Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 215 (2024) 1–20
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

A superlinear Scaling Factor Regula Falsi root finder that detects the simple or multiple character of the root

Julio M. Fernández-Díaz[a,*], César O. Menéndez-Pérez[b,1]

[a] Department of Physics, University of Oviedo, E.P. de Mieres, C/Gonzalo Gutiérrez Quirós, s/n, 33600, Mieres, Spain
[b] Department of Mathematics, University of Oviedo, Spain

Received 3 May 2023; received in revised form 31 July 2023; accepted 3 August 2023
Available online 9 August 2023

Abstract

In this work a new method of Scaling Factor Regula Falsi type is developed, by using parabolic interpolation. It has global convergence and a high computational efficiency for simple roots. The method, not being a hybrid one, allows changing the function used for calculating the scaling factor at every iteration, making possible to switch to other more adequate methods for multiple roots (e.g., a generalised Illinois one). As an important feature of the method, it allows determining with no additional calculations, whether the root is simple or multiple. The developed algorithm has been tested with numerous functions extracted from the bibliography, performing, in most cases, better than some routines (like brentq, brenth and toms748) found in common numerical libraries. Fully operative routines of the new method are provided as supplementary files in four different programming languages: `python`, `lua`, `C` and `Fortran90`.

Keywords: Root finding; Non-linear equation; Scaling Factor Regula Falsi; Bracketing

1. Introduction

The solution of $f(x) = 0$ in $\mathbb{R}$ is a main task in many engineering and science fields. Therefore, mathematicians and computer scientists have done much work on it, by creating methods and routines as efficient as possible, which give both the value of a root and information on the circumstances in which a solution has been reached.

A relevant set of methods are those using the Bolzano's Theorem: If $f \in C^0_{[a,b]}$, $f(a)f(b) < 0$, $a, b \in \mathbb{R}$ at least a root $\alpha \in [a, b]$ exists. They are methods of global convergence.

A wide collection of iterative methods with high convergence order has been developed, sometimes needing first and higher order derivatives, and often multipoint. Despite this exuberance of methods, just a few of them are included in the libraries in use nowadays (we will not comment on the reasons).

In some engineering problems the calculation of $f(x)$ is expensive, and it could be neither convenient nor feasible the derivative calculation. In those cases methods without derivatives are preferred, even with lower convergence order.

* Corresponding author.
E-mail addresses: julio@uniovi.es (J.M. Fernández-Díaz), omar@uniovi.es (C.O. Menéndez-Pérez).
[1] Retiree.

https://doi.org/10.1016/j.matcom.2023.08.003

Nevertheless, if the initial guess of the root place is bad it is not possible to reach the theoretical convergence order of the method, until the successive approximations are near enough the actual root. In this case, several iterations are consumed before the real power of the method is revealed.

Another subject is the conversion of mathematical derivations into useful algorithms, mainly since infinite precision is not possible. Nowadays, in scientific calculations, it is customary to use IEEE-754 double precision floating point [16], with a machine epsilon of about 2.2×10^{-16}. This limits the way we design algorithms from mathematical methods.

No perfect method exists, and every one has its advantages and drawbacks. Often the solving routine, at each iteration, intermixes various methods depending on some computational parameters during calculations. Therefore, ultimately they are hybrid methods.

Of this kind are the typical routines appearing in the numerical libraries. For example, in `python`, inside module `scipy.optimize` are `toms748`, `brentq` and `brenth`, here ordered perhaps by increasing efficiency. These routines perform well for simple roots, but they are far less efficient treating multiple ones, needing several times the number of function calls (*nfun*) than bisection.

For this cause, a routine performing like these above-mentioned for simple roots, but with a better handling of multiple ones, would be welcome. An initial guess for this is the use of Steffensen-like methods, for example, in [18,24,31]. Terms in the form $K(x) = f(x + \beta f(x)) - f(x)$ appear in these methods, in which β is more or less constant.

However, in these articles the need of multi-precision (until 1000 significant figures) are compulsory. Otherwise, the corresponding iterative formulas are not applicable. By naming ϵ the machine epsilon, in an open domain $U = \{x \mid |x - \alpha| < \delta, |\beta f(x)| < \epsilon|x|\}$, it is $K(x \in U) = 0$ and the calculation losses precision completely as multiplicity grows above a small value, typically for $m > 3$ in double precision.

For all the above, the search of suitable routines (in double precision) that behave well for simple roots and improve the performance of the state-of-the-art ones (e.g., `brentq` and `brenth`) for multiple ones is still an open work. It should be noted that for multiple roots the recommended method in double precision is bisection. The problem is that we do not know beforehand whether the root is simple or multiple.

Regula Falsi (RF) is one of the method without derivatives. Since frequently this method and the secant one are confused (see [22] for a comprehensive explanation), we use the definitions of [30, p. 112] for RF method (also named False Position), and of [30, p. 118] for the secant one.

RF has global convergence, but it is usually slow. However, some variants of several kinds, known as "Modified Regula Falsi" are faster, and they do not need to hybridise in order to keep global convergence. A typical methodology is to modify the ordinate at one bracketing extreme multiplying it by some factor. In [12] it is demonstrated that methods of this kind, which can be named generically Scaling Factor Regula Falsi (SFRF), form a bi-parametric family of methods with interesting properties.

In [12] the properties of the scaling factor, γ, are analysed for both the convergence of the successive approximations of the root and the bracketing interval radius to go to zero. Also, an appealing property is exposed: after each iteration we can change the way in which γ is calculated, always keeping the global convergence.

In this article we design a new SFRF method, which uses direct parabolic interpolation to calculate the scaling factor. We call it PRF from "Parabolic Regula Falsi". We will demonstrate the convergence properties, both for simple and multiple roots, setting up the working conditions in both cases.

A very important property of PRF method is that it detects whether the root is simple or not, and it will allow us to treat the multiple root cases in an acceptable way as compared to bisection.

Finally, we will show the goodness of the PRF method, by comparing it to others fully established in the numerical codes nowadays in use, for a set of functions with simple and multiple roots.

Moreover, in the Supplementary files, full operative implementations of these routines in `python`, `lua`, `C` and `Fortran90` are provided.

2. Scaling Factor Regula Falsi method

The SFRF method can be used to calculate a root α of function $f \in C^0([a, b])$ starting with $[a, b]$ under the conditions of Bolzano's Theorem, $f(a)f(b) < 0$. It generates two sequences: one, $\{x_n\}$, with the approximations to the root; another with the enclosing intervals $\{[a_n, b_n]\}$, such that:

$$x_n \in [a_n, b_n] \subset [a_{n-1}, b_{n-1}] \subset \cdots \subset [a_1, b_1] \subset [a, b], \qquad f(a_i)f(b_i) < 0, \quad i = 1, 2, \ldots, n. \tag{1}$$

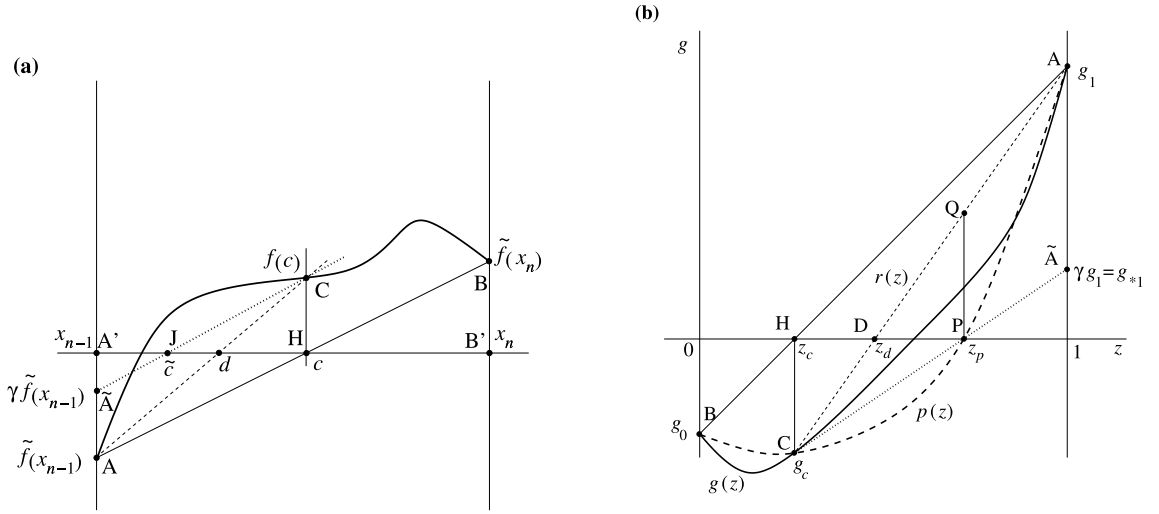


Fig. 1. (a) The first scaled step in the Scaling Factor Regula Falsi after the initial secant one. Points A, H and B are collinear by construction. C, J and $\tilde{A}$ are also collinear from (3). (b) The SFRF method in transformed coordinates (see the text). $(x_n, \tilde{f}(x_n)) \xrightarrow{T} (0, g_0)$, $(x_{n-1}, \tilde{f}(x_{n-1})) \xrightarrow{T} (1, g_1)$, $(c, f(c)) \xrightarrow{T} (z_c, g_c)$, $(x_{n-1}, \gamma\tilde{f}(x_{n-1})) \xrightarrow{T} (1, \gamma g_1)$. Besides, $z_c = -g_0$ and the straight line AB has slope of 1. The dashed line corresponds to $p(z)$, a parabola that crosses points B, C and A, not the function $g(z)$. The straight line $r(z)$ crosses C and A.

