Alexandria Engineering Journal (2013) 52, 531–541

Alexandria University

Alexandria Engineering Journal

www.elsevier.com/locate/aej
www.sciencedirect.com

ORIGINAL ARTICLE

Computing multiple zeros using a class of quartically convergent methods

F. Soleymani [a,*], D.K.R. Babajee [b]

[a] Department of Mathematics, Mashhad Branch, Islamic Azad University, Mashhad, Iran
[b] Scientific & Academic Research Council, African Network for Policy Research & Advocacy for Sustainability, Mauritius

Received 6 December 2012; revised 30 April 2013; accepted 5 May 2013
Available online 2 June 2013

KEYWORDS
Multiplicity
Two-step methods
Mathematica
All the real solutions
Finitely many zeros

Abstract Targeting a new multiple zero finder, in this paper, we suggest an efficient two-point class of methods, when the multiplicity of the root is known. The theoretical aspects are investigated and show that each member of the contributed class achieves fourth-order convergence by using three functional evaluations per full cycle. We also employ numerical examples to evaluate the accuracy of the proposed methods by comparison with other existing methods.

For functions with finitely many real roots in an interval, relatively little literature is known, while in applications, the users wish to find all the real zeros at the same time. Hence, the second aim of this paper will be presented by designing a fourth-order algorithm, based on the developed methods, to find all the real solutions of a nonlinear equation in an interval using the programming package MATHEMATICA 8.

1. Preliminaries

Many methods have been developed for solving nonlinear equations or systems using different methodology, see [2] and the references therein. On the other hand, solutions themselves, can be divided into the simple or the multiple cases. That is to say, a function might have finitely many zeros in an interval which some of them are simple while the other ones could be multiple.

A multiple zero is a root with multiplicity $m \geqslant 2$, also called a multiple point or repeated root. Clearly, working and developing on multiple roots of a nonlinear equation is not an easy task in numerical analysis. Herein, we develop iterative methods to find the multiple root x^* with multiplicity m of a nonlinear equation $f(x) = 0$, i.e., $f^{(i)}(x^*) = 0$, $i = 0, 1, \cdots, m-1$, and $f^{(m)}(x^*) \neq 0$. We will also discuss on finding simple zeros and also multiple zeros when the multiplicity of the roots are unknown.

When searching for multiple roots, there are some problems, which need special attention. The first one is that there is a neighborhood of x^*, here called the error ball, where the accurate computation of $f(x)$ is not possible because of

* Corresponding author. Tel.: +98 9151401695.
E-mail address: fazl_soley_bsb@yahoo.com (F. Soleymani).
Peer review under responsibility of Faculty of Engineering, Alexandria University.

Production and hosting by Elsevier

1110-0168 © 2013 Production and hosting by Elsevier B.V. on behalf of Faculty of Engineering, Alexandria University.
http://dx.doi.org/10.1016/j.aej.2013.05.001

computational (cancelation) errors, actually the errors are bigger than function values.

As a result, an in-appropriate iterative method returns an entirely erroneous root estimate such that it may break down. Even if having some methods to recognize error ball, we can only hope a result of lower accuracy [5]. A less severe-problem is slow convergence. Thus, it is more important if we can devise methods that ensure computation of the multiple roots at high precision.

Along with the main above questions, the most important problem will be remained in implementation. In fact, in applications, one might to find all the (real) critical points of a nonlinear function in a given interval, while such iterative methods are sensitive upon the initial points!

All of such needs and questions will be answered and solved in the following sections. It is assumed in the following that the derivatives of the function exist and are easily computable.

The remainder sections of this paper unfold the contents in what follows. In Section 2, a brief discussion on the existing methods in the literature will be given. In order to contribute, we first in Section 3 present a two-step method for finding simple zeros of nonlinear equations. Theoretical aspects support the quartical convergence. Then, we extend the scheme for approximating multiple zeros when the multiplicity of the zero is known. Section 4 dedicates to remind some of the ways of approximating the order of multiplicity. Next, in Section 5, we extend one of the methods from the suggested class of Section 3 to find multiple zeros when the multiplicity of the zeros is unknown. In Section 6, some tests will be given to show the numerical behavior of the attained iterative methods for finding multiple roots by comparison with the existing methods. Since all the obtained iteration functions up to this point, are locally fourth-order convergent, and in applications we need to have the convergence to be guaranteed alongside finding all the real zeros in an interval, in Section 7, we design an efficient *hybrid* algorithm to capture all the real solutions by applying our high-order optimal methods using the efficient programming package MATHEMATICA. Finally, Section 8 will be drawn the conclusion of this study with some outlines for future works.

2. Brief review

It is well-known that when the multiplicity m of the root is given then the modified Newton's method converges quadratically and can be defined in the following form $x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$. According to the definition of efficiency index (defined in [27]), it has $2^{1/2} \approx 1.414$ as its index of efficiency. In fact, we considered that all function and derivative evaluations per computing step have the same computational cost. In order to provide better orders and efficiencies many developments have been given to the literature, see e.g. [22,26].

A third-order Chebyshev's method [27] for finding multiple roots is given by

$$x_{n+1} = x_n - \frac{m(3-m)}{2}\frac{f(x_n)}{f'(x_n)} - \frac{m^2}{2}\frac{f^2(x_n)f''(x_n)}{f'^3(x_n)}. \tag{1}$$

A note on this scheme is that it needs the computation of the second derivative along the knowledge of multiplicity to be implemented, which is costly in many problems. Scheme (2) has the efficiency index 1.442.

Now, we recall an important finding in this topic regarding the optimal relation between the number of evaluations and the local order of convergence. The upper bound for order of multi-step methods was discussed in [12] by Kung and Traub, who conjectured that the order of convergence of any multipoint method without memory, consuming $n + 1$, (functional) evaluations per iteration, cannot exceed the bound 2^n (called optimal order). This hypothesis has not been proved yet but it turned out that all existing methods constructed at present support it. Another interesting point is that this conjecture is valid for iterative methods, which are designed for simple zeros; or for multiple zeros with the known multiplicity. That is to say, if the multiplicity be unknown, then the conjecture is not anymore supported.

In case of multiple root solvers which are optimal, we can name the following schemes. Sharifi et al. in [18] proposed the following optimal fourth-order method

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n + \left(\frac{1}{4}m(m^2+2m-4)u_n - \frac{1}{4}m(m+2)^2p^m v_n\right) \\ \times\left(1 + \frac{m^4}{8(m+2)p^{2m}}(w_n)^2 - \frac{69}{64}(w_n)^3 + v_n^4\right). \end{cases} \tag{2}$$

wherein $u_n = \frac{f(x_n)}{f'(x_n)}$, $v_n = \frac{f(x_n)}{f'(y_n)}$, $w_n = \frac{f'(y_n)}{f'(x_n)} - p^{m-1}$ and $p = \frac{m}{m+2}$.

