Mathematical Sciences (2019) 13:1–20
https://doi.org/10.1007/s40096-018-0272-2

ORIGINAL RESEARCH

A new family of adaptive methods with memory for solving nonlinear equations

Vali Torkashvand[1] · Taher Lotfi[1] · Mohammad Ali Fariborzi Araghi[2]

Received: 7 March 2018 / Accepted: 19 November 2018 / Published online: 11 December 2018

Abstract
In this work, an adaptive method with memory is developed such that all previous information are applied. The importance of the proposed method can be seen because of the optimization in important effecting factors, i.e., least number of iterations steps, least number of functional evaluations, least value of absolute error, and maximum efficiency index in final as well as in individual step as compared with the other methods. Indeed, it is proved that this adaptive method with memory has efficiency index 2 and competes all the existing methods without and with memory in the literature. The order of convergence is obtained by using two self-accelerating parameters, which is increased from 2 to 4 without any new function evaluation. It means that, the order of convergence can be improved until 100%. Numerical examples and the comparison with existing methods are included to demonstrate exceptional convergence speed of the proposed method and confirm theoretical results.

Keywords Newton's interpolatory polynomial · Adaptive method with memory · Self-accelerator · Nonlinear equation

Mathematics Subject Classification 65H05 · 65B99

Introduction and preliminaries

Many of the complex problems in science and engineering contain the function of nonlinear and transcendental nature in the equation of the form $f(x)=0$. Numerical iterative schemes like Newton's method [42] are often used to obtain the approximate solution of such problems because it is not always possible to obtain its exact solution by usual algebraic process. However, the condition $f'(x)\neq 0$ in a neighborhood of the required root is severe indeed for convergence of Newton's method, which restricts its applications in practice. To overcome on this difficulty, Steffensen replaced the first derivative of the function in the Newton's iterate by forward finite difference approximation [58]. Traub in his book classified iterative methods for solving such equations as one point or multipoints [61]. We classify the iterative formulas by information they need as follows [61]:

1. *One-point iterative method without memory* In this type of methods, x_{k+1} can be determined by only new data at x_k. No previous information is reused.
   Thus, $x_{k+1}=\phi(x_k)$. Then ϕ will be called a one-point iterative formula (I.F.).
   The most commonly known example is Newton's I.F. (iterative formula) [42]:
$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)},\quad k=0,1,\ldots, \tag{1}$$
   and free derivative Steffensen's [58] :
$$\begin{cases} w_k=x_k+\beta f(x_k), & k=0,1,2,\ldots, \\ x_{k+1}=x_k-\frac{f(x_k)}{f[x_k,w_k]}. \end{cases} \tag{2}$$
2. *One-point iterative method with memory* In this category x_{k+1} can be determined by new information at x_k and reused information at $x_{k-1},\ldots,x_{k-n}$. Thus, $x_{k+1}=\phi(x_k;x_{k-1},\ldots,x_{k-n})$. Then ϕ will be called a one-

✉ Taher Lotfi
lotfi@iauh.ac.ir; lotfitaher@yahoo.com

Vali Torkashvand
torkashvand1978@gmail.com

Mohammad Ali Fariborzi Araghi
fariborzi.araghi@gmail.com

1 Faculty of Basic Science, Islamic Azad University, Hamedan Branch, Hamedan, Iran

2 Department of Mathematics, Central Tehran branch, Islamic Azad University, Tehran, Iran

point I.F. with memory. The best-known examples of a one-point I.F. with memory are the secant I.F. [47]

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} f(x_k), \quad k = 1, 2, \dots, \tag{3}$$

and Traub's method [61]

$$\begin{cases} \lambda_k = \frac{-1}{f[x_k, x_{k-1}]}, \quad k = 1, 2, \dots, \\ w_k = x_k + \lambda_k f(x_k),\ x_{k+1} = x_k - \frac{f(x_k)}{f[x_k, w_k]}, \quad k = 0, 1, \dots. \end{cases} \tag{4}$$