Before describing the algorithm we need some notation.

Definition. The scaling factor for the ordinate $\gamma \in \mathbb{R}$ fulfils $0 < \gamma < 1$, and it is in general a function of the data known at current iteration. Every SFRF method has a different way to calculate γ.

Definition. We call "ordinate associated to x" some value $\tilde{f}(x)$ that verifies:

$$0 < |\tilde{f}(x)| = \gamma |f(x)| \leq |f(x)|, \qquad \text{sign}(\tilde{f}(x)) = \text{sign}(f(x)). \tag{2}$$

Definition. At iteration $n > 1$, we call radius of the interval $R_n = |b_n - a_n|$ where $f(a_n)f(b_n) < 0$.

Initially $x_0 = a$, $\tilde{f}(x_0) = f(a)$, $x_1 = b$, $\tilde{f}(x_1) = f(b)$ are taken. Two controlling parameters are used to limit the iterations: one is the maximum acceptable radius, *atol*, and the other is the maximum acceptable absolute value of the function, *ftol*.

SFRF algorithm, described below, is iterative. In Fig. 1-a the geometric description of a scaled step in the method is shown. The method consists of a sequence of secant and scaled steps (intermixed depending on some condition).

Algorithm 1. SFRF at step $n + 1$ ($n > 0$).
Input: $(x_{n-1}, \tilde{f}(x_{n-1}))$, $(x_n, \tilde{f}(x_n))$

(i) Let put

$$\delta = -\tilde{f}(x_n)\frac{x_{n-1} - x_n}{\tilde{f}(x_{n-1}) - \tilde{f}(x_n)}, \qquad c = x_n + \delta. \tag{3}$$

(ii) (Radius control) If $|x_n - x_{n-1}| < atol$ then c is an acceptable value for the solution; **stop**.

(iii) (Function control) If $|f(c)| < ftol$ then c is an acceptable value for the solution; **stop**.

(iv) (secant step) If $\tilde{f}(x_n)f(c) < 0$ (sign inversion) then **output**

$$(x_{n+1}, \tilde{f}(x_{n+1})) \leftarrow (c, f(c)),$$
$$(x_n, \tilde{f}(x_n)) \leftarrow (x_n, \tilde{f}(x_n)).$$

(v) (scaled step) If $\tilde{f}(x_n)f(c) > 0$, where γ satisfies $0 < \gamma < 1$, then **output**

$$(x_{n+1}, \tilde{f}(x_{n+1})) \leftarrow (c, f(c)),$$
$$(x_n, \tilde{f}(x_n)) \leftarrow (x_{n-1}, \gamma\tilde{f}(x_{n-1})).$$

(vi) At the end of step n:

$$(a_{n+1}, b_{n+1}) \leftarrow (\min(x_n, x_{n+1}), \max(x_n, x_{n+1})).$$

End of algorithm SFRF at step $n+1$.

In [12] it is demonstrated that for any SFRF method, coefficient γ must depend at every step on only two parameters, for example:

$$\xi = \frac{f(c)}{\tilde{f}(x_n)}, \qquad \zeta = -\frac{f(c)}{\tilde{f}(x_{n-1})}, \tag{4}$$

which are always positive. Then we put in a general way:

$$\gamma = F(\xi, \zeta). \tag{5}$$

The way in which we calculate γ defines the specific SFRF method. For example, by putting $\gamma = 1$ we recover the classic RF. Other known methods are the Illinois one:

$$\gamma^{\text{Ill}} = \frac{1}{2}, \tag{6}$$

the Pegasus [11] one:

$$\gamma^{\text{Peg}} = \frac{1}{1+\xi}, \tag{7}$$

and the Anderson and Björck's (AB) [4] one:

$$\gamma^{\text{AB}} = \begin{cases} 1-\xi, & \xi < 1, \\ \frac{1}{2}, & \xi \geq 1. \end{cases} \tag{8}$$

When $\xi \geq 1$ the first expression of AB gives an invalid value for γ, and the authors use as a commitment solution the Illinois method because they discovered that the case $\xi \geq 1$, seldom appears in the iterations and have little interest.

in [13], several SFRF methods from various approximations of an auxiliary slope are developed. The one that interests us the most is the third one, that we name F3. It sometimes gives invalid values when $\xi \geq 1$. Therefore it is convenient to complete F3 in this range, for example:

$$\gamma^{\text{F32}} = \begin{cases} 1 - \frac{\xi}{1+\zeta}, & \xi < 1, \\ \frac{1}{2}, & \xi \geq 1, \end{cases} \tag{9}$$

similarly to AB method.

3. A new SFRF method based on parabolic interpolation

In order to achieve a higher convergence rate than that of the secant method, which uses the last two known points with no derivative calculations, two methodologies are typically used:

1. Keeping at least three points $(x_{n-2}, f(x_{n-2}))$, $(x_{n-1}, f(x_{n-1}))$, and $(x_n, f(x_n))$. This allows the use of approximation functions that depend on three known points.
2. Taking, at each step, some auxiliary points, $y_n, z_n, \ldots$ not included in the main sequence $\{x_n\}$, but used to determine x_{n+1}. This requires the calculations of $f(y_n), f(z_n), \ldots$

The first approach is followed by Muller [20], which uses direct parabolic interpolation to improve the secant method. For simple roots Muller obtained an asymptotic convergence order of 1.8393, positive root of $\lambda^3 - \lambda^2 - \lambda - 1 = 0$. For roots of multiplicity $m = 2$ a convergence order of 1.2337 is obtained. For higher multiplicity the method is always linear.

The second approach is followed, for example, in [1] with one auxiliary point per step, requiring two function calls per step. This fact reduces the computational efficiency (see Section 4), which is not so good.

Since these methods sometimes extrapolate outside the actual interval $[a_n, b_n]$, with $a_n = \min(x_{n-2}, x_{n-1}, x_n)$, $b_n = \max(x_{n-2}, x_{n-1}, x_n)$, they have no global convergence, so they need to hybridise with other methods (usually bisection) to maintain bracketing.

More recently, Brent [6], in turn based on a previous work by Dekker [9], developed a method based on inverse parabolic interpolation that has, near the root, the same convergence properties as Muller's one. It uses hybridisation with bisection when the algorithm detects that the advance towards the root is not fast enough.

Some slight improvements of Brent's procedure have been proposed since then, like the one in [29], but these do not normally appear in the routines of current numerical libraries.

A hyperbolic interpolation gives rise to another method developed in [7], which has the same convergence properties as Muller's one. It also hybridises with bisection if needed.

Nevertheless, we have said that SFRF methods can be used without hybridisation. We will develop such methodologies hereunder.

In order to obtain a suitable function for γ calculation, it is convenient to use a transformed coordinate system. If we name $[x_{n-1}, x_n]$ the actual bracketing interval let us consider the transformation T:

$$(x, y) \xrightarrow{T} (z, w), \qquad z = \frac{x - x_n}{x_{n-1} - x_n}, \qquad w = \frac{y}{\tilde{f}(x_{n-1}) - \tilde{f}(x_n)}. \tag{10}$$

In this case we have:

$$\tilde{f}(x) \xrightarrow{T} g(z) = \frac{\tilde{f}(x(z))}{\tilde{f}(x_{n-1}) - \tilde{f}(x_n)}, \tag{11}$$

If the root of $f(x)$ is in the interval, $\alpha \in (x_{n-1}, x_n)$, then by (10) $0 < \alpha^* < 1$, $\alpha^* = (\alpha - x_n)/(x_{n-1} - x_n)$, being in this case $g(\alpha^*) = f(\alpha) = 0$, so that the new function $g(z)$ also has a root in the interval $(0, 1)$.

With transformations (10) and (11), and the dimensionless parameters defined through (4), we have for abscissas:

$$x_{n-1} \xrightarrow{T} 1, \qquad x_n \xrightarrow{T} 0, \qquad c \xrightarrow{T} z_c = \frac{\zeta}{\xi + \zeta}, \tag{12}$$

and ordinates:

$$\tilde{f}(x_{n-1}) \xrightarrow{T} g_1 = \frac{\xi}{\xi + \zeta}, \quad \tilde{f}(x_n) \xrightarrow{T} g_0 = -\frac{\zeta}{\xi + \zeta}, \quad f(c) \xrightarrow{T} g_c = -\frac{\xi\zeta}{\xi + \zeta}. \tag{13}$$

Two properties are interesting:

$$g_1 = 1 + g_0, \qquad z_c = -g_0. \tag{14}$$

The use of expression (3) for a scaled step requires scaling ordinate at x_{n-1}. In transformed coordinates, we have a new ordinate, γg_1, fulfilling:

$$z_p = z_c - g_c \frac{1 - z_c}{\gamma g_1 - g_c}, \tag{15}$$

which gives us an improved approximation to the root value.

The main idea of PRF method is to generate an adequate expression for z_p by using some superlinear interpolation method. By equating this new expression with (15) and clearing γ we obtain a formula to calculate this scaling factor.

In the following we also use a parabola always interpolating, keeping bracketing automatically, without hybridisation. We need some formula in the form (5). The condition $0 < \gamma < 1$ is fulfilled in all cases.

For determining γ in a scaled step, we begin with the condition $\tilde{f}(x_n) f(c) > 0$ (otherwise a secant step will follow and γ calculation is not required). With this condition (see Fig. 1-a) we establish:

Lemma 1. *Let $q(x)$ be the parabola that passes through points B, C and A of Fig. 1-a. The parabola $p(z)$ transformed from it by (10) and (11) (see Fig. 1-b) is convex between B and A.*

Proof. The parabola that passes through B, C and A, has a constant second derivative of value:

$$p''(z) = 2p[z_c, 0, 1],$$

with $p[z_1, z_2, z_3]$ the second order divided difference. Since $g_1 > 0$, $g_0 < 0$ and $g_c < 0$, we have:

$$p''(z) = \frac{2g_c}{g_0 g_1} > 0,$$

and the parabola is convex between B and A. □

From this Lemma, seeing Fig. 1-b, we can demonstrate the following result.

