = \big(-177147 + 649539x - 1082565x^2 + 1082565x^3$ $-721710x^4 + 336798x^5 - 112266x^6 + 26730x^7$ $-4455x^8 + 495x^9 - 33x^{10} + x^{11}\big)^{1/4}$ $\times(x+1)^4(x^2+2x+2)^2$	$11/4^*$	2	3^*
$f_6(x) = \left(e^{x^2+4x+8} - 1\right)^3\left(\sinh(2+2i+ix)\right)^2$	5	$-1.6+1.7i$	$-2+2i$

List of tested functions

T a b l e 2

Table 2: Errors of approximation for $f_1(x)$

T a b l e 3

Table 3: Errors of approximation for $f_2(x)$

T a b l e 4

Table 4: Errors of approximation for $f_3(x)$

T a b l e 5

Table 5: Errors of approximation for $f_4(x)$

T a b l e 6

Table 6: Errors of approximation for $f_5(x)$

Table 7

Table 7: Errors of approximation for $f_6(x)$

Tables 2–7 show the errors $|g_3(x_k) - \alpha|$, $|\mu_2(x_k) - m|$ and $|\mu_3(x_k) - m|$ for $k = 1, 2, 3$ while $|x_4 - \alpha|$ is the error of approximation obtained by the method (30). The notation $A(-\tau)$ means $A \times 10^{-\tau}$. The meaning of * is explained in Remark 7. The multiplicity of the sought zero of the function f_5 is a fraction so that we have slightly modified the program using *Mathematica* statement `Rationalize[z,10^(-4)]` (instead of `Round[z]`) that provides the representation of a real number z in the form of a fraction a/b $(a, b \in \mathbb{N})$. We assume that such fraction exists, otherwise, the program does not work. Fortunately, order of multiplicity unrepresentable as a fraction appears in artificially constructed functions, not in practical problems. The methods works for complex zeros too, as the example for f_6 shows. In this case we assume that the multiplicity is a positive integer and use the command `Re`$[\mu_\lambda(x)]$ $(\lambda = 2, 3)$ to eliminate imaginary part since the calculation of μ_λ is performed with complex approximations to the complex zero.

Remark 7. From the expressions for the functions f_1, f_2 and f_6 it is clear that the multiplicity is $m = 4, 4, 5$, respectively. However, it is hard to assume that f_3 has a zero $(\alpha = 0)$ of multiplicity 6. Besides, for the polynomial $f_4(x)$ given above in the expanded form, it is not possible to detect zeros and their multiplicities without a checking procedure. In fact, the factorized form of f_4 reads

$$f_4(x) = (x-1)^4(x-2)^3(x-3)^2(x-4),$$

whence we can observe the values of zeros and multiplicities. Furthermore, since the initial approximation $x_0 = -40$ (taken from [30]) is rather far from actual zeros, the determination of the zero and its multiplicity was left to Algorithm 1/Algorithm 2, which have perfectly done this job. The same story is valid for the zero of f_5 whose multiplicity is a fraction. With regard to the described facts Tables 3–5 were formed *a posteriori*, which is marked with * in the list of test functions.

From Tables 2–7 we observe that the presented algorithms (25) and (26) successfully detect the order of multiplicity, which is the necessary condition for the application of very fast method (30) working with known multiplicity of the zero. The application of simpler formula $\mu_2(x_k)$ (Algorithm 1) gives satisfactory results, which is clear having in mind that $\mu_2(x_k)$ and $\mu_3(x_k)$ are both cubically convergent. For less computational cost it is recommendable to apply Algorithm 1.

8 Conclusions

The main goal of this paper is to present the application of symbolic computation in solving mathematical problems which belong to the group of problems unsolvable by hand derivation and manipulation due to very lengthy and complicated expressions. In particular, we have demonstrated the use of symbolic computation (i) for generating higher order iterative methods for finding multiple zeros of nonlinear equations with unknown multiplicity of zeros and (ii) for constructing coupled algorithms that calculate simultaneously approximations to the sought zero and the exact order of multiplicity during the iterative process. In our concrete case symbolic computation was used basically for automatic differentiation and simplification of very complicated expressions, but also for other manipulations with mathematical objects such as expanding out products, finding limit values and collecting together terms involving the same powers of objects matching some variables. These operations with mathematical symbols are not possible by classical methods, which points out that symbolic computation is a powerful modern tool that enables developing new methods and ideas, their testing, checking derived results and analysis. Its application is supported by computer algebra systems, in our case by Wolfram's *Mathematica*.