3. *Multipoint iterative method without memory* In this type of methods x_{k+1} can be determined by new at $x_k, w_1(x_k), \dots, w_n(x_k), n \geq 1$. No old information is reused. Thus $x_{k+1} = \phi[x_k, \dots, x_{k-n}]$. Hence, $x_{k+1} = \phi[x_k; w_1(x_k), \dots, w_n(x_k)]$. In this case, ϕ will be called a multipoint I.F. Pioneers in the field: Ostrowski's [43]

$$\begin{cases} y_k = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \dots, \\ x_{k+1} = y_k - \frac{f(x_k)}{f'(x_k)} \frac{f(y_k)}{f(x_k) - 2f(y_k)}, \end{cases} \tag{5}$$

and Jarratt [25]

$$\begin{cases} y_k = x_k - \frac{2}{3} \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \dots, \\ x_{k+1} = x_k - \frac{1}{2} \frac{f(x_k)}{f'(x_k) - 3f'(y_k)}, \end{cases} \tag{6}$$

also, Neta [41]

$$\begin{cases} y_k = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \dots, \\ z_k = y_k - \frac{f(y_k)}{f'(x_k)} \frac{f(x_k) + \beta f(y_k)}{f(x_k) + (\beta - 2) f(y_k)}, \\ x_{k+1} = z_k - \frac{f(z_k)}{f'(x_k)} \frac{f(x_k) - f(y_k)}{f(x_k) - 3f(y_k)}. \end{cases} \tag{7}$$

4. *Multipoint iterative method with memory* Finally, in this category, let us define another iteration function ϕ having arguments z_j, where each such argument represents $k+1$ quantities $x_j, w_1(x_j), \dots, w_n(x_j), (n \geq 1)$. Let the iteration mapping be defined by $x_{k+1} = \phi(z_k; z_{k-1}, \dots, z_{k-n})$. Then ϕ is called a multipoint IF with memory. In the above-mentioned mapping, semicolon separates the points at which new information is used from the point at which old information is reused, i.e., at each iterative step, we must preserve information of the last n approximations x_j and for each approximation, we must calculate n expressions $w_1(x_j), \dots, w_n(x_j)$. Some other researchers worked on this method such as: Cordero [10–13], Dezunic [15–17], Petkovic [44–49], Lotfi [33–37], Soleymani [55, 56], Wang [63, 64], and, ….

Conjecture Kung and Traub [31]: Kung and Traub proved the best one-point iterative method should achieve order of convergence n using n function evaluations. Also, any multipoint method should achieve optimal order convergence 2^n using $n+1$ evaluations. Abbasbandy [1], Chun [7], Kou [29], and, … worked on one-step methods and also, Petkovic [44], Sharma [53] and Thukral [60], and …, worked on multi-step methods.

Efficiency Index (EI) We recall the so-called efficiency index defined by Ostrowski [43], as $\mathrm{EI} = p^{1/n}$, where p is the order of convergence and n is the total number function evaluations per iteration. Lotfi [33] and Soleymani [62] have checked iterative methods with high efficiency index.

Note 1 We use the symbols $\rightarrow$, O, and $\sim$ according to the following conventions [61]. If $\lim_{x_n \rightarrow \infty} g(x_n) = C$, we write $g(x_n) \rightarrow C$ or $g \rightarrow C$. If $\lim_{x \rightarrow a} g(x) = C$, we write $g(x) \rightarrow C$ or $g \rightarrow C$. If $f/g \rightarrow C$ where C is a nonzero constant, we write $f = O(g)$ or $f \sim g$.

Traub investigated that it is possible to increase the order of convergence of without memory methods by reusing the obtained information of the previous iteration. If one can increase the order of convergence in a without memory method by reusing the old information, then he/she can develop it with a memory method. To our surprise, there is not any method with memory that reuses the information from the all previous information. This motivated us to focus on this problem. Therefore, in this work, we will develop an adaptive memory method that uses the information not only from the last two steps, but also from all the previous iterations. This technique enables us to achieve the highest efficiency both theoretically and practically. Indeed, we will prove that this adaptive memory method has efficiency index 2 and hence competes all the existing methods without and with memory in the literature. Also, we later compare both numerical performances and efficiency index of our proposed method with some significant methods to show our claims. We approximate and update the introduced accelerator parameters in each iteration by suitable kind and optimal of Newton's interpolation. We conclude that even with this one-step method, we need not to pay attention to higher kinds of steps in multipoint methods since this adaptive with memory method can achieve the efficiency index near 2 after three iterations, so from the theoretical and numerical aspects, it is enough to consider and utilize it practically. This paper is organized as follows:

In "A family of two-parameter iterative methods" section deals with modifying the optimal one-point method without memory introduced by family Khaksar [28], constructed by introducing two iterative parameters which are calculated with helped of Newton's interpolatory polynomial of different degrees. In "Recursive adaptive method with memory" section, the aim of this work is presented by contributing an iterative method adaptive with memory for solving nonlinear equations, improved order of convergence from 3.56 to 4 without adding more evaluations is presented, and achieve in maximum performance index. It means that, without any new function calculations, we can improve convergence order by 100%. The comparisons of absolute errors and computational efficiencies are given in "Numerical examples" section to illustrate convergence behavior. In "Conclusion" section, we give the concluding remarks.