Lemma 2. *The value of γ in the PRF method, which uses direct parabolic interpolation from abscissa $z = z_p$ such that $p(z_p) = 0$, fulfils $0 < \gamma < 1$.*

Proof. First, since the parabola is a continuous function, by Bolzano's Theorem it has the root $z_p \in (z_c, 1)$, because $g_c < 0$, $g_1 > 0$. Then, $\gamma > 0$ because the limit $\gamma = 0$ corresponds to $z_p = 1$ ($g_1 \neq 0$ and the root is not at this point).

Second, the parabola being convex, the straight line $r(z)$ that passes through C and A, cuts the parabola only at points C and A, with $r(z) - p(z) > 0$, $z \in (z_c, 1)$. The slope s of $r(z)$ is positive since:

$$s = \frac{g_1 - g_c}{1 - z_c},$$

taking into account (14) we have:

$$s = 1 - \frac{g_c}{g_1} > 0,$$

because $g_c < 0$, $g_1 > 0$.

Therefore:

$$z_p - z_d = \frac{r(z_p) - p(z_p)}{s} > 0.$$

This implies $z_p > z_d$, then $\gamma < 1$ because the case $\gamma = 1$ corresponds to $z_p = z_d$, which is not reached. □

At this point we can obtain the form of $F(\xi, \zeta)$ in PRF method.

Theorem 1. *In the step $n + 1$ of a SFRF method, knowing $(x_{n-1}, \tilde{f}(x_{n-1}))$, $(x_n, \tilde{f}(x_n))$, $\tilde{f}(x_n)\tilde{f}(x_{n-1}) < 0$, and $(c, f(c))$, by using (3), with $\tilde{f}(x_n)f(c) > 0$ (a scaled step is the next), direct parabolic interpolation gives:*

$$\gamma^{\mathrm{PRF}} = -\zeta + \frac{1}{2}\left[(1 - \xi + \zeta) + \sqrt{(1 - \xi + \zeta)^2 + 4\xi\zeta}\right], \tag{16}$$

taking the positive square root.

Proof. For the parabolic interpolation it is better to follow the development of Hildebrand [14] rather than of Muller's. We use the transformed variables (the result is the same for the original ones, although the development is more verbose):

$$p(z) = g_k + (z - z_k)g[z_k, z_{k-1}] + (z - z_k)(z - z_{k-1})g[z_k, z_{k-1}, z_{k-2}], \tag{17}$$

in which the first and second order divided differences $g[z_k, z_{k-1}]$ and $g[z_k, z_{k-1}, z_{k-2}]$, respectively, appear. Three auxiliary parameters are used:

$$\begin{aligned} \Omega_k &= (g[z_k, z_{k-1}] + (z_k - z_{k-1})g[z_k, z_{k-1}, z_{k-2}])^{-1}, \\ \lambda_k &= \Omega_k g(z_k), \\ M_k &= 4\Omega_k^2 g(z_k) g[z_k, z_{k-1}, z_{k-2}], \end{aligned}$$

then calculate:

$$z_p = z_c - \frac{2\lambda_k}{1 + \sqrt{1 - M_k}}, \tag{18}$$

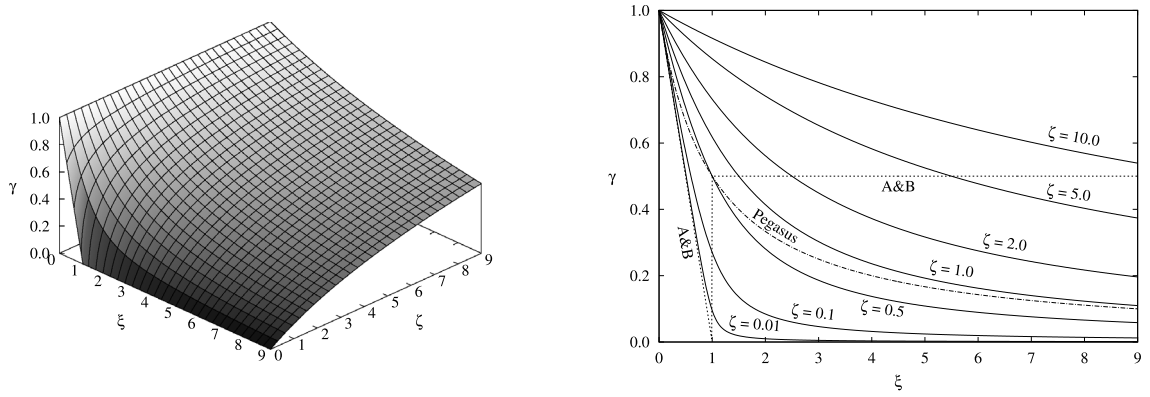


Fig. 2. At left: Graphic representation of the surface $\gamma = F(\xi, \zeta)$, for PRF method by using (16). At right: Diagram of γ vs. ξ for several values of ζ in the PRF method. The curves for AB and Pegasus methods are also shown.

taking the positive root included in (0, 1). In these expressions we use the equivalence:

$$z_{k-2} \equiv 1, \quad z_{k-1} \equiv 0, \quad z_k \equiv z_c, \quad g_{k-2} \equiv g_1, \quad g_{k-1} \equiv g_0, \quad g_k \equiv g_c. \tag{19}$$

(Expressions for Ω_k, λ_k and M_k, as functions of z_k, g_k, etc., are not shown for brevity.) By using (15), taking into account that $g_1 = 1 - z_c$, then:

$$z_p = z_c - \frac{g_c}{\gamma^{\text{PRF}} - \frac{g_c}{g_1}}. \tag{20}$$

By comparing this to (18), we have:

$$\gamma^{\text{PRF}} - \frac{g_c}{g_1} = g_c \frac{1 + \sqrt{1 - M_k}}{2\lambda_k}, \tag{21}$$

then:

$$\gamma^{\text{PRF}} - \frac{g_c}{g_1} = \frac{1}{2}\left[1 - \frac{g_c}{g_0} - \frac{g_c}{g_1}\right]\left[1 + \sqrt{1 - 4\frac{g_c^2}{g_0 g_1}\left[1 - \frac{g_c}{g_0} - \frac{g_c}{g_1}\right]^{-2}}\right], \tag{22}$$

or:

$$\gamma^{\text{PRF}} = \frac{g_c}{g_1} + \frac{1}{2}\left[1 - \frac{g_c}{g_0} - \frac{g_c}{g_1}\right] + \frac{1}{2}\sqrt{\left[1 - \frac{g_c}{g_0} - \frac{g_c}{g_1}\right]^2 - 4\frac{g_c^2}{g_0 g_1}}, \tag{23}$$

and because

$$\frac{g_c}{g_0} = \xi, \qquad \frac{g_c}{g_1} = -\zeta,$$

expression (16) is obtained. □

When iterating several scaled steps in the PRF method the tuple (ξ, ζ, γ) will evolve on the surface $\gamma = F(\xi, \zeta)$, shown in Fig. 2. In the same Fig., at right, the curves corresponding to AB method and Pegasus one are also shown.

We must emphasise that this kind of diagram here presented is valid for all SFRF methods (as defined in [12]) but only for these. Other methods that do not fulfil the collinearity of points B, H and A of Fig. 1 would have similar diagrams but more complex, in which x_{n-2}, x_{n-1} and x_n could also enter as control variables.

The first expression of γ^{AB} in (8) is a particular case of our method (16) for $\zeta = 0$ when $\xi < 1$. Below we will see that in the limit when $\xi \to 0$, AB, F3 and PRF coincide up to $O(\xi)$. Hence, we will use AB and F3 methods for some important demonstrations of the convergence of PRF method.

4. Convergence properties of PRF method

We use the standard notation for a Taylor series around the root, α:

$$f(x) = \sum_{k=1}^{\infty} c_k (x-\alpha)^k, \qquad c_k = \frac{f^{(k)}(\alpha)}{k!},$$

since $f(\alpha) = 0$. We call $e_n = x_n - \alpha$, the error in the n-step. For roots of multiplicity m we have:

$$f_n = f(x_n) = c_m e_n^m + O(e_n^{m+1}), \qquad c_m \neq 0. \tag{24}$$

The convergence order, $r \geq 1$, of a sequence of approximations, x_n, towards the root α is defined [28] by:

$$\lim_{n\to\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^r} = C > 0. \tag{25}$$

For $r = 1$ an additional condition we need is $|C| < 1$. With the previous notation, asymptotically, we have:

$$\lim_{n\to\infty} |e_{n+1}| = C \lim_{n\to\infty} |e_n|^r = 0. \tag{26}$$

On the other hand, the methods for roots calculation need at every iteration step, some specific number of function evaluations, *nfc*. In this regard, Traub [28, appendix C] defines the computational efficiency, p, as:

$$p = r^{1/nfc}, \tag{27}$$

as a measure of the computational effort in order to compare different methods.

4.1. Simple root case

SFRF methods combine the secant one, which is equivalent to a step with $\gamma = 1$, and another scaled with γ calculated by some function, in our case with PRF formula (16). Therefore, it is necessary to analyse both.