Another methodology used in this paper is dynamic study of iterative methods based on basins of attraction. Namely, in many situations numerical experiments and the study of computational efficiency of considered iterative methods do not give sufficiently good measures/data for a real estimation of the quality of these methods and their ranking. To overcome this flaw to a certain extent, basins of attraction are considered in Section 6 in order to offer additional quantitative iteration data as well as a visual convergence behavior of iterative methods depending on areas of starting points. This approach provide considerably better understanding of iterative processes although numerous open questions leave to be answered, see the book [32].

Appendix

Although two iterative sequences in (25) and (26) are independent, we can consider them as a couple of sequences $\{\varepsilon_i^{(k)}\}$ and $\{\eta_i^{(k)}\}$ and determine their convergence order using the following very useful assertion [31]:

Theorem A. *Given the error-recursion*

$$v_i^{(k+1)} \le a_i \prod_{j=1}^{q} \big(v_j^{(k)}\big)^{t_{ij}}, \quad (i \in \mathbb{I}_q := \{1,\dots,q\};\ k \ge 0), \tag{A.1}$$

where $t_{ij} \ge 0,\ a_i > 0,\ 1 \le i,j \le q$, and $v_i^{(k)}$ are some convergent sequences. Denote the matrix of exponents appearing in (A.1) *with T_q, that is $T_q = [t_{ij}]_{q\times q}$. If the non-negative matrix T_q has the spectral radius $\rho(T_q) > 1$ and the corresponding eigenvector $\boldsymbol{x}_\rho > 0$, then all sequences $\{v_i^{(k)}\}$ $(i \in \mathbb{I}_q)$ have the R-order at least $\rho(T_q)$.*

Let $O_R(IM)$ denote the R-order of convergence of an iterative method IM, and let

$$v_1^{(k)} = \varepsilon_k = x_k - \alpha, \quad v_2^{(k)} = \eta_k = m_k - m, \quad \text{where} \ \ m_k = \mu_\lambda(x_k)\ (\lambda = 2,3,4).$$

According to Theorem 4 (see relation (21)) we can write for the method (5)

$$\varepsilon_{k+1} \sim \varepsilon_k^3, \tag{A.2}$$

where the notation $a \sim b$ means $a = O(b)$. Further, from (27), (28) and (29) we have

$$\eta_{k+1}^{(\lambda)} \sim \varepsilon_k^3, \quad (\lambda = 2,3,4). \tag{A.3}$$

According to the relations (A.2) and (A.3) we form the matrix $T_2 = [t_{ij}]$ that appears in Theorem A:

$$T_2 = \begin{bmatrix} 3 & 0 \\ 3 & 0 \end{bmatrix}.$$

The spectral radius of T_2 is $\rho(T_2) = 3$ and the corresponding eigenvector is $\boldsymbol{x}_\rho = (1,1) > 0$. Hence, according to Theorem A, we obtain the lower bound of the R-order of the coupled sequences $(\varepsilon_k, \eta_k^{(\lambda)})$

$$O_R(g_3, \mu_\lambda) \ge \rho\,(T_2) = 3.$$

Since the limits

$$|H_3| = \lim_{k\to\infty} \frac{|g_3(x_k) - \alpha|}{|x_k - \alpha|^3} \quad \text{and} \quad |M(\mu_\lambda)| = \lim_{k\to\infty} \frac{|m_{k+1} - \alpha|}{|m_k - \alpha|^3}$$

exist, $0 < |H_3| < \infty$ and $0 < |M(\mu_\lambda)| < \infty$, according to Ortega and Rheinboldt [25, E 9.3-4] it follows that the R-order and Traub's C-order are equal. Therefore, the methods defined by iteration functions $g_3(x)$ and $\mu_\lambda(x)$ $(\lambda = 2,3,4)$ have cubic convergence.

Acknowledgement. The authors wish to thank anonymous reviewers for careful reading and valuable comments which improved the quality of the paper. This work is supported by the Serbian Ministry of Education and Science.