Sharma and Sharma in [19] suggested the following quartically technique

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{m}{8}\left((m^3-4m+8) - (m+2)^2\left(\frac{m}{m+2}\right)^m\frac{f'(x_n)}{f'(y_n)}(2(m-1)\right. \\ \left.-(m+2)\left(\frac{m}{m+2}\right)^m\frac{f'(x_n)}{f'(y_n)}\right)\frac{f(x_n)}{f'(x_n)}, \end{cases} \tag{3}$$

and Zhou et al. in [30] presented the following scheme with the same order as (2), (3)

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{m}{8}\left((m^3+6m^2+8m+8) + m^3\left(\frac{m+2}{m}\right)^{2m}\left(\frac{f'(y_n)}{f'(x_n)}\right)^2\right. \\ \left.-2m^2(m+3)\left(\frac{m+2}{m}\right)^m\frac{f'(y_n)}{f'(x_n)}\right)\frac{f(x_n)}{f'(x_n)}. \end{cases} \tag{4}$$

The techniques (2)–(4) approximate the multiple roots, when the multiplicity of the root is available by consuming three (functional) evaluations. They also possess the optimal order four and the optimal efficiency index 1.587 in the sense of Kung–Traub.

For further related developments and applications, see [16,20,21,23–25], where some other aspects of nonlinear equation solving by iterative methods have been discussed.

Inspired by these new optimal developments and also by the use of weight function, we present in the next section, a general technique for solving nonlinear scalar equations.

3. Construction of the new technique

In this section, we derive a new technique of two-point methods of order four, requiring three (functional) evaluations per

iteration. This means that the proposed technique supports the Kung–Traub conjecture. Besides, this technique will be extended for finding multiple roots of nonlinear equations, when the multiplicity of the root is available.

Let us consider third-order Halley-like methods with one f and two f' evaluations as comes next

$$\begin{cases} y_n = x_n - \beta\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{2\beta f(x_n)}{(2\beta-1)f'(x_n)+f'(y_n)}, \ \beta\neq 0. \end{cases} \tag{5}$$

The member $\beta = \frac{2}{3}$ was recently rediscovered in [1] using the Closed-Open quadrature formula and is given by

$$\begin{cases} y_n = x_n - \frac{2}{3}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{4f(x_n)}{f'(x_n)+3f'(y_n)}. \end{cases} \tag{6}$$

We obtain a new class of fourth-order methods from this member. We build our class according to (6) without any additional evaluations of the function or its derivatives.

Let us consider the following class of methods by the use of weight function:

$$\begin{cases} y_n = x_n - \frac{2}{3}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{4f(x_n)}{f'(x_n)+3f'(y_n)}\ [H(\tau_n)], \end{cases} \tag{7}$$

wherein $\tau_n = \frac{f'(y_n)}{f'(x_n)}$. We prove that the class (7) is of local fourth-order for simple roots.

Theorem 1. *Let a sufficiently smooth function $f: D \subset R \to R$ has a simple root x^* in the open interval D. Then, the class of methods without memory (7) is of local fourth-order convergence, when $H(1) = 1,\ H'(1) = 0,\ H''(1) = \frac{9}{8}, |H^{(3)}(1)| < \infty$ and it satisfies the error equation*

$$e_{n+1} = \left(\left[3 + \frac{32}{81}H^{(3)}(1)\right]c_2^3 - c_2c_3 + \frac{c_4}{9}\right)e_n^4 + O\left(e_n^5\right). \tag{8}$$

Proof. Using Taylor series and symbolic computations in the programming package MATHEMATICA 8, we can determine the asymptotic error constant of the two-step class (7). We now assume that $c_k = f^{(k)}(x^*)/(k!f'(x^*))$, $k \geqslant 2$ and $e_n = x_n - x^*$. Therefore $f(x_n) = f'(x^*)\left[e_n + c_2e_n^2 + c_3e_n^3 + c_4e_n^4 + O\left(e_n^5\right)\right]$, and $f'(x_n) = f'(x^*)\left[1 + 2c_2e_n + 3c_3e_n^2 + 4c_4e_n^3 + O\left(e_n^4\right)\right]$. Dividing these two relations on one another gives us

$$\frac{f(x_n)}{f'(x_n)} = e_n - c_2e_n^2 + 2\left(c_2^2 - c_3\right)e_n^3 + \left(7c_2c_3 - 4c_2^3 - 3c_4\right)e_n^4 + O\left(e_n^5\right). \tag{9}$$

Now we have $y_n = x^* + \frac{e_n}{3} + \frac{2c_2e_n^2}{3} - \frac{4}{3}\left(c_2^2 - c_3\right)e_n^3 + O\left(e_n^4\right)$. Furthermore, using again the Taylor expansion, we will have $\frac{4f(x)}{f'(x)+3f'(y)} = e_n - c_2^2e_n^3 + \left(3c_2^3 - 3c_2c_3 - \frac{c_4}{9}\right)e_n^4 + O\left(e_n^5\right)$, and also

$$x_n - \frac{4f(x_n)}{f'(x_n)+3f'(y_n)} - x^* = c_2^2e_n^3 + \left(-3c_2^3 + 3c_2c_3 + c_4/9\right)e_n^4 + O\left(e_n^5\right). \tag{10}$$

We now by using (10) in (7) and considering the weight function, obtain that $x_{n+1} - x^* = (1 - H(1))e_n + \frac{4}{3}c_2H'(1)e_n^2 + \left(\frac{8}{3}c_3H'(1) + c_2^2(H(1) - 4H'(1) - \frac{8H''(1)}{9})\right)e_n^3 + \frac{1}{81}(c_4(9H(1) + 312H'(1) + 9c_2c_3(27H(1) - 8(15H'(1) + 4H''(1))) + c_2^3(-243H(1) + 756H'(1) + 432H''(1) + 32H^{(3)}(1)))e_n^4 + O\left(e_n^5\right)$. This completely reveals that the weight function in (7) must satisfy $H(1) = 1,\ H'(1) = 0,\ H''(1) = \frac{9}{8},\ |H^{(3)}(1)| < \infty$ to let the local order arrives at four. This completes the proof and shows the contributed class (7) arrives at quartical local order of convergence using two evaluations of the first-order derivative and one evaluation of the function, and read the error Eq. (8). □

A simple choice for $H(\tau_n)$ satisfying Theorem 1, is $H(\tau_n) = 1 + \frac{9}{16}(\tau_n - 1)^2$, so that we get the following fourth-order method:

$$\begin{cases} y_n = x_n - \frac{2}{3}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{4f(x_n)}{f'(x_n)+3f'(y_n)}\left[1 + \frac{9}{16}\left(\frac{f'(y_n)}{f'(x_n)} - 1\right)^2\right], \end{cases} \tag{11}$$

with error equation

$$e_{n+1} = \left(3c_2^3 - c_2c_3 + \frac{c_4}{9}\right)e_n^4 + O\left(e_n^5\right). \tag{12}$$

Remark 1. It could be shown that some of the existing methods are the special cases of (7). For example, rearrange the term from (7) as follows

$$\frac{4f(x_n)}{f'(x_n)+3f'(y_n)}\ [H(\tau_n)] = u(x_n)\frac{4}{1+3\tau_n}H(\tau_n), \tag{13}$$

wherein $\tau_n = f'(y_n)/f'(x_n)$, and $u(x_n) = f(x_n)/f'(x_n)$. Substituting the conditions $H(1) = 1$, $H'(1) = 0$, and $H''(1) = 9/8$ from the proposed class (7) gives directly the family

$$\begin{cases} y_n = x_n - \frac{2}{3}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - u(x_n)G(s_n), \end{cases} \tag{14}$$

where $s_n = (3/2)(1 - \tau_n)$, and presented recently in the paper [3] with the conditions $G(0) = 1$, $G'(0) = 1/2$, $G''(0) = 1$ of the weight function $G(s_n)$. This simply shows the generality of the scheme (7).