A family of two-parameter iterative methods

In this section, we deal with modifying one-point without memory methods by Khaksar [28]. So that their error equation has two accelerator elements. Khaksar's method has the iterative expression:

$$\begin{cases} w_k = x_k - \beta f(x_k), \quad k = 0, 1, 2, \dots, \\ x_{k+1} = x_k - \frac{f(x_k)}{f[x_k, w_k]}\left(1 + \xi \frac{f(w_k)}{f[x_k, w_k]}\right). \end{cases} \tag{8}$$

Denoted by KM, where $\beta \in \Re - \{0\}$, its error equation is given by

$$e_{k+1} = (-1 + f'(\alpha)\beta)(\xi - c_2)e_k^2 + O(e_k^3). \tag{9}$$

To transform Eq. (8) in a method with memory, with two accelerators, we consider the following modification of (8) [28]:

$$\begin{cases} w_k = x_k - \beta_k f(x_k), \quad k = 0, 1, 2, \dots, \\ x_{k+1} = x_k - \frac{f(x_k)}{f[x_k, w_k]}\left(1 + \xi_k \frac{f(w_k)}{f[x_k, w_k]}\right), \end{cases} \tag{10}$$

where β and ξ are nonzero arbitrary parameters. In what follows, we present the error of Eq. (10).

Remark 1 It is worth noting that to the best of our knowledge although there are many methods with memory, however, developing adaptive methods with memory has not been considered in the literature.

The next theorem states of the error equation of Eq. (10).

Theorem 1 *Let $I \subseteq \mathbf{R}$ be an open interval, $f : I \to \mathbf{R}$ be a scalar function which has a simple root α in the open interval I, and also the initial approximation x_0 is sufficiently close the simple zero, and then, the one-step iteration method (10) has two orders, which satisfies the following error equation:*

$$e_{k+1} = (-1 + f'(\alpha)\beta)(\xi - c_2)e_k^2 + O(e_k^3). \tag{11}$$

Proof Let α be a simple zero of equation $f(x) = 0$ and $x_k = \alpha + e_k$. By Taylor expansion, we have :

$$f(x_k) = f'(\alpha)(e_k + c_2e_k^2 + c_3e_k^3), \tag{12}$$

where $c_k = \frac{f^{(k)}(\alpha)}{k!f'(\alpha)}$, $k = 2, 3, \dots$.

$$w_k = e_k - f'(\alpha)\beta(e_k + c_2e_k^2) + O(e_k^3). \tag{13}$$

Expanding $f(w_k)$ about α, we get :

$$\begin{aligned} f(w_k) = {} & f'(\alpha)(e_k - f'(\alpha)\beta(e_k + c_2e_k^2) \\ & + c_2(e_k - f'(\alpha)\beta(e_k + c_2e_k^2))^2)e_k^2 + O(e_k^3). \end{aligned} \tag{14}$$

If $f[x, y] = \frac{f(x)-f(y)}{x-y}$ is a divided difference, then the expression $f[x_k, w_k]$ can be written in terms of e_k as:

$$\begin{aligned} & f[x_k, w_k] \\ & = \frac{f'(\alpha)(e_k + c_2e_k^2) - f'(\alpha)(e_k - f'(\alpha)\beta(e_k + c_2e_k^2) + c_2(e_k - f'(\alpha)\beta(e_k + c_2e_k^2))^2)}{f'(\alpha)\beta(e_k + c_2e_k^2)}. \end{aligned} \tag{15}$$

Dividing (14) by (15) gives us :

$$\begin{aligned} & \frac{f(w_k)}{f[x_k, w_k]} \\ & = -\frac{f'(\alpha)^2\beta(e_k + c_2e_k^2)(e_k - f'(\alpha)\beta(e_k + c_2e_k^2) + c_2(e_k - f'(\alpha)\beta(e_k + c_2e_k^2))^2)}{-f'(\alpha)(e_k + c_2e_k^2) + f'(\alpha)(e_k - f'(\alpha)\beta(e_k + c_2e_k^2) + c_2(e_k - f'(\alpha)\beta(e_k + c_2e_k^2))^2)}. \end{aligned} \tag{16}$$