References

[1] J. Borwein, D. Bailey, Mathematics by Experiments, A K Peters, Ltd., Wellesley, MA, 2008

[2] S.M. Watt, Making Computer algebra more symbolic, Proc. Transgressive Computing 2006, pp. 43–49.

[3] J.S. Cohen, Computer Algebra and Symbolic Computation: Mathematical Methods, AK Peters Ltd., 2003.

[4] D.H. Beiley, J.M. Borwein, N.J. Calkin, R. Girgensohn, D.R. Luke, V.H. Moll, Experimental Mathematics in Action, A K Peters, Ltd., Wellesley, MA, 2007.

[5] C. Chun, B. Neta, A third-order modification of Newton's method for multiple roots, Appl. Math. Comput. 211 (2009) 474-479.

[6] S. Li, L. Cheng, B. Neta, Some fourth-order nonlinear solvers with closed formulae for multiple roots, Comp. Math. Appls 59 (2010) 126–135.

[7] S. Li, X. Liao, L. Cheng, A new fourth-order iterative method for finding multiple roots of nonlinear equations, Appl. Math. Comput. 215 (2009) 1288–1292.

[8] M.S. Petković, B. Neta, L.D. Petković, J. Džunić, Multipoint Methods for Solving Nonlinear Equations, Elsevier, 2013.

[9] B. Neta, A.N. Johnson, High-order nonlinear solver for multiple roots, Comp. Math. Appls 55 (2008) 2012–2017.

[10] J.R. Sharma, R. Sharma, Modified Jarratt method for computing multiple roots, Appl. Math. Comut. 217 (2010) 878–881.

[11] X. Zhou, X. Chen, Y. Song, Construction of higher order methods for multiple roots of nonlinear equations, J. Comput. Appl. Math. 235 (2011) 4199–4206.

[12] P. Kravanja, M. Van Barel, Computing the Zeros of Analytic Functions, Lecture Notes in Mathematics 1727, Springer, Berlin, 2000.

[13] C. Chun, H.J. Bae, B. Neta, New families of nonlinear third-order solvers for finding multiple roots, Comput. Math. Appls 57 (2009) 1574–1582.

[14] B.I. Yun, A non-iterative method for solving nonlinear equations, Appl. Math. Comput. 198 (2008) 691–699.

[15] B.I. Yun, Transformation methods for finding multiple roots of nonlinear equations, Appl. Math. Comput. 217 (2010) 599–606.

[16] G. Collins, Continued fraction real root isolation using the Hong root bound, J. Symb. Comput. 72 (2016).

[17] A. Herman, H. Hong, Quality of positive root bounds, J. Symb. Comput. 74 (2016) 592–602.

[18] A. Iliev, N. Kyurkchiev, Nontrivial Methods in Numerical Analysis: Selected Topics in Numerical Analysis, LAP LAMBERT Academic Publishing, Saarbrücken, 2010.

[19] J.F. Traub, Iterative Methods for the Solution of Equations, Prentice Hall, New York, 1964.

[20] E. Schröder, Über unendlich viele Algorithmen zur Auflösung der Gleichungen, Math. Ann. 2 (1870) 317–365.

[21] M.S. Petković, L.D. Petković, J. Džunić, Accelerating generators of iterative methods for finding multiple roots of nonlinear equations, Comput. Math. Appls 59 (2010) 2784–2793.

[22] B. Jovanović, A method for obtaining iterative formulas of higher order, Mat. Vesnik 9(24) (1972) 365–369.

[23] M. Petković, D. Herceg, On rediscovered iteration methods for solving equations, J. Comp. Appl. Math. 107 (1999) 275–284.

[24] Wolfram Research, Inc., Mathematica, Version 9.0, Champaign, IL, 2012.

[25] J.M. Ortega, W.C. Rheinboldt, Iterative Solution of Nonlinear Equations in several Variables, Academic Press, New York, 1970.

[26] J.L. Lagouanelle, Sur une métode de calcul de l'ordre de multiplicité des zéros d'un polynôme, C. R. Acad. Sci. Paris Sér. A 262 (1966) 626–627.