Choosing different suitable weight functions in (7) according to Theorem 1 can provide more optimal two-point methods. We here note that such iterations without memory in which the order four can be attained by using two evaluations of the first-order derivative and one function evaluation are called in the literature as Jarratt-type schemes.

As another example from the class, we produce the following uni-parametric family

$$\begin{cases} y_n = x_n - \frac{2}{3}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{4f(x_n)}{f'(x_n)+3f'(y_n)}\left[1 + \frac{9}{16}\left(\frac{f'(y_n)}{f'(x_n)} - 1\right)^2 + \theta\left(\frac{f'(y_n)}{f'(x_n)} - 1\right)^3\right], \theta \in R, \end{cases} \tag{15}$$

with error equation $e_{n+1} = \left(-c_2c_3 + \frac{c_4}{9} + c_2^3\left(3 + \frac{64\theta}{27}\right)\right)e_n^4 + O\left(e_n^5\right)$.

Now, we aim at extending the class (7) for multiple roots and to obtain one of the main aims in this article by giving a

new general two-point multiple zero-finder. Li et al. in [14] developed the multiple root version of the third-order Halley-like family (5) as follows

$$\begin{cases} y_n = x_n - \beta\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{m\beta(m\beta+\beta-2m)\left(\frac{m-\beta}{m}\right)^m f(x_n)}{((m-\beta)^2-m\beta+m\beta^2)\left(\frac{m-\beta}{m}\right)^m f'(x_n)-(m-\beta)^2 f'(y_n)}, \quad \beta\neq 0. \end{cases} \tag{16}$$

Let us consider the case $\beta = \frac{2m}{m+2}$. Then, we obtain the multiple version of the method (6)

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n + \frac{4mp^m f(x_n)}{p^m(m^2+2m-4)f'(x_n)-m^2 f'(y_n)}, \end{cases} \tag{17}$$

where $p = \frac{m}{m+2}$. Let us further take into account the following class for multiple roots using again weight function approach

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n + \frac{4mp^m f(x_n)}{p^m(m^2+2m-4)f'(x_n)-m^2 f'(y_n)}[H(\tau_n)], \tau_n = \frac{f'(y_n)}{f'(x_n)}. \end{cases} \tag{18}$$

We show in what follows that this class is of fourth-order for multiple roots.

Theorem 2. *Let $x^* \in D$ be a multiple zero of a sufficiently differentiable function $f: D \subset R \to R$ for an open interval D with the multiplicity m, which includes x_0 as an initial approximation of x^*. Then, the family of methods without memory (18) is of optimal local order four, when $H(p^{m-1}) = 1$, $H'(p^{m-1}) = 0$, $H''(p^{m-1}) = -\frac{m^3(m-2)}{8p^{2m}}$, $|H^{(3)}(p^{m-1})| < \infty$.*

Proof. To find the asymptotic error constant of (18) where $C_j = \frac{(m)!}{(m+j)!} \times \frac{f^{(m+j)}(x^*)}{f^{(m)}(x^*)}$, $j \geqslant 1$, we expand any terms of (18) around the multiple root x^* in the nth iterate whence $e_n = x_n - x^*$. A Taylor expansion around x^* yields

$$f(x_n) = \frac{f^{(m)}(x^*)}{m!}e_n^m\left(1 + \sum_{j=1}^{\infty} C_j e_n^j\right), \tag{19}$$

and

$$f'(x_n) = \frac{f^{(m)}(x_n^*)}{(m-1)!}e_n^{m-1}\left(1 + \sum_{j=1}^{\infty}\frac{m+j}{m}C_j e_n^j\right), \tag{20}$$

so that using algebraic software,

$$\begin{aligned} x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)} &= x^* + \frac{m}{m+2}e_n + \frac{2C_1}{m(m+2)}e_n^2 \\ &\quad - \frac{((m+1)C_1^2 - 2mC_2)}{m^2(m+2)}e_n^3 \\ &\quad + \frac{2((-3m^2-4m)C_1C_2 + (m^2+2m+1)C_1^3 + 3m^2C_3)}{m^3(m+2)}e_n^4 + O(e_n^5). \end{aligned} \tag{21}$$

We similarly have

$$\begin{aligned} x_n + \frac{4m\,p^m f(x_n)}{p^m(m^2+2m-4)f'(x_n) - m^2 f'(y_n)} &= x^* - \frac{m-2}{m^3}C_1^2 e_n^3 \\ &+ \left(\left(-\frac{m^2+4m-8}{m^3}\right)\frac{C_1C_2}{m^3} + (m^4+8m^3-10m^2 \right. \\ &\left. + 4m - 12)\frac{C_1^3}{3m^5} + \frac{mC_3}{(m+2)^2}\right)e_n^4 + O(e_n^5). \end{aligned} \tag{22}$$

Furthermore, we have by Taylor expansion

$$\begin{aligned} \tau_n &= \frac{f'(y_n)}{f'(x_n)} \\ &= p^{m-1} - \frac{4p^m}{m^3}C_1 e_n + \left(\frac{4(m^2+2)p^m}{m^5}C_1^2 - \frac{8p^m}{m^3}C_2\right)e_n^2 \\ &\quad + \cdots + O(e_n^5). \end{aligned} \tag{23}$$

Thus, it would be easy to obtain

$$\begin{aligned} H(\tau_n) &= H(p^{m-1}) + H'(p^{m-1})(\tau_n - p^{m-1}) \\ &+ \frac{H''(p^{m-1})}{2}(\tau_n - p^{m-1})^2 + \frac{H^{(3)}(p^{m-1})}{6}(\tau_n - p^{m-1})^3 \\ &+ O(e_n^4) = 1 - \frac{(m-2)C_1^2}{m^3}e_n^2 \\ &+ \left(\frac{-4(m-2)C_1C_2}{m^3} - (16p^{3m}H^{(3)}(p^{m-1}) - 6m^5 + 6m^6 \right. \\ &\left. - 3m^7 + 12m^4)\frac{2C_1^3}{3m^9}\right)e_n^3 + O(e_n^4). \end{aligned} \tag{24}$$

Substituting Eqs. (21) and (23) into Eq. (18), and again using algebraic software, we have

$$\begin{aligned} H(p^{m-1}) = 1,\ H'(p^{m-1}) = 0,\ H''(p^{m-1}) \\ = -\frac{m^3(m-2)}{8p^{2m}}, |H^{(3)}(p^{m-1})| < \infty, \end{aligned} \tag{25}$$

to let the order arrive at four. By applying (25), the class (18) satisfies the error equation

$$\begin{aligned} e_{n+1} &= \left([m^8 + 2m^7 + 2m^6 - 8m^5 + 12\,m^4 \right. \\ &\left. + 32p^{3m}H^{(3)}(p^{m-1})]\frac{C_1^3}{3m^9} - \frac{C_1C_2}{m} + \frac{m\,C_3}{(m+2)^2}\right)e_n^4 + O(e_n^5). \end{aligned} \tag{26}$$

This shows that the suggested class reaches the local quartically convergence using three evaluations, i.e. the same to (1)–(4), but it is more general. The proof is complete. □

A simple choice for $H(\tau)$ in (18) satisfying Eq. (25) is $H(\tau_n) = 1 + \frac{9}{16}(\tau_n - p^{m-1})^2$ so that we get the multiple root version of the method (11):

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n + \frac{4m\,p^m f(x_n)}{p^m(m^2+2m-4)f'(x_n)-m^2 f'(y_n)}\left[1 - \frac{m^3(m-2)}{16p^{2m}}\left(\frac{f'(y_n)}{f'(x_n)} - p^{m-1}\right)^2\right], \end{cases} \tag{27}$$

with error equation

$$\begin{aligned} e_{n+1} &= \left(\frac{(m^4+2\,m^3+2\,m^2-8m+12)C_1^3}{3m^5} - \frac{C_1C_2}{m} + \frac{m\,C_3}{(m+2)^2}\right)e_n^4 \\ &\quad + O(e_n^5). \end{aligned} \tag{28}$$

Clearly, each member from the new class (18) for multiple zeros comprises three evaluations per full cycle, i.e. two derivative and one function evaluations. Accordingly, the derived methods and the classes are consistent with optimality hypothesis of Kung–Traub for construction optimal multi-point iterations without memory. They possess the optimal efficiency index $4^{1/3} \approx 1.587$.