Theorem 2 (*Convergence of Secant Method*)**.** *Let $\alpha \in [a, b]$ be a root of $f \in \mathcal{C}^2_{[a,b]}$ and x_{n-2} and x_{n-1} be two successive abscissas belonging to an open domain $U(\alpha, \delta) = \{x \mid |x - \alpha| < \delta\}$ small enough, with their associated errors, e_{n-2} and e_{n-1}. For simple roots the error in the next step n of iteration, in the secant method, expression (3) with $\tilde{f}(x_{n-2}) = f(x_{n-2})$, $\tilde{f}(x_{n-1}) = f(x_{n-1})$ fulfils:*

$$e_n = (c_2/c_1) e_{n-1} e_{n-2} + O(e^3). \tag{28}$$

with $e = \max(|e_{n-2}|, |e_{n-1}|)$.

The demonstration is set out, for example, in [25, pp. 341–342]. For simple roots, iterated secant method has a convergence rate of about 1.6180.

Theorem 3 (*Convergence of Muller's Method*)**.** *Let $\alpha \in [a, b]$ be a root of $f \in \mathcal{C}^3_{[a,b]}$, and x_{n-3}, x_{n-2} and x_{n-1} be three successive abscissas belonging to an open domain $U(\alpha, \delta) = \{x \mid |x - \alpha| < \delta\}$ small enough, with their associated errors, e_{n-3}, e_{n-2} and e_{n-1}. For simple roots, in the next step n of iteration with the Muller's method, the error fulfils:*

$$e_n = -(c_3/c_1) e_{n-1} e_{n-2} e_{n-3} + O(e^4), \tag{29}$$

with $e = \max(|e_{n-3}|, |e_{n-2}|, |e_{n-1}|)$.

The demonstration is set out in [20]. For simple roots, iterated Muller's method has a convergence rate of about 1.8393.

The last two Theorems give a superlinear convergence because $\lim_{n\to\infty} e_n = \lim_{n\to\infty} e_{n-1}^r = 0$, with $r > 1$. Therefore, any combination of a scaled PRF (based on Muller's formula) and secant steps will have superlinear convergence for simple roots, and (26) is fulfilled.

The actual convergence order and computational efficiency will be between one of these two methods, and we will obtain it below, but previously we need some additional properties.

Lemma 3. *In the PRF method, for simple roots,* $\lim_{n\to\infty} \xi_n = 0$.

Proof. From Theorems 2 and 3, for simple roots we have (24) with $m = 1$ and (26), and from the definition of ξ:

$$\lim_{n\to\infty} \xi_n = \lim_{n\to\infty} \frac{f_n}{f_{n-1}} = \lim_{n\to\infty} \frac{e_n}{e_{n-1}} = \lim_{n\to\infty} C|e_{n-1}|^{r-1} = 0,$$

since $r > 1$. □

This property will be used as a discriminant, separating cases with simple roots from the multiple ones.

In order to determine the computational efficiency of the PRF method we will first demonstrate that the method F3 is the limit of our PRF when $\xi \to 0$, therefore, the asymptotic properties in this limit of both methods are the same.

Lemma 4. *For simple roots, in the limit when* $n \to \infty$, *the scaling factors of methods PRF and F3 are the same, that is,* $\lim_{n\to\infty} \gamma^{\mathrm{PRF}} = \gamma^{\mathrm{F3}}$, *therefore they share the same computational efficiency.*

Proof. By developing expression (16) as a Taylor series around $\xi = 0$ we have

$$\gamma^{\mathrm{PRF}} = 1 - \frac{1}{1+\zeta}\xi + O(\xi^2) = \gamma^{\mathrm{F3}} + O(\xi^2), \tag{30}$$

By using Lemma 3:

$$\lim_{n\to\infty} \gamma^{\mathrm{PRF}} = \lim_{\xi\to 0} \gamma^{\mathrm{PRF}} = \gamma^{\mathrm{F3}}. \quad \square \tag{31}$$

Ford [13, pp. 16–18] demonstrated that his method F3 also fulfils expression (29). This allows us to study the sequence of secant and PRF methods in practice. Ford [13, pp. 18–20] points out that, asymptotically, when $c_3/c_1 < 0$ the typical sequence has an F3 step followed by two secant steps with $e_{n+3} \propto e_n^5$ (asymptotically), and therefore the computational efficiency is $p = 5^{1/3} \approx 1.7100$. For $c_3/c_1 > 0$ the typical sequence has two F3 steps followed by two secant ones, with $e_{n+4} \propto e_n^8$ (asymptotically), obtaining $p = 8^{1/4} \approx 1.6818$. Therefore, when $e_n \to 0$, since Lemma 4 showed that both methods are equivalent, these properties apply also to our PRF method.

We can affirm that for simple roots PRF has an asymptotic computational efficiency between 1.6818 and 1.7100 in the normal cases. In general, it lies between 1.6180 and 1.8393. The first number corresponds to iterate the secant method (when a function sign inversion happens at every step), the second one to iterate the parabolic interpolation (when no sign inversion is produced in the step sequence).

4.2. Multiple root case

For multiple roots all methods cited before advance slowly because the function shape near the root does not fit either straight lines, hyperbolas or parabolas well. As Muller [20] demonstrated for multiplicity $m = 2$ the order of convergence of his method is 1.2337. However, for $m = 2$ the requirement of bracketing forces a discontinuity in $f'''(x)$ at $x = \alpha$, so in this case the PRF method degrades to linear convergence.

For $m > 2$ all methods based on polynomial interpolation with a memory of at most two points (as in our case) have linear convergence [28]. For any method of linear convergence be actually convergent

$$e_{n+1} = Ce_n + O(e_n^2), \qquad |C| < 1, \tag{32}$$

must be fulfilled. Actually any SFRF method verifies at every scaled step $|C| < 1$ because $0 < e_{n+1}/e_n < 1$, as established in Lemma 2 of [12].

Lemma 5. *In the PRF method, for roots with finite multiplicity* $m > 1$, *in the limit when* $n \to \infty$, *a sequence of scaled steps fulfils* $\lim_{n\to\infty} \xi_n = C^m$, *with* $0 < |C| < 1$.

Proof. From the definition of ξ we have:

$$\xi_n = \frac{f_n}{f_{n-1}},$$

and by using (24) and (32):

$$\lim_{n\to\infty}\xi_n=\lim_{n\to\infty}\frac{f_n}{f_{n-1}}=\lim_{n\to\infty}\left(\frac{e_n}{e_{n-1}}\right)^m=C^m>0. \quad \square$$

Obviously, the value of C is function-dependent, but combining Lemmas 3 and 5 allows us to distinguish cases with simple roots from cases with multiple ones. Besides, this facilitates the change of the SFRF formula when the values of ξ begin to stall.

For example, we can use the generalised Illinois method (see [12]) with a small value for γ when a multiple root is detected. In [12] an analysis of this method shows that it performs well for multiplicity $m \leq 4$, with a behaviour near the bisection method.

We detach that combining PRF with SFRFm [12], prepared for multiple roots, is not suitable because we do not know previously the multiplicity of the searched root. Therefore, we can begin with the PRF formula, but when the values of ξ stall, not tending to 0, we switch to the generalised Illinois method with a small constant value for γ, because a multiple root is detected. Below we show out the way.

5. Translating PRF into a practical routine

First, a note on performance is convenient. Since our method uses in (16) a square root of a positive argument (then no complex arithmetic involved) some might think that an overload in the calculation time is involved.

However, in modern computers the calculation of a square root in double precision, normally implemented as a specific CPU instruction, is fast (a little slower than floating point division). Therefore, the above-mentioned problem is of little importance.

5.1. Some notes on abscissa tolerance control

Mainly due to the limitations imposed by the floating point representation, this task needs some cautions.

The first one has to do with the abscissa tolerance. This can be expressed both through absolute, *xtol*, and relative (to the value of α), *rtol*, ones. Following [6, pp. 51], a radius control tolerance is used:

$$atol=\max(xtol,4\epsilon)+\max(rtol,4\epsilon)\max(|c|,\epsilon), \tag{33}$$

also protecting the case in which $c=0$. This avoids some trouble with round-off errors.

The second caveat is for avoiding tiny steps towards the root. In some simple roots cases a sequence of scaled steps in the vicinity of α with very small δ values in (3) appears. In this case the interval radius hardly decreases because one extreme is not changing anymore. This can be cured by avoiding these tiny δ, by using an "effective" value:

$$\delta_e=\operatorname{sign}(\delta)\max\left(|\delta|,\frac{1}{2}atol\right). \tag{34}$$

When $|\delta|<\frac{1}{2}atol$, PRF method ensures that $c=x_n+\delta\in(a,b)$, otherwise we have to test if $c=x_n+\delta_e\notin(a_n,b_n)$. When this c lies outside the bracketing interval we take $\alpha=x_n$ (the error is obviously less than *atol*).

A third improvement of the step *(ii) (Radius control)* of Algorithm 1 is possible. After determining c in (3), before we calculate $f(c)$, the radius control can be:

$$\max(|a_n-c|,|b_n-c|)<atol. \tag{35}$$

In this case we can stop the iterations before calculating f, which could be expensive. This is valid for both simple and multiple roots. Nevertheless, we have checked this occurs occasionally and the simple $|a_n-b_n|<atol$ is almost equally effective.

5.2. Strategy to treat multiple roots

From Lemma 5 we know in which conditions the root is perceived as multiple, that is, $m>1$: when successive scaled steps with PRF formula give ξ values differing in a small positive value and they do not tend to zero. We

store the value ξ_{old} of the previous step, and it is compared to the current one. The root is considered multiple if in `MAXFAIL` steps it is:

$$\xi < 1 - E_\xi \quad \text{and} \quad \xi > E_\xi \quad \text{and} \quad |1 - \xi_{\text{old}}/\xi| < E_{r\xi}, \tag{36}$$

both E_ξ and $E_{r\xi}$ being small values (we have chosen 0.01 for both). The first condition avoids problems in cases when the function varies slowly near an interval extreme.