[27] A.M. Ostrowski, Solution of Equations and Systems of Equations, Academic Press, New York, 1960.

[28] M.R. Farmer, G. Loizou, An algorithm for the total, or partial, factorization of a polynomial, Math. Proc. Cambridge Philos. Soc. 82 (1977) 427–437.

[29] C. Dong, A family of multipoint iterative functions for finding multiple roots of equations, Int. J. Comput. Math. 21 (1987) 363–367.

[30] N. Osada, Chebyshev-Halley methods for analytic functions, J. Comp. Appl. Math. 216 (2008) 585–599.

[31] J. Herzberger, L. Metzner, On the Q-order and R-order of convergence for coupled sequences arising in iterative numerical processes. In: Numerical Methods and Error Bounds (eds G. Alefeld, J. Herzberger), Akademie Verlag, Berlin, 1996.

[32] B. Kalantari, Polynomial Root-Finding and Polynomiography, World Scientific, New Jersey, 2009.

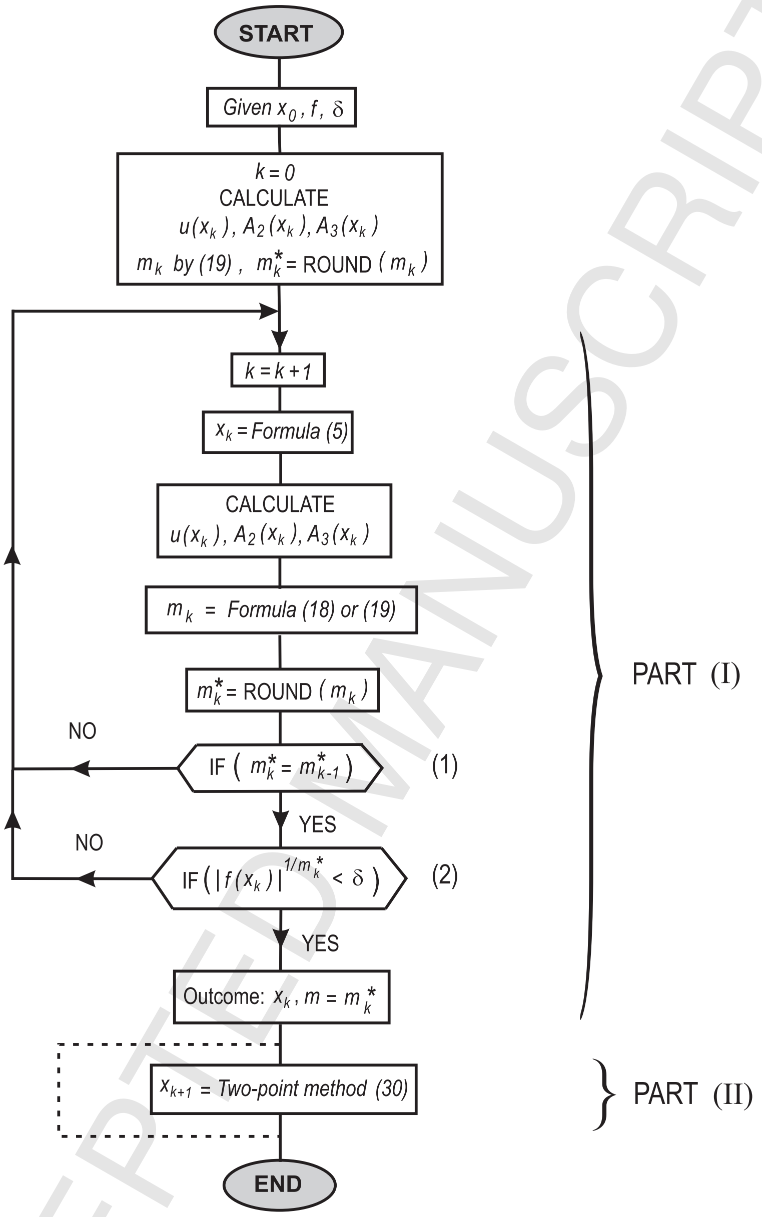
START
Given x_0, f, δ
$k = 0$
CALCULATE
$u(x_k), A_2(x_k), A_3(x_k)$
m_k by (19), $m_k^* =$ ROUND (m_k)
$k = k+1$
$x_k =$ Formula (5)
CALCULATE
$u(x_k), A_2(x_k), A_3(x_k)$
m_k = Formula (18) or (19)
$m_k^* =$ ROUND (m_k)
NO
IF ($m_k^* = m_{k-1}^*$)
(1)
YES
NO
IF ($|f(x_k)|^{1/m_k^*} < \delta$)
(2)
YES
Outcome: x_k, $m = m_k^*$
x_{k+1} = Two-point method (30)
END
PART (I)
PART (II)