Another example from the proposed class of iterations can be

$$\begin{cases} y_n = x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n + \frac{4m\ p^m f(x_n)}{p^m(m^2+2m-4)f'(x_n)-m^2 f'(y_n)} \\ \times\left[1 - \frac{m^3(m-2)}{16p^{2m}}\left(\frac{f'(y_n)}{f'(x_n)} - p^{m-1}\right)^2 - 1000\left(\frac{f'(y_n)}{f'(x_n)} - p^{m-1}\right)^3\right]. \end{cases} \quad (29)$$

Remark 2. Some well-known multiple zero finders belong to our class. For example, a slight transformation of (18),

$$\frac{4mp^m f(x_n)}{p^m(m^2+2m-4)f'(x_n) - m^2 f'(y_n)} H(\tau_n) = -u(x_n).\frac{4mp^m}{m^2\tau_n - p^m(m^2+2m-4)} H(\tau_n), \quad (30)$$

and the substitution of the conditions $H(p^{m-1}) = 1$, $H'(p^{m-1}) = 0$, $H''(p^{m-1}) = -m^3(m-2)/(8p^{2m})$, lead to the family of two-point methods for multiple roots presented by Zhou et al. in [30].

4. Finding the order of multiplicity

An important challenge in the iterative methods based on the known multiplicity is to find the order of multiplicity correctly. However, most algorithms to determine the order of multiplicity may lead to mutually opposite requirements. In what follows, three multiplicity-finding methods are reminded:

(i) Traub in [27] showed that

$$m \approx \frac{\log|f(x)|}{\log|f(x)/f'(x)|}, \quad (31)$$

when x is *very close* to the multiple root of f.

(ii) Lagouanelle in [13] proposed the following approximate formula

$$m \approx \frac{f'(x)^2}{f'(x)^2 - f(x)f''(x)}, \quad (32)$$

when once again x is *very close* to the multiple root of f.

(iii) A two-point way for finding the multiplicity has been given in [11] as

$$m \approx \frac{1}{f[x,y]}, \quad (33)$$

wherein $f[x,y] = \frac{f(x)-f(y)}{x-y}$ and also by assuming that x and y are *very close* to the multiple root of f. Therefore, m is approximately the reciprocal of the divided difference of f for successive iteration x and y. It may be computed, and displayed at each step along with the current iteration.

All of the above-mentioned ways demand a very close approximation to calculate a multiplicity of high accuracy. On the other hand, to find a very close approximation to a multiple root, it is necessary to use precise multiplicity. However, both of the requirements cannot be attained at the same time. Taking into account the opposite demands mentioned and additional calculations to find multiplicity, in those cases where we cannot provide an accurate multiplicity, it is sometimes better to apply a method which does not explicitly require the order of multiplicity (the aim of the next section), in spite of its lower computational efficiency arising from additional functional evaluations per iteration. For further discussion, see [9].

Note that the method (iii), is un-stable due to the use of divided difference. We need to use large number of floating point arithmetics, which increase the computational time. To clearly observe the above methods (i) and (ii), for finding the multiplicity-order, we provide the following illustration.

Illustration 1. We applied the three above-mentioned methods to find improved approximations to the root $x^* = 0$ of the multiplicity $m = 6$ for the nonlinear function $f(x) = x\sin(x) - 2\left(\sin\left(x/\sqrt{2}\right)\right)^2$. We chose $x_0 = 1.6$ as the initial approximation. Results are given in Table 1. The results show that the combination of the method (ii) and the scheme (11) (though it is of order one for multiple zeros) performs really well in terms of the number of evaluations alongside the CPU running time.

All computations have been done in MATHEMATICA 8 [6,7]. For example, the scheme (11) by using (ii) could be coded in what follows in which 32 digits has been used with the stopping criterion 0.001 on the two successive approximations of the multiplicity order.

```
ClearAll[n, x, flx, fly, m]
f[x_] := x * Sin[x] - 2 (Sin[x/Sqrt[2]])^2;
n = 0; m = 1; x = 1.6;
While[Abs[(f'[x]^2)/(f'[x]^2 - f[x] * f''[x])-
m] > 0.001,
  m = SetAccuracy[(f'[x]^2)/(f' [x]^
2 - f[x] * f''[x]), 32];
  fx = f[x]; flx = f'[x];
  y = SetAccuracy[x - (2/3)*(fx/flx), 32];
fly = f'[y];
  x = SetAccuracy[x - ((4 fx)/(flx + 3 fly)) * (1 + (9/
16) * (fly/flx - 1)^2), 32];
  n++]; // AbsoluteTiming
Print[n]
Print[Round[m]]
```

5. Unknown multiplicity

We now consider the application of iteration function (11) to the case when the order of multiplicity is unknown. Let x^* be a multiple root of a function $f(x)$, then x^* is a simple root of the function $h(x) := f(x)/f'(x)$. In this way, the order of convergence will be preserved by costing more functional evaluations.

Table 1 Approximate multiplicity.

Iterative methods	Newton	(6)	(11)
Multiplicity method (i)			
Number of iterations	70	45	40
Computational time	0.1	0.07	0.06
Multiplicity method (ii)			
Number of iterations	17	12	11
Computational time	0.03	0.03	0.03

Table 2 The considered test functions in this paper.

Test Functions	Zeros	Multiplicity
$f_1(x) = ((\sin(x))^2 + x)^5$	$x_1^* = 0$	5
$f_2(x) = (1 + x + \cos(\frac{\pi x}{2})$ $-\sqrt{1-x^2})^3$	$x_2^* \approx -0.728584046444826\ldots$	3

Remark 3. Existing literature has attributed this transformation function to make the multiple root as a simple one to various authors. However, Schroder was the first to derive this method in his paper [17]. Note that it is also known as Traub's transformation [27].

Now by applying the optimal fourth-order method (11) on the transformation $h(x)$, we can easily extend it for dealing with multiple roots, when high precision alongside high order is needed. For (11), we thus obtain

$$\begin{cases} y_n = x_n + \frac{2f(x_n)f'(x_n)}{-3f'(x_n)^2+3f(x_n)f''(x_n)}, \\ x_{n+1} = x_n - \frac{4f(x_n)\left(1+\frac{9}{16}\left(-1+\frac{1-v_n}{1-u_n}\right)^2\right)}{f'(x_n)(4-u_n-w_n)}, \end{cases} \tag{34}$$

wherein

$$\begin{cases} u_n = \frac{f(x_n)f''(x_n)}{f'(x_n)^2}, \\ v_n = \frac{f(y_n)f''(y_n)}{f'(y_n)^2}, \\ w_n = \frac{3f(y_n)f''(y_n)}{f'(y_n)^2}. \end{cases} \tag{35}$$

Therefore, now we have an efficient method (34) of order four for finding the multiple roots too. All the other optimal simple root solvers can be easily constructed in this way for finding multiple zeros, when high precision is required. Note that until now, we have distinguished two kinds of methods; those which deal with a known order of multiplicity and others, such as (34), with no information on multiplicity.