In order to avoid consuming many iterations with little advance towards the root, a small value of `MAXFAIL` is convenient (we use `MAXFAIL` $= 3$).

For the rest of iterations we can use $\gamma = 0.1$ (GIll01), as exposed in [12]. Another option is to switch to bisection method after the values of ξ stall. These two possibilities are programmed in the computer routines in the Supplementary files.

5.3. Description of the algorithm `arlos`

We implement PRF methodology, and we call `arlos` method (from "Advanced Root Locator On Segment") the strategy sets out in two versions. The first, `arlos0`, is a direct application of (16). The second, `arlos1`, differs from the first in which two consecutive secant steps are forbidden, following a modification proposed in [17, p. 424] for the Pegasus method. This gives robustness to the method, improving the convergence in practice. However, the codification of the algorithm is more complex.

Besides, another routine, `arlos2`, after detecting the multiple character of the root, replaces GIll01 in `arlos1` by bisection. Note that the use of bisection after the PRF method might not be called hybridisation, since the two methods are not intermixed in each iteration, but applied one after the other.

The algorithm has as input parameters: the function f, the initial bracketing interval (a, b), the absolute tolerance in abscissas *xtol*, the relative tolerance in abscissas *rtol*, the absolute value of the function to stop the iterations *ftol*, and the maximum accepted number of function calls *nfunmax*.

The algorithm gives as results: the estimation of the root, the total number of function calls used, a message about how convergence is reached, and finally whether the root is simple or not.

In the Supplementary files, full operative implementations of these routines in `python`, `lua`, `C` and `Fortran90` are provided.

6. Numerical experiments

The computational efficiencies cited before have been deduced under iteration conditions near the root. Besides, each function will have different multiplicative coefficients in the convergence rates (they depend on the derivatives of some orders at the root). In practice, every case (function and initial a and b) has enough particularity to produce significant differences as compared with the theoretical convergence rates, because n cannot tend to infinity.

Therefore, we analyse several functions, 50 with simple roots (see Table 1) and 10 with multiple ones of various multiplicity values (see Table 2), many of them chosen from the bibliography. None of the methods cited above are specifically designed to treat the multiple root case, then we have not exposed many examples.

We must point out a note about the functions with even multiplicity of Table 2: to achieve bracketing (and to be able to apply our method) some functions found in the bibliography have been converted from cases without bracketing into another with bracketing by multiplying the original function by $\text{sign}(x - \alpha)$, in every case. It is not common to find these functions in practice, but it is the only option we have to analyse the behaviour of various methods for functions with even multiplicity at the root.

The choice of the test functions will obviously influence the results, and some functions might cause problems to some methods. Nevertheless, the functions in our Tables have been chosen to have a wide range of possibilities, with no intention to benefit any particular method.

All calculations have been conducted using double precision with `python-3.10.6`, `numpy-1.21.5` and, for some routines, `scipy-1.8.0` in an i5-7400 computer at a frequency of 2.7 GHz, with 8 GB of RAM, and Linux Ubuntu 22.04 LTS as operating system.

The methods used for comparison with our `arlos0`, `arlos1` and `arlos2` are: `brentq`, `brenth`, `toms748` and `ridder` (by directly using the specific functions in `scipy.optimize`).

Table 1
Function definitions for simple root cases. For some of them, indicated as "new limits", a and b have been redefined (from the reference) in order to create asymmetric intervals around the root. The column headed with i is a number for function labelling.

i	$f(x)$	a	b	Reference
1	$x^3 - 1$	−0.40	1.50	[2], #1, new limits
2	$11x^{11} - 1$	0.10	1.00	[2], #3
3	$\log x$	0.50	5.00	[32], #1
4	$\arctan x$	−1.00	5.00	[32], #2
5	$x - \exp(\sin x) + 1$	1.00	4.00	[32], #3
6	$x\exp(-x) - 0.1$	0.00	1.00	[32], #4
7	$x^{1/3} - 1$	0.00	5.00	[32], #5
8	$x^2 - \sin^2 x - 1$	−1.00	2.00	[27], #15
9	$3x^2 - 11.12x + 9.1389$	−30.00	2.00	[23], #2
10	$x^6 - 36x^5 + 450x^4 - 2400x^3 + 5400x^2 - 43200x + 720$	10.00	22.00	[23], #4
11	$x^2(x^2/3 + \sqrt{2}\sin x) - \sqrt{3}/18$	0.10	1.00	[2], #2
12	$x^3 + 1$	−1.80	0.00	[2], #4
13	$x^3 - 2x - 5$	0.00	3.00	[21], Group A 18
14	$2x\exp(-5) + 1 - 2\exp(-5x)$	0.00	1.00	[4], Ex. 2 (n = 5)
15	$2x\exp(-10) + 1 - 2\exp(-10x)$	0.00	1.00	[4], Ex. 2 (n = 10)
16	$2x\exp(-20) + 1 - 2\exp(-20x)$	0.00	1.00	[4], Ex. 2 (n = 20)
17	$(1 + (1 - 5^2))x^2 - (1 - 5x)^2$	0.00	1.00	[4], Ex. 3 (n = 5)
18	$(1 + (1 - 10^2))x^2 - (1 - 10x)^2$	0.00	1.00	[4], Ex. 3 (n = 10)
19	$(1 + (1 - 20^2))x^2 - (1 - 20x)^2$	0.00	1.00	[4], Ex. 3 (n = 20)
20	$x^2 - (1 - x)^5$	0.00	1.00	[4], Ex. 4 (n = 5)
21	$x^2 - (1 - x)^{10}$	0.00	1.00	[4], Ex. 4 (n = 10)
22	$x^2 - (1 - x)^{20}$	0.00	1.00	[4], Ex. 4 (n = 20)
23	$(1 + (1 - 5)^4)x - (1 - 5x)^4$	0.00	1.00	[4], Ex. 5 (n = 5)
24	$(1 + (1 - 10)^4)x - (1 - 10x)^4$	0.00	1.00	[4], Ex. 5 (n = 10)
25	$(1 + (1 - 20)^4)x - (1 - 20x)^4$	0.00	1.00	[4], Ex. 5 (n = 20)
26	$(x - 1)\exp(-5x) + x^5$	0.00	1.00	[4], Ex. 6 (n = 5)
27	$(x - 1)\exp(-10x) + x^{10}$	0.00	1.00	[4], Ex. 6 (n = 10)
28	$(x - 1)\exp(-20x) + x^{20}$	0.00	1.00	[4], Ex. 6 (n = 20)
29	$x^2 + \sin(x/5) - 1/4$	0.00	1.00	[2], #10 (n = 5)
30	$x^2 + \sin(x/10) - 1/4$	0.00	1.00	[2], #10 (n = 10)
31	$x^2 + \sin(x/20) - 1/4$	0.00	1.00	[2], #10 (n = 20)
32	$\sin x - x^3 - 1$	−2.00	−1.00	[30], p. 118, ex. 3a
33	$x - \log x - 3$	2.00	6.00	[30], p. 118, ex. 3b
34	$(x - 1)(x - 2)(x - 3)(x - 4)(x - 5)(x - 6)$	3.10	4.50	[21], Group A 1d
35	$\sin x$	1.00	6.00	[21], Group A 2a
36	$(x^2 + 1)\sin x - \exp(\sqrt{\lvert x\rvert})(x - 1)(x^2 - 5)$	0.00	1.00	[21], Group A 3
37	$\frac{x+1}{x^2+2}$	−2.30	0.50	[21], Group A 4, new limits
38	$x^2 - 1$	−1.50	0.00	[21], Group A 5a
39	$x^9 + x$	−0.75	0.50	[21], Group B-II 1c
40	$x^{19} + x$	−0.75	0.50	[21], Group B-II 1d
41	$x^5 + x + 0.0001$	−0.75	0.50	[21], Group B-II 3b
42	$4\cos(x) - \exp(x)$	−1.00	3.00	[13], #1 range 2
43	$\sum_{i=1}^{10}\{\exp(xt_i) - \exp(5t_i)\}$, where $t_i = 0.1i$	4.00	6.50	[13], #2 range 1
44	$10^{10}x^{1/x} - 1$	0.08	0.50	[13], #6 range 3
45	$\sqrt{x} - 3 - 1/x$	5.00	30.00	[8], #7
46	$(15x - 1)/(14x)$	0.01	1.00	[4], Ex. 7 (n = 15)
47	$(20x - 1)/(19x)$	0.01	1.00	[4], Ex. 7 (n = 20)
48	$x^{1/5} - 5^{1/5}$	1.00	100.00	[3], #12 (n = 5)
49	$x^{1/10} - 10^{1/10}$	1.00	100.00	[3], #12 (n = 10)
50	$x^{1/20} - 20^{1/20}$	1.00	100.00	[3], #12 (n = 20)

In the Tables with values of the total function calls, *nfun*, obtained by different methods, we take $ftol = 10^{-100}$. Also we take the smaller value for the relative tolerance, $rtol = 4\epsilon$.

We treat several functions with different bracketing intervals. For avoiding the effect of the initial radius, we chose *xtol* proportional to the initial $|b - a|$, with two different cases.