Figure 1: Flow chart of Algorithm 2

Examples	Methods	A	B	C	D	E
Example 1 $p_1(z) = (z^2-1)^2$	g_2	251.83	5.83	3	17.48	601
	g_3	247.85	3.88	4	15.50	601
	g_4	285.42	3.20	5	15.98	601
	g_5	347.55	2.84	6	17.04	601
Example 2 $p_2(z) = (z^2-1)^3$	g_2	321.5	5.83	3	17.48	601
	g_3	329.91	3.88	4	15.50	601
	g_4	335.22	3.20	5	15.98	601
	g_5	387.04	2.84	6	17.04	601
Example 3 $p_3(z) = (z^2-1)^4$	g_2	345.95	5.83	3	17.48	601
	g_3	356.45	3.88	4	15.50	601
	g_4	433.42	3.20	5	15.98	601
	g_5	470.9	2.84	6	17.04	601
Example 4 $p_4(z) = (z^3-1)^3$	g_2	581.37	8.24	3	24.72	25
	g_3	484.43	4.20	4	16.81	1
	g_4	529.12	3.42	5	17.12	64
	g_5	588.62	3.00	6	18.02	1
Example 5 $p_5(z) = (z^4-1)^4$	g_2	1030.5	11.82	3	35.46	2545
	g_3	708.4	4.88	4	19.53	1201
	g_4	769.38	3.82	5	19.11	1201
	g_5	915.27	3.50	6	20.99	1329
Example 6 $p_6(z) = (z^3-z)^4$	g_2	1097.73	15.08	3	45.24	88546
	g_3	1142.18	7.51	4	30.04	23008
	g_4	953.63	4.64	5	23.22	3856
	g_5	1648.18	16.19	3	48.56	8562
Example 7 $p_7(z) = (z^5-1)^5$	g_2	1648.18	16.19	3	48.56	8562
	g_3	1010.11	5.80	4	23.18	9
	g_4	1045.97	4.09	5	20.46	1
	g_5	1187.59	3.42	6	20.54	2
Average over all examples	g_2	753.87	9.83	3	29.49	14497.29
	g_3	611.33	4.86	4	19.44	3717.43
	g_4	621.74	3.65	5	18.26	989.29
	g_5	700.94	3.13	6	18.76	477.86

A – CPU time in second; B – Average number iterations per point; C – Number of function evaluations per step; D – Average number of function evaluations per point; E – Number of points required 40 iterations

Table 1: Iteration data for the methods $g_2 - g_5$ and Examples 1–7

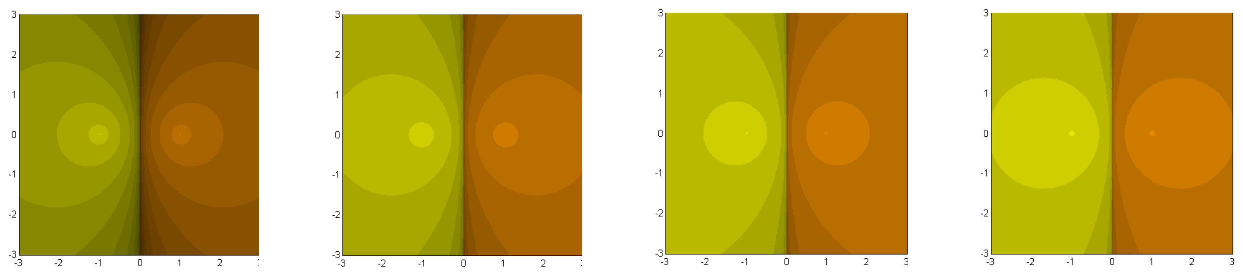

Figure 2: g_2 (left), g_3 second from left), g_4 (third from left) and g_5 (right) for the roots of the polynomial $(z^2 - 1)^2$.