Remark 4. When the multiplicity of the zeros for a nonlinear function is quite high, for instance $m > 15$, then finding the m, explicitly, in an acceptable piece of time is a hard task. Thus, one may rely on iterative methods such as (34), in which the multiple zeros could be found without the knowledge of multiplicity in general. Some robust discussions for this problem and how to resolve them are given in [4].

6. Numerical testing

We have tested the class (18) of the two-point methods without memory using the programming package MATHEMATICA 8. Apart from this class, some two-point iterative methods (2)–(4) of optimal order four for multiple roots, which also require three functional evaluations and the same computational efficiency, have been tested. The list of test nonlinear functions including multiple roots with their multiplicity are presented in Table 2.

The results of comparisons are summarized in Tables 3, 4 after three full iterations, respectively, for two different initial guesses. As they show, novel schemes are comparable with all of the methods. All numerical instances were performed by using 500 digits floating point arithmetic. We have computed the root of each test function for the initial guess x_0.

It should be remarked that all of the discussed iterative methods up to now, have the ability to find complex zeros as well. For this aim, one may apply a complex initial guess to find the complex solution.

As can be seen, the obtained results in Tables 3, 4 are in harmony with the analytical procedures given in Section 3. Although two-point method (27) produce the best approximations in the case of considered functions, we cannot claim that, in general, they are better than other two-point methods without memory for multiple roots of optimal local order four; the tests show that the considered methods generate results of approximately same accuracy.

An efficient way to observe the behavior of the order of convergence is to use the (local) computational order of convergence. We use the following way

Table 3 Results of comparisons for different methods after three full iterations.

$\lvert f\rvert$	x_0	(2)	(3)	(4)	(27)	(29)
$\lvert f_1(x_3)\rvert$	0.3	1.77×10^{-170}	1.80×10^{-165}	1.29×10^{-164}	9.32×10^{-170}	4.06×10^{-170}
ρ_1		3.99	3.99	3.99	3.99	3.99
$\lvert f_2(x_3)\rvert$	−0.6	1.04×10^{-153}	2.87×10^{-152}	4.84×10^{-152}	1.39×10^{-152}	2.49×10^{-153}
ρ_2		3.99	3.99	3.99	3.99	3.99

Table 4 Results of comparisons for different methods after three full iterations.

$\lvert f\rvert$	x_0	(2)	(3)	(4)	(27)	(29)
$\lvert f_1(x_3)\rvert$	0.2	1.04×10^{-153}	2.87×10^{-152}	1.77×10^{-170}	1.77×10^{-170}	1.77×10^{-170}
ρ_1		3.99	3.99	3.99	3.99	3.99
$\lvert f_2(x_3)\rvert$	−0.8	3.58×10^{-143}	7.94×10^{-143}	3.16×10^{-142}	1.35×10^{-143}	1.58×10^{-145}
ρ_2		3.99	3.99	3.99	3.99	3.99

$$\rho \approx \frac{\ln\left|\frac{f(x_{n+1})}{f(x_n)}\right|}{\ln\left|\frac{f(x_n)}{f(x_{n-1})}\right|}. \tag{36}$$

The computer characteristics are Microsoft Windows XP Intel(R), Pentium(R) 4 CPU, 3.20 GHz with 4 GB of RAM, throughout this paper.

In this work, the computational time has been computed using the command `AbsoluteTiming[]`. Note that CPU run time is not unique and completely depends on the specification of the Computer, but herein we present a mean over 5 performances to ensure the readers of their robustness. We also report the computational order of convergence using (36), and in fact based on $\rho \approx \frac{\ln|f(x_3)/f(x_2)|}{\ln|f(x_2)/f(x_1)|}$.

7. Finding all the real solutions

Knowledge of efficiency is of interest in designing a package of root-solvers. Hopefully the schemes from the class (18) are of optimal fourth-order. Furthermore, as given in Remark 4, and also throughout the paper till now, the most important aspect of iterative methods for solving nonlinear equations is to have robust initial approximations of the solutions to start the process. Unfortunately, an inappropriate initial guess will ruin the whole theoretical aspects and may lead to unwanted solution or divergency.

To remedy this, only a few well-known works are available in the literature. For example, Johnson and Tucker [8] recently proposed an efficient quadrature approach to find the number of roots inside a given rectangle and to calculate their multiplicities. Their method is based on the argument principle and supported by the use of validated integration of contour integrals.

The newest way for this aim, is the work of Yun in [29]. In fact, author solved $f(x) = 0$ with having finitely many roots in a bounded interval, based on the so-called numerical integration method (Signum Based Iteration) without any initial guess, and then he applied iterative methods to obtain all the (real) roots of the nonlinear equations. In the result, an algorithm to find all of the simple roots and the multiple ones as well as the extrema of $f(x)$ was suggested. The problem of this technique is to compute a numerical integration on the intervals, furthermore when having so many zeros in an interval or we have root-cluster, the implementation of this technique might be costly.

In this section, we propose an algorithm to find robust initial guesses and then correct their accuracies by the proposed methods of this paper. Toward this goal, we pay heed to the interval mathematics [10,15]. Using an interval scheme such as Bisection method we provide a list of real approximations for the given nonlinear function on a bounded domain.

Note that, by using the computer algebra software MATHEMATICA 8 [28], we are able to work on a list very easily. In this case, and for having accuracy along with simplicity we code our algorithm in this programming package.

Therefore, in our *hybrid algorithm*, we first provide a sequence of starting points based on interval bisection method which could have some flexibility to find all the intervals (up to any pre-desired accuracy) and then in the second step, we apply the iterative methods given in Sections 3 and 5 to boost up their accuracies.

We now first set up the primitive requirements for the interval bisection method as follows

```
ClearAll[f, x, y, fx, fy, f1x, f2x, fy, f1y, f2y,
setInitial, u, v, w, j, NumberOfGuesses, nMax]
intervalbisection::rec = "MaxRecursion exceeded.";
split[f_, x_, int_Interval, eps_, n_] :=
  Block[{a=int[[1, 1]], b=int[[1, 2]], c},
    If[! IntervalMemberQ[f /. x → int, 0],
Return[{}]];
    If[b − a < eps, Return[int]];
    If[n==0, Message[intervalbisection::rec];
      Return[int]];
    c = (a + b)/2;
    split[f, x, #, eps, n − 1] & /@ {Interval[{a, c}],
       Interval[{c, b}]}];
Options[intervalbisection]
= {MaxRecursion → 1000};
```