Table 2
Function definitions for multiple root cases. Some functions have been modified from the original, multiplying by $\text{sign}(x-\alpha)$. The multiplicity is m. The column headed with i is a number for function labelling.

i	$f(x)$	m	a	b	reference
51	$(\log x)^2 \,\text{sign}(x-1)$	2	0.50	5.00	[12]
52	$(x^2\exp(x)-\sin(x)+x)\,\text{sign}(x)$	2	−0.20	5.00	[15], f_3 modified
53	x^3	3	−0.50	1/3	[27], #17
54	$\left[\arctan\frac{\sqrt{5}}{2}-\arctan\sqrt{x^2-1}+\sqrt{6}\left(\arctan\sqrt{\frac{x^2-1}{6}}-\arctan\left(\frac{1}{2}\sqrt{\frac{5}{6}}\right)\right)-\frac{11}{63}\right]^3$	3	1.50	2.00	[26], ex. 3, modified
55	$x^2\sin^2 x\,\text{sign}(x)$	4	−2.00	1.00	[12]
56	$\text{sign}(x-2)(x-2)^4/((x-1)^2+1)$	4	1.50	2.40	[24], ex. 1, modified
57	x^5	5	−0.50	1/3	[27], #18
58	$(\exp(-x)-1+x/5)^5$	5	4.00	5.20	[5], ex. 4, modified
59	$x^3\sin^3 x\,\text{sign}(x)$	6	−1.00	0.50	[12]
60	$\text{sign}(x-2)(x-2)^6/((x-1)^2+1)$	6	1.90	2.20	[24], ex. 1, modified

In order to try avoiding rounding issues, a commitment value for $xtol/|b-a|$ is 2×10^{-14} (that is, 100 times the machine epsilon in double precision in `python`). This small value is convenient for numerical convergence studies (see results in Tables 3 and 5).

Nevertheless, in engineering and scientific practice some greater values are sometimes used. We also take $xtol/|b-a| = 0.5\times10^{-6}$, that is, 6 significant figures of improvement compared to the initial radius (see results in Tables 4 and 6).

6.1. Simple root cases

Results of the total function calls, *nfun*, for simple root cases are presented in Tables 3 and 4. We show in boldface the best results for each function. Many times several methods have the same *nfun* value.

The average of *nfun* is usually used for comparing among different methods: the lower this value the better. In our case we have ordered by goodness: `arlos1` (and `arlos2`), `arlos0`, `brenth`, `brentq`, `toms748` and `ridder`.

We begin with the analysis of results in Table 3, for $xtol/|b-a|$ is 2×10^{-14}. For simple roots, the new methods have been beaten clearly (difference in *nfun* greater than 1) only for functions with hyperbolic form (*#46, #47*) by `brenth`, and others occasionally (*#2, #27*) by `brentq`, and *#2* by `brenth`, and *#7* by `toms748`. In the rest of cases our methods either tie or win.

The standard deviation values for *nfun*, also shown in Table 3, approximately show the robustness of each method: a small value indicating very robust, and a large one, little robust. With this measure, we classify: `brenth`, `arlos1` (and `arlos2`), `brentq`, `arlos0`, `ridder` and `toms748`, in that order, with no much difference. Therefore, in this regard all analysed methods behave almost the same.

The similar values of the iterations average number of *nfun* for all methods, except `ridder`, are a byproduct of the superlinear interpolation formulas applied in the methods. The small differences among them are, in part, probably due to the initial iterations not close to the root. The `ridder` method uses an exponential approximation, and it performs worse in all cases.

Although the use of the average is customary in comparing among numerical methods, in [19] a better methodology is exposed, by defining a benchmark in terms of a set $\mathcal{P}$ of problems, a set $\mathcal{S}$ of solvers, and a convergence test $\mathcal{T}$. We use a performance measure $nfun_{p,s} > 0$ (in our case the number of function evaluations required to satisfy the desired convergence) obtained for each $p\in\mathcal{P}$, $s\in\mathcal{S}$. For any tuple (p,s) the performance ratio is defined by:

$$r_{p,s} = \frac{nfun_{p,s}}{\min\{nfun_{p,s} \mid s\in\mathcal{S}\}}. \tag{37}$$

The performance profile of a solver $s\in\mathcal{S}$ is the probability distribution for the ratio $r_{p,s}$. It is the fraction of problems where the performance ratio is at most ω, that is:

$$\rho_s(\omega) = \frac{\text{size}\{p\in\mathcal{P} \mid r_{p,s}\le\omega\}}{\text{size}\{p\in\mathcal{P}\}}. \tag{38}$$

Table 3
Results of the numerical experiments for simple roots by different methods. $ftol = 10^{-100}$, $xtol = 2\times10^{-14}|a-b|$, $rtol = 4\epsilon$. Bisection method needs 48 function calls in every case.

i	ridder	toms748	brentq	brenth	arlos0	arlos1	arlos2
1	16	12	11	**10**	11	**10**	**10**
2	14	17	12	**11**	15	14	14
3	14	10	10	9	9	**8**	**8**
4	20	11	**9**	**9**	**9**	10	10
5	16	13	13	13	12	**10**	**10**
6	14	**9**	**9**	**9**	**9**	**9**	**9**
7	14	**7**	10	11	11	11	11
8	18	**11**	**11**	**11**	12	**11**	**11**
9	24	15	19	19	**5**	**5**	**5**
10	14	18	13	13	14	**12**	**12**
11	16	**10**	13	12	12	11	11
12	12	11	**10**	**10**	11	**10**	**10**
13	16	**11**	**11**	**11**	**11**	**11**	**11**
14	12	11	**10**	**10**	**10**	**10**	**10**
15	12	11	12	12	**10**	**10**	**10**
16	12	12	12	12	12	**11**	**11**
17	16	9	10	10	**5**	**5**	**5**
18	16	8	10	9	**5**	**5**	**5**
19	16	8	9	9	**5**	**5**	**5**
20	14	11	**9**	10	10	10	10
21	16	12	**10**	11	11	11	11
22	16	13	13	**12**	13	13	13
23	12	**8**	**8**	**8**	9	9	9
24	12	11	**7**	**7**	8	8	8
25	14	**7**	**7**	**7**	**7**	8	8
26	14	12	**9**	**9**	**9**	**9**	**9**
27	16	13	**9**	11	11	11	11
28	16	17	13	14	**12**	13	13
29	12	10	11	10	8	**7**	**7**
30	12	10	10	10	8	**7**	**7**
31	12	10	10	10	8	**6**	**6**
32	14	10	9	9	**8**	**8**	**8**
33	12	10	8	**7**	**7**	8	8
34	14	12	**10**	**10**	**10**	**10**	**10**
35	16	13	**10**	**10**	**10**	**10**	**10**
36	16	**9**	**9**	**9**	**9**	**9**	**9**
37	12	10	11	11	10	**9**	**9**
38	14	8	10	9	**4**	**4**	**4**
39	**8**	**8**	**8**	**8**	**8**	**8**	**8**
40	8	**7**	**7**	**7**	**7**	**7**	**7**
41	10	9	9	9	**8**	**8**	**8**
42	12	**10**	12	11	11	**10**	**10**
43	12	10	10	**9**	10	**9**	**9**
44	20	20	20	20	19	**17**	**17**
45	12	**8**	**8**	9	**8**	**8**	**8**
46	18	18	13	**8**	12	12	12
47	18	18	14	**9**	12	13	13
48	14	**10**	11	11	**10**	**10**	**10**
49	14	11	11	11	**10**	**10**	**10**
50	14	11	11	**10**	**10**	**10**	**10**
Mean	14.3	11.2	10.6	10.3	9.7	**9.4**	**9.4**
Std dev	2.9	3.0	2.5	**2.4**	2.7	2.5	2.5

It is a non-decreasing, piecewise constant function, continuous from the right at each breakpoint. The best method at any value of ω is the one with higher ρ_s. The value of $\rho_s(1)$ is the probability that the solver will win over the

Table 4
Results of the numerical experiments for simple roots by different methods. $ftol = 10^{-100}$, $xtol = 0.5 \times 10^{-6}|a-b|$, $rtol = 4\epsilon$. Bisection method needs 23 function calls in every case.

i	ridder	toms748	brentq	brenth	arlos0	arlos1	arlos2
1	12	11	10	**9**	10	**9**	**9**
2	12	13	**10**	**10**	13	12	12
3	12	9	9	**8**	**8**	**8**	**8**
4	14	9	**8**	**8**	**8**	**8**	**8**
5	12	11	12	12	10	**9**	**9**
6	10	9	**8**	**8**	**8**	**8**	**8**
7	12	**7**	9	9	9	9	9
8	12	**9**	**9**	**9**	10	10	10
9	20	13	18	17	**5**	**5**	**5**
10	**10**	14	12	11	12	11	11
11	12	**10**	12	11	**10**	**10**	**10**
12	10	10	**9**	**9**	**9**	**9**	**9**
13	12	11	10	**9**	10	**9**	**9**
14	10	11	**9**	**9**	**9**	**9**	**9**
15	12	11	10	10	10	**9**	**9**
16	12	11	11	11	**10**	**10**	**10**
17	12	7	9	8	**5**	**5**	**5**
18	12	7	8	8	**5**	**5**	**5**
19	12	7	8	7	**5**	**5**	**5**
20	12	9	**8**	**8**	9	9	9
21	12	10	**9**	**9**	10	10	10
22	14	12	12	**11**	12	**11**	**11**
23	10	**7**	**7**	**7**	**7**	**7**	**7**
24	10	7	**6**	**6**	7	7	7
25	10	7	**6**	**6**	7	**6**	**6**
26	12	10	**8**	**8**	**8**	**8**	**8**
27	14	11	**8**	10	9	9	9
28	14	13	**11**	12	**11**	**11**	**11**
29	10	9	9	9	8	**7**	**7**
30	10	9	10	9	**7**	**7**	**7**
31	10	9	9	9	**6**	**6**	**6**
32	10	9	**8**	**8**	**8**	**8**	**8**
33	10	**7**	**7**	**7**	**7**	**7**	**7**
34	12	11	**9**	**9**	**9**	10	10
35	12	11	**9**	**9**	**9**	**9**	**9**
36	12	9	**7**	**7**	**7**	**7**	**7**
37	10	9	10	10	9	**7**	**7**
38	10	8	9	8	**4**	**4**	**4**
39	8	**7**	**7**	**7**	**7**	**7**	**7**
40	8	7	**6**	**6**	**6**	**6**	**6**
41	10	9	**7**	8	**7**	**7**	**7**
42	10	**9**	10	10	10	**9**	**9**
43	10	10	9	**8**	9	**8**	**8**
44	18	16	18	18	17	**15**	**15**
45	10	**7**	**7**	8	8	8	8
46	14	17	11	**8**	11	11	11
47	16	16	13	**9**	11	11	11
48	12	**9**	10	10	**9**	**9**	**9**
49	12	10	10	10	**9**	**9**	**9**
50	12	10	10	**9**	10	**9**	**9**
Mean	11.7	9.9	9.4	9.1	8.7	**8.4**	**8.4**
Std dev	2.2	2.4	2.4	2.2	2.3	**2.0**	**2.0**