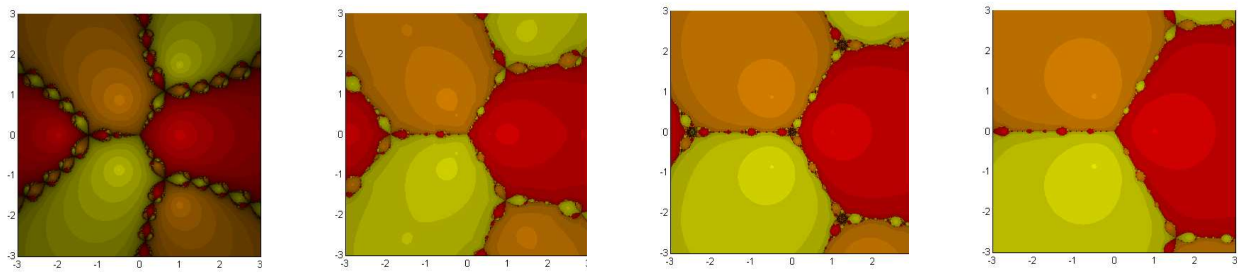

Figure 3: g_2 (left), g_3 second from left), g_4 (third from left) and g_5 (right) for the roots of the polynomial $(z^3 - 1)^3$.

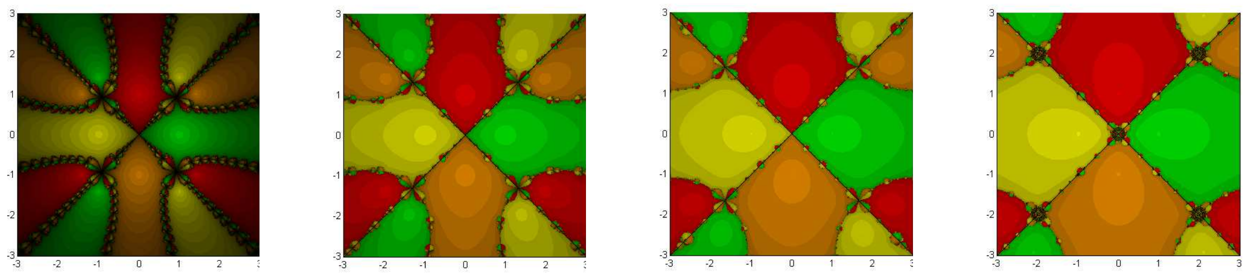

Figure 4: g_2 (left), g_3 second from left), g_4 (third from left) and g_5 (right) for the roots of the polynomial $(z^4 - 1)^4$.

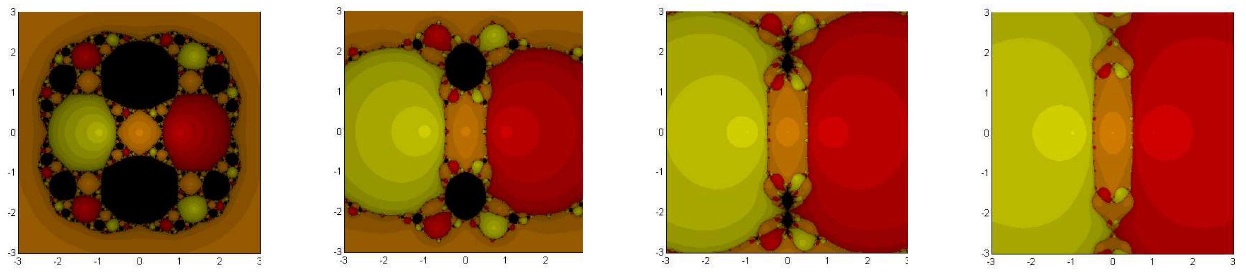

Figure 5: g_2 (left), g_3 second from left), g_4 (third from left) and g_5 (right) for the roots of the polynomial $(z^3 - z)^4$.