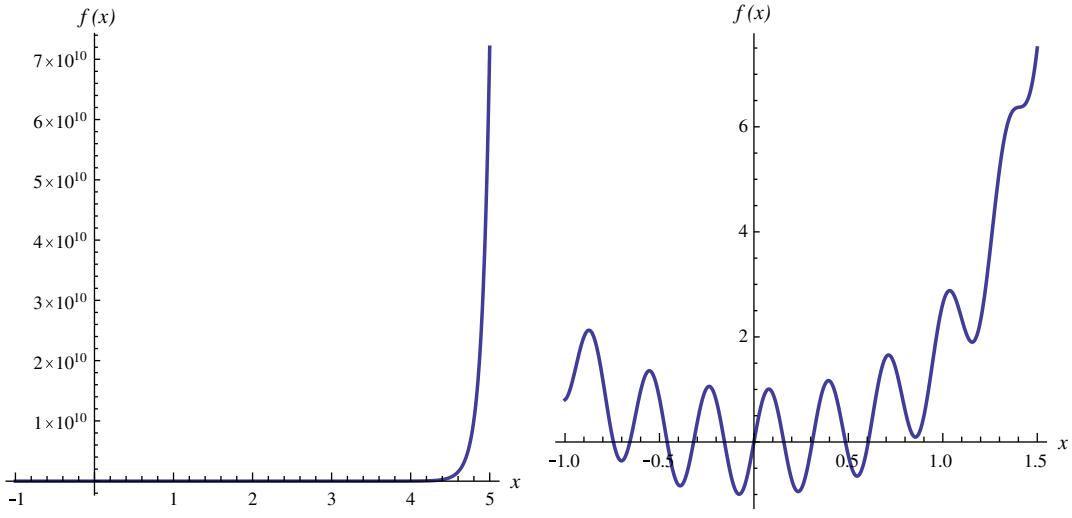

Figure 1 The plot of the function $f(x)$ in two different views (intervals) including the zeros.

In the above piece of code, the mid-point (i.e. c) of the interval $[a,b]$ including a zero, is calculated until all real zeros are detected (up to the machine precision). The note is that we considered the maximum recursion to be 1000. Clearly, if one knows that the interval has a few zeros then, this could also be changed to some lower value, while if we have a root cluster in which there are so many zeros, then to capture all of them, this setting must be considered tighter. Note that one reason for choosing interval bisection method as the predictor is that, it is totally based on having an interval in each iteration containing a zero. That is to say, no matter we have simple or multiple zero, and how the function is so much ill-conditioned, it mostly successfully finds intervals with roughly a same convergence rate.

We are now able to apply the interval bisection method to capture all the intervals having one unique solution in themselves by

```
intervalbisection[f_, x_, intab_, eps_, opts___] :=
Block[{int, n,}
  n = MaxRecursion /. opts /.
Options[intervalbisection];
  int = Interval /@ (List @@ intab);
IntervalUnion @@ Flatten[split[f, x, #, eps, n] & /@
int]];
```

In the above lines `eps` is the tolerance. We also may note that we are working with double precision in the first step of the hybrid algorithm to rapidly obtain a list of robust initial intervals. In most practical problem `eps` $= 10^{-4}$ is enough, but in some cases it would be necessary to increase this tolerance to capture all the solutions in a root cluster.

At this time, we have a list of intervals each including a zero. Thus, now we pick out the mid point of each interval as the robust initial approximation for the zeros. We now keep going by considering an oscillatory function in an interval. We consider the nonlinear function $f(x) = \sin(20x) + \exp(x^2) - 1$, on the interval $D = [-1., 5.]$. The tolerance is chosen to be 0.0001. The last line in the following piece of code extract all the mid-points of the intervals (including a zero), and provide our robust list of starting points.

```
f[x_] := Sin[20 x] + Exp[x^2] - 1;
IntervalSol = intervalbisection[f[x], x,
       Interval[-1., 5.], 0.0001];
setInitial=Mean /@ List @@ IntervalSol
```

The graph of the function f has been portrayed in Fig. 1. One may observe that in the considered interval, there are some roots next to each other. The number of zeros can now be obtained as follows:

```
NumberOfGuesses = Length[setInitial]
nMax = 2;
```

The `nMax` is the number of full cycles, we are going to let the iteration functions take to correct the accuracy of the initial points. In this example, we have 10 simple zeros in the given interval, i.e. $\{-0.74379, -0.656815, -0.459335, -0.319534, -0.155838, -0.0000152588, 0.158371, 0.309158, 0.484665, 0.605423\}$. The second step of our hybrid algorithm now could be written as comes next:

```
digits = 528;
For[j = 1, j <= NumberOfGuesses, j++,
  {x = setInitial[[j]],
    If[Round[f'[x]]==0,
     {Do[
       fx = SetAccuracy[f[x], 528];
flx = SetAccuracy[f'[x], digits];
       f2x = SetAccuracy[f''[x], digits];
       y = SetAccuracy[x + (2 fx * flx)/
(-3 * flx^2+3 * fx * f2x), digits];
       fy = SetAccuracy[f[y], digits];
fly = SetAccuracy[f'[y], digits];
       f2y = SetAccuracy[f''[y], digits];
       u = SetAccuracy[(fx * f2x)/(flx^2), digits];
       v = SetAccuracy[(fy * f2y)/(fly^2), digits];
       w = SetAccuracy[(3 * fy * f2y)/(fly^2),
digits];
       x = SetAccuracy[x - (4 fx (1 + (9/16) * (-1
         + (1 - v)/(1 - u))^2))/(flx * (4 - u - w)),
digits]; , n, nMax];
     Print[Column[
        {"The zero is multiple=" x,
        ""The zero is multiple and its function
        value is =" N[f[x]]}, Frame -> All]];},
     {Do[
fx = SetAccuracy[f[x], digits]; f1x = SetAccuracy[f'[x],
digits];
y = SetAccuracy[x - (2/3) * (fx/flx), digits];
fly = SetAccuracy[f'[y], digits];
x = SetAccuracy[x - ((4 fx)/(flx + 3 fly)) * (1 + (9/
16) * (fly/flx - 1)^2), digits];
, {n, nMax}];
Print[Column[
     {"The zero is simple=" x,
       ""The zero is simple and its function
       value is =" N[f[x]]}, Frame -> All]];}];};];
```

We provided the test `If[Round[f'[x]]== 0, ..., ...]`, to distinguish that the zero is simple or multiple. In the above piece of code, 528 digits floating point has been used, this could be changed according the aims of the users as well. Note that in the following first and third example we considered $nMax = 3$ while for the second example we have chosen $nMax = 1$ because of having multiple zeros. Thus, the number of digits must be enough high when dealing with multiple zeros along the fact that the considered stopping criterion might be replaced with another one. The final results for this problem and the solutions that we have obtained are reported in Table 5. Note that the whole computational time is 0.12 s.

In the suggested way, all the real solutions of the nonlinear (twice differentiable) function in a bounded domain can be easily attained simultaneously up to any number of decimal places. We are able now to find all the critical points in an interval without the difficulty in computing numerical integration based scheme of Yun in [29]. An important attention should be paid to the *options* we have taken into account. In some situations, some tighter conditions on the first step must be taken to capture all the real solutions. It should be re-

Table 5 Results of applying the hybrid algorithm for $f(x) = \sin(20\ x) + \exp(x^2) - 1$.

Initial point	Absolute value of $f(x_3)$
−0.74379 (Simple)	1.65×10^{-212}
−0.656815 (Simple)	3.23×10^{-243}
−0.459335 (Simple)	3.97×10^{-230}
−0.319534 (Simple)	9.24×10^{-272}
−0.155838 (Simple)	3.59×10^{-262}
−0.0000152588 (Simple)	4.98×10^{-281}
0.158371 (Simple)	2.69×10^{-258}
0.309158 (Simple)	5.19×10^{-251}
0.484665 (Simple)	4.99×10^{-238}
0.605423 (Simple)	1.69×10^{-228}

Table 6 Results of applying the hybrid algorithm for illustration 2.