rest of the solvers. Then, if we are interested solely in the number of wins, we need only to compare the values of $\rho_s(1)$ for all solvers [10].

Table 5
Results of the numerical experiments for multiple roots by different methods. $ftol = 10^{-100}$, $xtol = 2 \times 10^{-14}|a-b|$, $rtol = 4\epsilon$. Bisection method needs 48 function calls in every case.

i	ridder	toms748	brentq	brenth	arlos0	arlos1	arlos2
51	66	89	67	67	**40**	**40**	53
52	80	94	123	126	**41**	42	58
53	78	99	117	117	**46**	**46**	56
54	80	71	118	134	**43**	**43**	55
55	68	112	118	120	**51**	**51**	57
56	68	108	115	113	**53**	**53**	56
57	72	118	117	112	64	64	**56**
58	74	109	115	121	63	63	**57**
59	84	126	139	138	77	77	**56**
60	78	116	144	143	77	76	**55**
Mean	75	104	117	119	**56**	**56**	**56**
Std dev	5.8	15	19	20	13	13	**1.3**

Table 6
Results of the numerical experiments for multiple roots by different methods. $ftol = 10^{-100}$, $xtol = 0.5 \times 10^{-6}|a-b|$, $rtol = 4\epsilon$. Bisection method needs 23 function calls in every case.

i	ridder	toms748	brentq	brenth	arlos0	arlos1	arlos2
51	30	51	30	32	**20**	22	28
52	38	59	51	52	**22**	**22**	33
53	38	49	54	56	**25**	**25**	32
54	40	49	53	61	**25**	**25**	31
55	32	49	57	51	**29**	**29**	33
56	34	54	52	55	**28**	**28**	32
57	36	52	50	37	35	35	**32**
58	36	54	56	55	34	34	**32**
59	36	60	60	58	40	40	**31**
60	38	60	66	67	40	40	**31**
Mean	36	54	53	52	**30**	**30**	31
Std dev	2.9	4.3	8.9	10	6.8	6.5	**1.4**

An important property of performance profiles is that they are insensitive to large changes in the results of a few problems of the set. Moreover, they are also largely unaffected by small changes in results over many problems [10].

In Fig. 3 the performance profiles for some root finders for the simple root cases included in Table 1, calculated from data in the corresponding columns of Table 3, are exposed. Obviously other set of functions, initial bracketing intervals and specific tolerances would give somewhat different results.

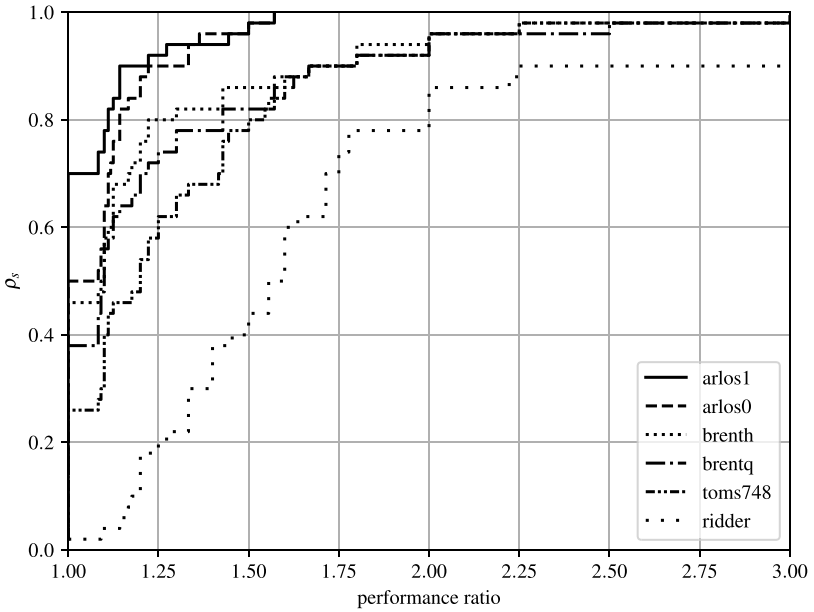
In regard to $\rho_s(1)$, the values shown at right in Fig. 3-a are very clear: `arlos1` (and `arlos2`) is the best, followed by `arlos0`. The rest of the methods are worse. As we can see `arlos1` and `arlos2` have a probability of 70% of beating the rest of the methods, and they will be the normal root finding choice.

For $xtol/|b-a|$ is 0.5×10^{-6} the results are shown in Table 4. We see only an improvement of about 11% for all methods, except for `ridder` (18%). This small value shows the superlinear character of the methods: only an increase of 1 in *nfun* is usually necessary for passing from 0.5×10^{-6} to 2×10^{-14}, nearly doubling the significant figures. We have to detach that the actual errors of the returned approximations to the root (not shown) are often much smaller than the requested tolerance.

Analysing the robustness of the methods indicated by the standard deviation of *nfun*, `arlos1` and `arlos2` are now the best, and the results are now slightly lower than the previous ones.

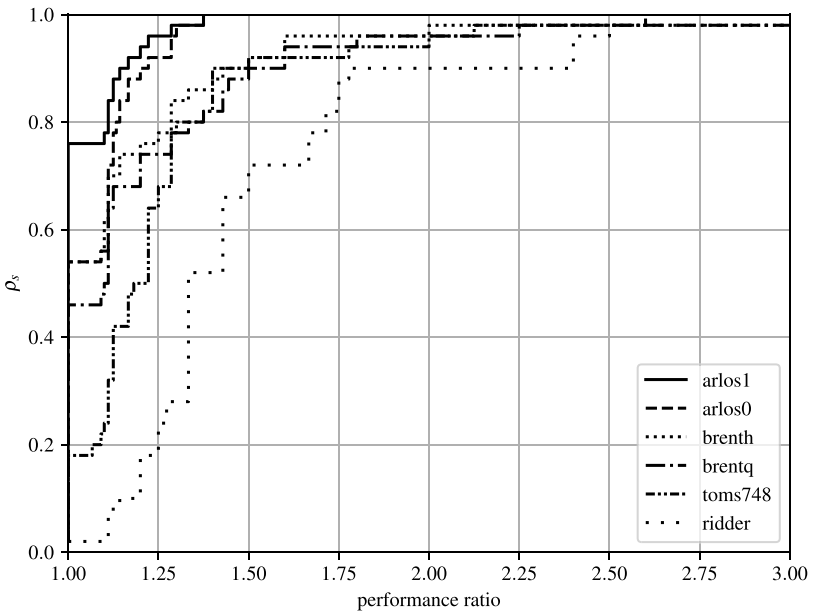
In regard to $\rho_s(1)$, the values exposed at right in Fig. 3-b show that, as before, `arlos1` and `arlos2` are the best, even improving a little the results (now a probability of 76%). Both parts of Fig. 3 show similar graphs, which indicates that the conclusions are little dependent on the requested tolerance.

(a) Simple roots, $xtol/|b-a| = 2.0 \times 10^{-14}$.



method	$\rho_s(1)$ percent
arlos1	70
arlos0	50
brenth	46
brentq	38
toms748	26
ridder	2

(b) Simple roots, $xtol/|b-a| = 0.5 \times 10^{-6}$.



method	$\rho_s(1)$ percent
arlos1	76
arlos0	54
brenth	54
brentq	46
toms748	18
ridder	2

Fig. 3. Cumulative distribution for the performance profile of simple root cases for `ridder`, `toms748`, `brentq`, `brenth`, `arlos0` and `arlos1`. `arlos2` have the same behaviour as `arlos1` for simple roots. The values at performance ratio equal to 1 is shown at right for better understanding. (a) for results in Table 3. (b) for results in Table 4.

6.2. Multiple root cases

The analysis of multiple root cases presented in Tables 5 and 6 is not deep, because none of the routines here developed (nor the ones in `scipy.optimize`) are specifically designed for it. Hence, we present results only as a sample, and the conclusions about the behaviour of our methods for multiple roots are approximate.