We also conclude by dividing Eq. (12) by (15) :

$$\frac{f(x_k)}{f[x_k,w_k]} = \frac{f'(\alpha)^2\beta(e_k+c_2e_k^2)^2}{f'(\alpha)(e_k+c_2e_k^2)-f'(\alpha)(e_k-f'(\alpha)\beta(e_k+c_2e_k^2)+c_2(e_k-f'(\alpha)\beta(e_k+c_2e_k^2))^2)}. \tag{17}$$

By substituting (12), (14), (16), and (17) in (8), it is obtained that

$$\begin{aligned} x_{k+1} &= \alpha+e_k-\frac{f(x_k)}{f[x_k,w_k]}\left(1+\xi\frac{f(w_k)}{f[x_k,w_k]}\right) \\ &= \alpha+(-1+f'(\alpha)\beta)(\xi-c_2)e_k^2+O(e_k^3). \end{aligned} \tag{18}$$

Therefore,

$$e_{k+1}=(-1+f'(\alpha)\beta)(\xi-c_2)e_k^2+O(e_k^3). \tag{19}$$

The proof is completed. □

Remark 2 The family of one-point methods mentioned in Eq. (10) requires two function evaluations and has order of convergence two. Therefore, this family is optimal in the sense of the Kung–Traub conjecture and possesses the computational efficiency EI $= 2^{1/2} \approx 1.4142$.

the best approximations. Hence, the following approximates are applied

$$\begin{cases} \beta_k = \frac{1}{f'(\alpha)} \approx \frac{1}{N_2'(x_k)}, \\ \xi_k = \frac{f''(\alpha)}{2f'(\alpha)} \approx \frac{N_3''(w_k)}{2N_3'(w_k)}, \end{cases} \tag{20}$$

where $k=1,2,\ldots$.

$N_2'(x_k), N_3'(w_k)$ and $N_3''(w_k)$ are Newton's interpolating polynomials of two and third degrees, set through three and four best available approximations (nodes) (x_k,x_{k-1},w_{k-1}) and $(w_k,x_k,x_{k-1},w_{k-1})$, respectively. It should be noted that if one uses lower Newton's interpolation, lower accelerators are obtained. Replacing the fixed parameters ξ and β in the iterative formula (10) by the varying β_k and ξ_k calculated by (20), we propose the following new methods with memory, x_0, ξ_0, β_0 are given then $w_0 = x_0 - \beta_0 f(x_0)$

$$\begin{cases} \beta_k = \frac{1}{N_{2k}'(x_k)},\ \xi_k = \frac{N_{2k+1}''(w_k)}{2N_{2k+1}'(w_k)}, \quad k=1,2,\ldots, \\ w_k = x_k-\beta_k f(x_k),\ x_{k+1}=x_k-\frac{f(x_k)}{f[x_k,w_k]}(1+\xi_k\frac{f(w_k)}{f[x_k,w_k]}), \quad k=0,1,2,\ldots. \end{cases} \tag{21}$$

Recursive adaptive method with memory

This section concerns with extracting the novel with memory method from (10) by using two self-accelerating parameters. Theorem (1) states that modified method (10) has order of convergence 2 if $\beta \neq \frac{1}{f'(\alpha)}$ and $\xi \neq c_2$. Now, we pose a main question: Is it possible to increase the order of convergence ? If so, how can it be done and what is the new convergence order? For answering these questions, we note the error equation (11). It can be seen that if we set $\beta = \frac{1}{f'(\alpha)}$ and $\xi = c_2 = \frac{f''(\alpha)}{2f'(\alpha)}$, then at least the coefficient of e_k^2 disappears. However, we do not know α and consequently, $f'(\alpha)$ and $f''(\alpha)$ cannot be computed. On the other hand, we can approximate α using available data and therefore improve order of convergence. Following the same idea in the methods with memory, this issue can be resaved. However, we are going to do it in a more efficient way, say recursive adaptively. Let us describe it a little more. If we use information from the current and only the last iteration, we come up with the method introduced in [34, 36]. Also, we have considered $N_{2k}'(x_k), N_{2k+1}'(w_k)$ and $N_{2k+1}''(w_k)$ are Newton's interpolating polynomials of $2k$ and $2k+1$ degrees, set through $2k+1$ and $2k+2$ best available approximations (nodes) $(x_k,x_{k-1},w_{k-1},\ldots,w_1,x_1,w_0,x_0)$ and $(w_k,x_k,x_{k-1},w_{k-1},\ldots,w_1,x_1,w_0,x_0)$ respectively. Here, we concern the second question regarding order of convergence of the method with memory (10). In what follows, we discuss the general convergence analysis of the recursive adaptive method with memory (10). It should be noted that the convergence order varies as the iteration go ahead. First, we need the following lemma:

Lemma 1 *If* $\beta_k = \frac{1}{N_{2k}'(x_k)}$, *and* $\xi_k = \frac{N_{2k+1}''(w_k)}{2N_{2k+1}'(w_k)}$, *then the estimate*

$$\begin{cases} (-1+\beta_k f'(\alpha)) \sim \prod_{s=0}^{k-1} e_s e_{s,w}, \\ (\xi_k - c_2) \sim \prod_{s=0}^{k-1} e_s e_{s,w}, \end{cases} \tag{22}$$

where $e_s = x_s - \alpha, e_{s,w} = w_k - \alpha$.

Proof The proof is similar to Lemma 1 mentioned in [64].

The following result determines the order of convergence through the one-point iterative method with memory (21). □

Theorem 2 *If an initial estimation x_0 is close enough to a simple root α of $f(x)=0$ and β_0 and ξ_0 must be uniformly bounded above, being f a real sufficiently differentiable function, then the R-order of convergence of the one-point method adaptive with memory (21) obtained from the following system of nonlinear equations.*

$$\begin{cases} r^k p-(1+p)(1+r+r^2+r^3+\cdots+r^{k-1})-r^k=0,\\ r^{k+1}-2(1+p)(1+r+r^2+r^3+\cdots+r^{k-1})-2r^k=0, \end{cases} \tag{23}$$

where r and p are the convergence order of the sequences $\{x_k\}$ and $\{w_k\}$, respectively. Also, k indicates the number of iterations.

Proof Let $\{x_k\}$ and $\{w_k\}$ be convergent with orders r and p respectively. Then

$$\begin{cases} e_{k+1}\sim e_k^r\sim e_{k-1}^{r^2}\sim\ldots\sim e_0^{r^{k+1}},\\ e_{k,w}\sim e_k^p\sim e_{k-1}^{rp}\sim\ldots\sim e_0^{pr^k}, \end{cases} \tag{24}$$

where $e_k=x_k-\alpha$ and $e_{k,w}=w_k-\alpha$. Now, by Lemma (1) and Eq. (24),

we obtain:

$$\begin{aligned}(-1+\beta_k f'(\alpha))\sim\prod_{s=0}^{k-1}e_s e_{s,w}&=(e_0e_{0,w})\ldots(e_{k-1}e_{k-1,w})\\ &=(e_0e_0^p)(e_0^re_0^{pr})\ldots(e_0^{r^{k-1}}e_0^{r^{k-1}p})\\ &=e_0^{(1+p)+(1+p)r+\cdots+(1+p)r^{k-1}}\\ &=e_0^{(1+p)(1+r+\cdots+r^{k-1})}.\end{aligned} \tag{25}$$

Similarly, we get:

$$(\xi_k-c_2)\sim e_0^{(1+p)(1+r+\cdots+r^{k-1})}. \tag{26}$$

By considering the errors of w_k and x_{k+1} in Eq. (21) and Eqs. (25)–(26), we conclude:

$$e_{k,w}\sim(-1+\beta_k f'(\alpha))e_k\sim e_0^{(1+p)(1+r+\cdots+r^{k-1})}e_0^{r^k}, \tag{27}$$

$$e_{k+1}\sim(-1+\beta_k f'(\alpha))(\xi_k-c_2)e_k^2\sim e_0^{((1+p)(1+r+\cdots+r^{k-1}))^2}e_0^{2r^k}. \tag{28}$$

equating the powers of e_{k+1} on the right-hand sides of Eqs. (24)–(27) and (24)–(28), one can obtain:

$$\begin{cases} r^k p-(1+p)(1+r+r^2+r^3+\cdots+r^{k-1})-r^k=0,\\ r^{k+1}-2(1+p)(1+r+r^2+r^3+\cdots+r^{k-1})-2r^k=0. \end{cases} \tag{29}$$

And thus we prove the result. □

Remark 3 It should be kept in mind that the system of equations (23) includes the previous iterations for $k=0,1,2,\ldots$. In this case, we have the regular methods with memory in which the information from the current and the previous steps are used.