Initial point	Absolute value of $g(x_1)$
0.333319 (Simple)	5.46×10^{-14}
0.666672 (Multiple)	1.52×10^{-61}
1.24999 (Simple)	1.14×10^{-13}
1.49998 (Multiple)	0
1.87498 (Simple)	9.97×10^{-12}

marked that for very oscillatory functions in an interval, or when the considered interval is too large, we recommend the users to first divide the main interval into some sub-intervals and then apply the above-mentioned algorithm to capture all the real roots properly. Applying the hybrid algorithm on $f'(x)$, will produce the extrema of $f(x)$ as well.

In what follows, we give two more examples to clearly observe the efficiency of the new algorithm in solving nonlinear equations.

Illustration 2. Let us find the zeros of the nonlinear function $g(x) = (2x - 3)^2(3x - 2)^4(96x^3 - 332x^2 + 325x - 75)$, on the bounded domain $D = [0.2, 2.]$. The results for solving this problem are listed in Table 6. The plot of $g(x)$ is given in Fig. 2.

Clearly, the first part of the above code, will produce the following list of initial approximations for the zeros in the interval: $\{0.333319, 0.666672, 1.24999, 1.49998, 1.87498\}$ with accuracy up to 4 decimal places at least. Now, we have a sequence of initial points, which guarantee the convergence order. The second part of our code will correct the accuracies and also distinguish that which zero is simple and which one is multiple. In this test, there are three simple and two multiple zeros.

The whole computational time for obtaining the results in Table 5 is only 0.25 s.

Remark 5. In case of having multiple zero in the interval, and though the algorithm is of fourth order, we cannot attain the rate of fourth order convergence in correcting the starting points per full cycle. Note that in the error ball, the accurate computation of $f(x)$ is not possible and that is why the rate of correcting the number of decimal places is lower than that of simple zeros.

Illustration 3. In this test, we try to find all the solutions of the function $h(x) = -\frac{100x^2}{\sqrt{10000x^4+1}} + \sin(10x) + \frac{1}{2}$, on the interval $D = [0, 30]$. The plot of $h(x)$ is given in Fig. 3.

The sequence of initial approximations would be {0.262985, 0.680637, 0.890121, 1.309, 1.51843, 1.93731, 2.14677, 2.56562, 2.77508, 3.19396, 3.40339, 3.82227, 4.03172, 4.45058, 4.66001, 5.07892, 5.28834, 5.70723, 5.91668, 6.33554, 6.54496, 6.96387, 7.1733, 7.59218, 7.80164, 8.22049, 8.42992, 8.84883, 9.05826, 9.47714, 9.68659, 10.1054, 10.3149, 10.7338, 10.9432, 11.3621, 11.5716, 11.9904, 12.1998, 12.6187, 12.8282, 13.247, 13.4565, 13.8754, 14.0848, 14.5037, 14.7131, 15.132, 15.3415, 15.7603, 15.9697, 16.3887, 16.5981, 17.0169, 17.2264, 17.6453, 17.8547, 18.2736, 18.483, 18.9019, 19.1114, 19.5302, 19.7397, 20.1586, 20.368, 20.7868, 20.9963, 21.4152, 21.6246, 22.0435, 22.253, 22.6718, 22.8813, 23.3001, 23.5096, 23.9285, 24.1379, 24.5568, 24.7662, 25.1851, 25.3945, 25.8134, 26.0229, 26.4417, 26.6512, 27.0701, 27.2795, 27.6984, 27.9078, 28.3267, 28.5362, 28.955, 29.1644, 29.5833, 29.7928}. In this test, as could be observed by running the above code, there would be 95 simple zeros in the considered interval. This reveals the difficulty of solving

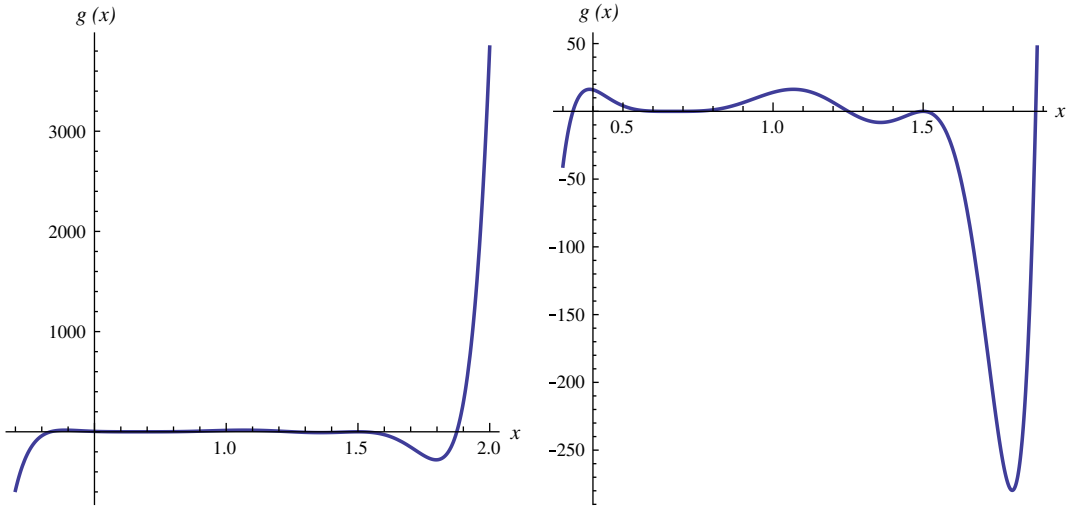

Figure 2 The plot of the function $g(x)$ in two different views (intervals) including the zeros.

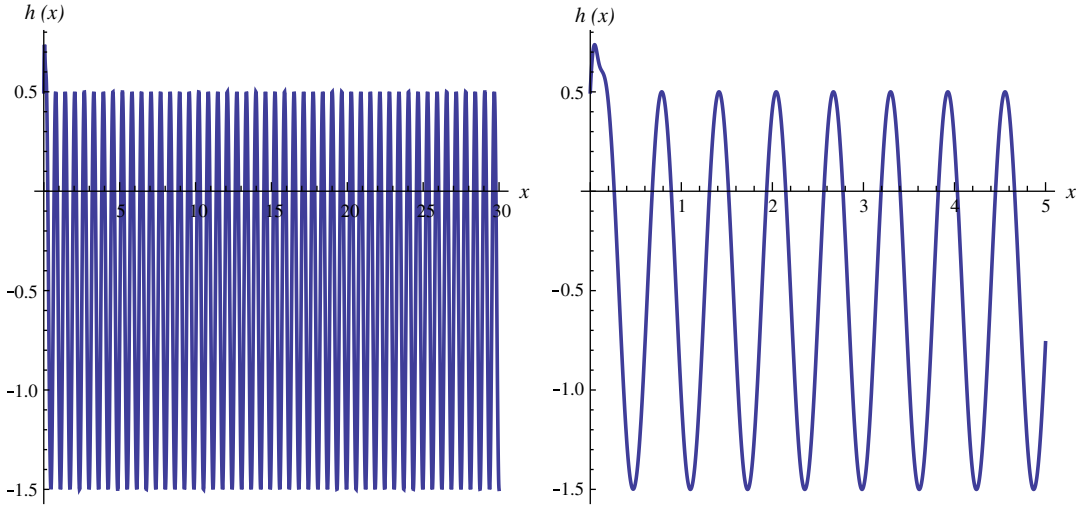

Figure 3 The plot of the function $h(x)$ in two different views (intervals) including the zeros.

nonlinear equations with finitely many roots in an interval, however the above algorithm could fulfill this need.