For multiple roots the results show that, even in this difficult case, `arlos0` and `arlos1`, not specifically designed for it, perform better than the standard routines in `scipy.optimize`, at least when $m \leq 6$. For $m \leq 4$ it appears that `arlos0` or `arlos1` would be the choice, but `arlos2` would be the best choice for unknown multiplicity: very good for simple roots and reliable for multiple ones. On average only 8 function calls more than bisection are needed in our examples both for $xtol = 2 \times 10^{-14}|a - b|$ and $xtol = 0.5 \times 10^{-6}|a - b|$.

Actually, `arlos2` consumes some iterations detecting the character of the root, with no much radius reduction. When it detects the root is multiple it switches to bisection in a reduced interval (compared to the initial one). Therefore, *nfun* becomes approximately non-dependent on multiplicity when $m > 1$ for `arlos2`.

A comment on `ridder` method: for simple roots it is the worst of the methods in `scipy.optimize`, but it is the best for multiple roots (apart from bisection). In any case the new method `arlos2` is better that `ridder` for multiple roots.

The robustness of these methods through the standard deviation is in the following order: Bisection (not shown) has a value or 0 for it; `arlos2` is the next with a value about 1.4, followed by `ridder` and the rest of methods with a similar worse behaviour.

The effect of the multiplicity in the results deserves a comment. `arlos2` is not much affected, but `arlos0` and `arlos1` methods have increasing *nfun* values as multiplicity increases. For $m \geq 6$ (results not shown) it seems more economic to use `arlos2`. Obviously, more studies have to be done for treating better multiple root cases, with the aim of improving the routines `arlos`.

7. Conclusions

We have developed a new SFRF method based on direct parabolic interpolation (named PRF). Global convergence is ensured because the method maintains bracketing, as all SFRF methods.

The developed methodology detects whether the root is simple or not, with no additional work, by analysing the successive values of a parameter calculated at every step in the iteration. This is an important feature not found normally in the routines appearing in the current numerical libraries.

For simple roots it has a high computational efficiency, between 1.6818 and 1.7100 in the typical cases. In the worst case this is 1.6180, corresponding to iterations of the secant method, and in the best case this is 1.8393, corresponding to iterations of the parabolic interpolation.

For multiple roots it has linear convergence, but the above-mentioned determination of root character can be used to switch as desired to other SFRF methods. This allows to improve the performance in this difficult case when double precision is used, and not multiprecision. Effectively, by combining the PRF with the generalised Illinois method (with a constant scaling factor 0.1) we have designed an algorithm (`arlos`) that keeps the advantages of both methods (each in their own "environment"). Another strategy is passing to bisection after detecting the multiple character of the root.

Finally, we have analysed many functions, both with simple and multiple roots, by using several methods. For simple roots our method `arlos` (in three versions) is equal or better than the hybrid methods appearing in the currently used numerical libraries, while it is better for multiple roots up to multiplicity six, on average only a little worse than bisection.

For all these reasons, these new routines can be used, with advantage, as a substitute for those considered as state-of-the-art, such as `brenth`, `brentq`, `toms748` and others.

CRediT authorship contribution statement

Julio M. Fernández-Díaz: Conceptualization, Formal analysis, Methodology, Writing – original draft, Computer programming. **César O. Menéndez-Pérez:** Conceptualization, Formal analysis, Methodology, Writing – original draft.

Acknowledgements

The authors would like to thank the anonymous reviewers for their help to improve the final version of this manuscript.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.matcom.2023.08.003.

References

[1] G.E. Alefeld, F.A. Potra, On two higher order enclosing methods of J.W. Schmidt, ZAMM Z. Angew. Math. Mech. 68 (1988) 331–337, http://dx.doi.org/10.1002/zamm.19880680802.
[2] G.E. Alefeld, F.A. Potra, Some efficient methods for enclosing simple zeros of nonlinear equations, BIT 32 (1992) 334–344, http://dx.doi.org/10.1007/BF01994885.
[3] G.E. Alefeld, F.A. Potra, Y. Shi, Algorithm 748: Enclosing zeros of continuous functions, ACM Trans. Math. Software 221 (1995) http://dx.doi.org/10.1145/210089.210111.
[4] N. Anderson, Å. Björck, A new high order method of regula falsi type for computing a root of an equation, BIT 13 (1973) 253–264, http://dx.doi.org/10.1007/BF01951936.
[5] R. Behl, A. Cordero, J.R. Torregrosa, A new higher-order optimal derivative free scheme for multiple roots, J. Comput. Appl. Math. 404 (3) (2022) 113773, http://dx.doi.org/10.1016/j.cam.2021.113773.
[6] R.P. Brent, Algorithms for Minimization Without Derivatives, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.
[7] J.C.P. Bus, T.J. Dekker, Two efficient algorithms with guaranteed convergence for finding a zero of a function, ACM Trans. Math. Software 1 (1975) 330–345, http://dx.doi.org/10.1145/355656.355659.
[8] C. Chun, M.Y. Lee, A new optimal eighth-order family of iterative methods for the solution of nonlinear equations, Appl. Math. Comput. 223 (2013) 506–519, http://dx.doi.org/10.1016/j.amc.2013.08.033.
[9] T.J. Dekker, Finding a zero by means of successive linear interpolation, in: B. Dejon, P. Henrici (Eds.), Constructive Aspects of the Fundamental Theorem of Algebra, Wiley-Interscience, London, 1969, pp. 37–48.
[10] E.D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2002) 201–213, http://dx.doi.org/10.1007/s101070100263.
[11] M. Dowell, P. Jarratt, The Pegasus method for computing the root of an equation, BIT 12 (1972) 503–508, http://dx.doi.org/10.1007/BF01932959.
[12] J.M. Fernández-Díaz, C.O. Menéndez-Pérez, A common framework for modified Regula Falsi methods and new methods of this kind, Math. Comput. Simulation 205 (2023) 678–696, http://dx.doi.org/10.1016/j.matcom.2022.10.019.
[13] J.A. Ford, Improved Algorithms of Illinois-Type for the Numerical Solution of Nonlinear Equations, Technical Report CSM-257, University of Essex, 1995.
[14] F.B. Hildebrand, Introduction to Numerical Analysis, second ed., Dover Pub., Inc., New York, 1987.
[15] J. Hueso, E. Martínez, C. Teruel, Determination of multiple roots of nonlinear equations and applications, J. Math. Chem. 53 (2015) 880–892, http://dx.doi.org/10.1007/s10910-014-0460-8.
[16] IEEE Computer Society, IEEE Standard for Floating-Point Arithmetic, IEEE STD 754-2019, IEEE, ISBN: 978-1-5044-5924-2, 2019, http://dx.doi.org/10.1109/IEEESTD.2019.8766229.
[17] R.F. King, An improved Pegasus method for root finding, BIT 13 (1973) 423–427, http://dx.doi.org/10.1007/BF01933405.
[18] R.F. King, A secant method for multiple roots, BIT 17 (1977) 321–328, http://dx.doi.org/10.1007/BF01932152.
[19] J.J. Moré, S.M. Wild, Benchmarking derivative-free optimization algorithms, SIAM J. Optim. 20 (1) (2009) 172–191, http://dx.doi.org/10.1137/080724083.
[20] D.E. Muller, A method for solving algebraic equations using an automatic computer, MTAC 10 (1956) 208–215, http://dx.doi.org/10.2307/2001916.
[21] D. Nerinckx, A. Haegemans, A comparison of non-linear equation solvers, J. Comput. Appl. Math. 2 (2) (1976) 145–148, http://dx.doi.org/10.1016/0771-050X(76)90017-6.
[22] J.M. Papakonstantinou, R.A. Tapia, Origin and evolution of the secant method in one dimension, Amer. Math. Monthly 120 (2013) 500–518, http://dx.doi.org/10.4169/amer.math.monthly.120.06.500.
[23] P.K. Parida, D.K. Gupta, An improved regula-falsi method for enclosing simple zeros of nonlinear equations, Appl. Math. Comput. 177 (2006) 769–776, http://dx.doi.org/10.1016/j.amc.2005.11.034.
[24] P.K. Parida, D.K. Gupta, An improved method for finding multiple roots and it's multiplicity of nonlinear equations in $\mathbb{R}$, Appl. Math. Comput. 202 (2008) 498–503, http://dx.doi.org/10.1016/j.amc.2008.02.030.
[25] A. Ralston, P. Rabinowitz, A First Course in Numerical Analysis, second ed., Dover Pub., Inc., New York, 2001.
[26] J.R. Sharma, S. Kumar, I.K. Argyros, Development of optimal eighth order derivative-free methods for multiple roots of nonlinear equations, Symmetry 11 (6) (2019) http://dx.doi.org/10.3390/sym11060766.
[27] A. Suhadolnik, Combined bracketing methods for solving nonlinear equations, Appl. Math. Lett. 25 (2012) 1755–1760, http://dx.doi.org/10.1016/j.aml.2012.02.006.
[28] J.F. Traub, Iterative Methods for the Solution of Equations, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1964.

[29] G. Wilkins, M. Gu, A modified Brent's method for finding zeros of functions, Numer. Math. 123 (2013) 177–188, http://dx.doi.org/10.1007/s00211-012-0480-x.
[30] D.M. Young, R.T. Gregory, A Survey of Numerical Mathematics, Dover Pub., Inc., New York, 1988.
[31] B.I. Yun, A derivative free iterative method for finding multiple roots of nonlinear equations, Appl. Math. Lett. 22 (2009) 1859–1863, http://dx.doi.org/10.1016/j.aml.2009.07.013.
[32] Y. Zhu, X. Wu, A free-derivative iteration method of order three having convergence of both point and interval for nonlinear equations, Appl. Math. Comput. 137 (2003) 49–55, http://dx.doi.org/10.1016/S0096-3003(02)00029-2.