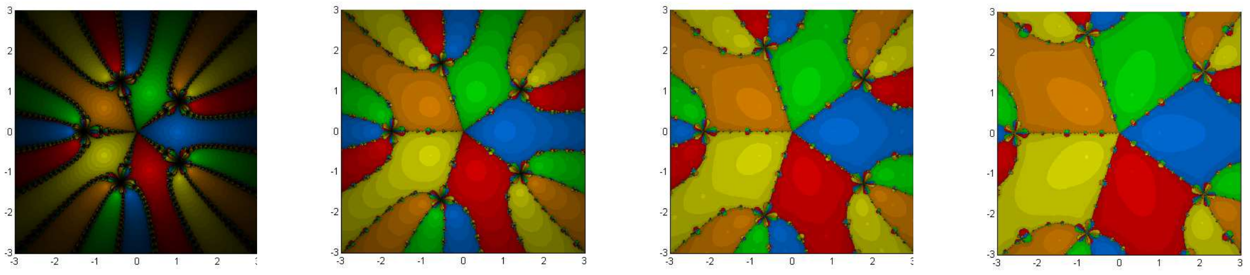

Figure 6: g_2 (left), g_3 second from left), g_4 (third from left) and g_5 (right) for the roots of the polynomial $(z^5 - 1)^5$.

	$k = 1$	$k = 2$	$k = 3$
$\|g_3(x_k) - \alpha\|$	$4.84(-2)$	$4.79(-4)$	$4.93(-10)$
$\|\mu_2(x_k) - m\|$	0.992	$1.17(-2)$	$1.20(-8)$
$\|\mu_3(x_k) - m\|$	0.583	$5.84(-3)$	$6.02(-9)$
(30) $\|x_4 - \alpha\|$	$1.94(-36)$		

Table 2: Errors of approximation for $f_1(x)$

	$k = 1$	$k = 2$	$k = 3$
$\|g_3(x_k) - \alpha\|$	$3.85(-2)$	$9.22(-5)$	$1.37(-12)$
$\|\mu_2(x_k) - m\|$	0.421	$1.11(-3)$	$1.65(-11)$
$\|\mu_3(x_k) - m\|$	0.23	$5.54(-4)$	$8.24(-12)$
(30) $\|x_4 - \alpha\|$	$1.05(-47)$		

Table 3: Errors of approximation for $f_2(x)$

	$k = 1$	$k = 2$	$k = 3$
$\|g_3(x_k) - \alpha\|$	0.798	$4.56(-2)$	$2.25(-6)$
$\|\mu_2(x_k) - m\|$	$0.78(-2)$	$8.91(-4)$	$2.17(-12)$
$\|\mu_3(x_k) - m\|$	0.402	$8.49(-7)$	$5.05(-24)$
(30) $\|x_4 - \alpha\|$	$1.33(-31)$		

Table 4: Errors of approximation for $f_3(x)$

	$k=1$	$k=2$	$k=3$
$\|g_3(x_k)-\alpha\|$	$6.94(-2)$	$7.62(-4)$	$9.21(-10)$
$\|\mu_2(x_k)-m\|$	0.293	$2.27(-3)$	$2.76(-9)$
$\|\mu_3(x_k)-m\|$	0.103	$1.14(-3)$	$1.16(-9)$
(30) $\|x_4-\alpha\|$	$9.07(-37)$		

Table 5: Errors of approximation for $f_4(x)$

	$k=1$	$k=2$	$k=3$
$\|g_3(x_k)-\alpha\|$	0.389	$7.57(-3)$	$7.26(-8)$
$\|\mu_2(x_k)-m\|$	1.47	$2.94(-2)$	$2.82(-7)$
$\|\mu_3(x_k)-m\|$	0.759	$1.47(-2)$	$1.41(-8)$
(30) $\|x_4-\alpha\|$	$1.69(-29)$		

Table 6: Errors of approximation for $f_5(x)$

	$k=1$	$k=2$	$k=3$
$\|g_3(x_k)-\alpha\|$	0.0982	$1.92(-4)$	$2.08(-12)$
$\|\mu_2(x_k)-m\|$	0.809	$2.29(-4)$	$7.37(-12)$
$\|\mu_3(x_k)-m\|$	0.391	$1.15(-4)$	$3.69(-12)$
(30) $\|x_4-\alpha\|$	$3.07(-47)$		

Table 7: Errors of approximation for $f_6(x)$