Due to page limitations, we do not provide the corrected values of the zeros after three full iterations. But the interesting point is that using the above algorithm and without any guess, we could find the zeros with at least 200 correct decimal places, in only 1.82 s which clearly manifest the applicability of the new hybrid algorithm.

8. Concluding remarks

The behavior of root-finding algorithms is studied in numerical analysis. Algorithms perform best when they take advantage of known characteristics of the given function. Besides, further attention for multiple root solvers need to be done along to write general codes to capture all the real solutions of nonlinear equations in an interval, since for this purpose only a few literature is available.

Hence, this paper has contributed a general way for solving nonlinear equations using two-point root solvers. Discussion on multiple roots have been done and it was observed that any derived method from the classes (7) and (18) comprises three functional evaluations, which shows their optimality in the sense of Kung–Traub. We have also extended one method for finding multiple zeros when the multiplicity is unknown.

Some methods from the class of multiple root solvers were tested through numerical examples. From Tables 2, 3 and the tested examples, we can conclude that all implemented methods for multiple roots converge fast, when the multiplicity of the root is available and the initial guesses are in the vicinity of the multiple zeros.

Fulfilling these two needs have also been discussed fully in the work. We also used the programming package MATHEMATICA 8 in our calculations and gave the necessary cautions and pieces of codes for the users to implement them in their own problems as easily as possible. The second aim of this paper was achieved by designing a hybrid algorithm to capture all the solutions of nonlinear equations as rapidly as possible. The algorithms worked efficiently in hard test problems. And thus, the proposed algorithm could be easily used in practical problems.

We end the paper by furnishing the outlines for future studies. We worked on all the aspects of the iterative methods to capture all the real solutions. Thus, the future work could focus on finding a robust way to find accurate complex initial guesses to find the complex zeros in a rectangle using optimal methods as well. Furthermore, the new quartcially convergent algorithm requires the functions to be twice differentiable in the interval, and in case of finding exterma, one needs the 3rd order derivative as well. Hence, it will be appropriate if the hybrid algorithm could have an optimal high-order Steffensen-type corrector, to resolve this problem (only once differentiable) in finding the critical points. This implementation could be one of the next aims.

References

[1] M. Aslam Noor, M. Waseem, Some iterative methods for solving a system of nonlinear equations, Comput. Math. Appl. 57 (2009) 101–106.
[2] D.K.R. Babajee, Analysis of Higher Order Variants of Newton's Method and their Applications to Differential and Integral Equations and in Ocean Acidification, Ph.D. thesis, University of Mauritius, December 2010.
[3] C. Chun, M.Y. Lee, B. Neta, J. Dzunic, On optimal fourth-order iterative methods free from second derivative and their dynamics, Appl. Math. Comput. 218 (2012) 6427–6438.
[4] B.H. Dayton, T.-Y. Li, Z. Zeng, Multiple zeros of nonlinear systems, Math. Comput. 80 (2011) 2143–2168.
[5] A. Galantai, C.J. Hegedus, A study of accelerated Newton methods for multiple polynomial roots, Numer. Algor. 54 (2010) 219–243.
[6] R. Hazrat, Mathematica: A Problem-Centered Approach, Springer-Verlag, 2010.
[7] J. Hoste, Mathematica Demystified, the McGraw-Hill Companies, Inc., 2009.
[8] T. Johnson, W. Tucker, Enclosing all zeros of an analytic function–a rigorous approach, J. Comput. Appl. Math. 228 (2009) 418–423.
[9] N.N. Kalitkin, I.P. Poshivailo, Determining the multiplicity of a root of a nonlinear algebraic equation, Comput. Math. Math. Phys. 48 (2008) 1113–1118.

[10] J.B. Keiper, Interval arithmetic in mathematica, Math. J. 5 (1995) 66–71.
[11] R.F. King, A secant method for multiple roots, BIT 17 (1977) 321–328.
[12] H.T. Kung, J.F. Traub, Optimal order of one-point and multipoint iteration, J. ACM 21 (1974) 643–651.
[13] J.L. Lagouanelle, Sur une mtode de calcul de l'ordre de multiplicit des zros d'un polynme, C.R. Acad. Sci. Paris Sr. A 262 (1966) 626–627.
[14] S.G. Li, H. Li, L.Z. Cheng, Some second-derivative-free variants of Halley's method for multiple roots, Appl. Math. Comput. 215 (2009) 2192–2198.
[15] R.E. Moore, R.B. Kearfott, M.J. Cloud, Introduction to Interval Analysis, SIAM, 2009.
[16] H. Saberi Nik, F. Soleymani, A Taylor-type numerical method for solving nonlinear ordinary differential equations, Alexandria Eng. J. (2013), http://dx.doi.org/10.1016/j.aej.2013.02.006.
[17] E. Schroder, Uber unendlich viele algorithmen zur Auflsung der Gleichungen, Math. Ann. 2 (1870) 317–365.
[18] M. Sharifi, D.K.R. Babajee, F. Soleymani, Finding the solution of nonlinear equations by a class of optimal methods, Comput. Math. Appl. 63 (2012) 764–774.
[19] J.R. Sharma, R. Sharma, Modified Jarratt method for computing multiple roots, Appl. Math. Comput. 217 (2010) 878–881.
[20] F. Soleymani, F. Soleimani, Novel computational derivative-free methods for simple roots, Fixed Point Theory 13 (2012) 247–258.
[21] F. Soleimani, F. Soleymani, S. Shateyi, Some iterative methods free from derivatives and their basins of attraction, Discrete Dynamics in Nature and Society, vol. 2013, Article ID 301718, 11 pages
[22] F. Soleymani, D.K.R. Babajee, T. Lotfi, On a numerical technique for finding multiple zeros and its dynamic, J. Egypt. Math. Soc. (2013), http://dx.doi.org/10.1016/j.joems. 2013.03.011.
[23] F. Soleymani, Some optimal iterative methods and their with memory variants, J. Egypt. Math. Soc. (2013), doi:/10.1016/j.joems.2013.01.002.
[24] F. Soleymani, Some efficient seventh-order derivative-free families in root-finding, Opuscula Math. 33 (2013) 163–173.
[25] M.D. Stuber, V. Kumar, P.I. Barton, Nonsmooth exclusion test for finding all solutions of nonlinear equations, BIT 50 (2010) 885–917.
[26] R. Thukral, New sixteenth-order derivative-free methods for solving nonlinear equations, Am. J. Comput. Appl. Math. 2 (2012) 112–118.
[27] J.F. Traub, Iterative Methods for the Solution of Equations, Prentice Hall, New York, 1964.
[28] S. Wagon, Mathematica in Action, Third edition., Springer, 2010.
[29] B.I. Yun, Iterative methods for solving nonlinear equations with finitely many roots in an interval, J. Comput. Appl. Math. 236 (2012) 3308–3318.
[30] X. Zhou, X. Chen, Y. Song, Constructing higher-order methods for obtaining the multiple roots of nonlinear equations, J. Comput. Appl. Math. 235 (2011) 4199–4206.