Remark 4 For $k=1$, we use the information from the current and the one previous step. In this case, the order of convergence of the method with memory can be computed from the following of system of equations

$$\begin{cases} rp-(1+p)-r=0,\\ r^2-2(1+p)-2r=0. \end{cases} \tag{30}$$

This system of equations has the solution $p=\frac{1}{4}(3+\sqrt{17})\simeq 1.78078$, and $r=\frac{1}{2}(3+\sqrt{17})\simeq 3.56155$.

This special case gives the given result by khaksar haghani [28]. This is a new kind of adaptive approach with memory method .

Remark 5 For $k=2$, the system of equations (23) becomes :

$$\begin{cases} r^2p-(1+p+rp+r+r^2)=0,\\ r^3-2(1+p+rp+r+r^2)=0. \end{cases} \tag{31}$$

This system of equations has the solution: $p\simeq 1.95029$ and $r\simeq 3.90057$.

Remark 6 If $k=3$, we get:

$$\begin{cases} (-1+\beta_k f'(\alpha))\sim e_{k-3}e_{k-3,w}e_{k-2}e_{k-2,w}e_{k-1}e_{k-1,w}\sim e_{k-3}^{1+p+r+rp+r^2+r^2p},\\ (\xi_k-c_2)\sim e_{k-3}e_{k-3,w}e_{k-2}e_{k-2,w}e_{k-1}e_{k-1,w}\sim e_{k-3}^{2(1+p+r+rp+r^2+r^2p)}. \end{cases} \tag{32}$$

and equating the powers of e_{k+1} and $e_{k,w}$ error exponents of in pairs of relations (24), and (29) we obtain:

$$\begin{cases} r^3p-(1+p+r+rp+r^2+r^2p+r^3)=0,\\ r^4-2(1+p+r+rp+r^2+r^2p+r^3)=0. \end{cases} \tag{33}$$

Positive solution of the system of equations (33) is given by: $p\simeq 1.98804$ and $r\simeq 3.97609$.

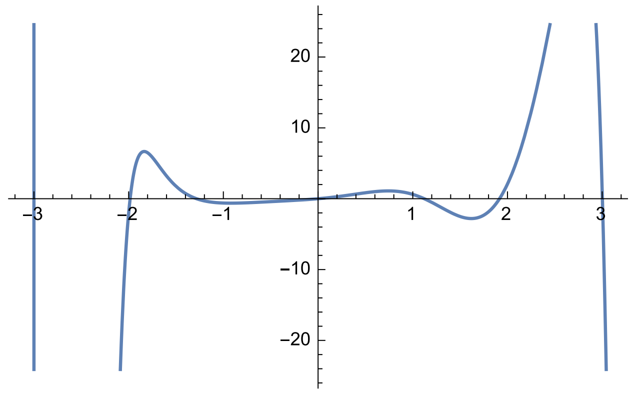

Fig. 1 $f_1(t), t \in [-\pi, \pi]$

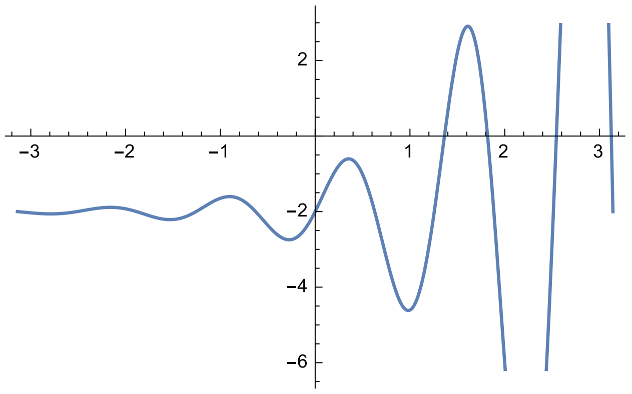

Fig. 2 $f_2(t), t \in [-\pi, \pi]$

Remark 7 Also, if $k = 4$, we conclude by the system of equations (29): (shown by TLAM)

$$\begin{cases} r^4p - (1 + p + r + rp + r^2 + r^2p + r^3 + r^3p + r^4) = 0, \\ r^5 - 2(1 + p + r + rp + r^2 + r^2p + r^3 + r^3p + r^4) = 0. \end{cases} \tag{34}$$

Solving these equations, we get : $p \simeq 1.99705$ and $r \simeq 3.9941$.

Remark 8 As can be easily seen that the improvement in the order of convergence from 2 to 4 (100% of an improvement) is attained without any additional functional evaluations, which points to very high computational efficiency of the proposed method. Therefore, the efficiency index of the proposed method (23) is $\text{EI} = 4^{1/2} = 2, \;(k \geq 4)$.

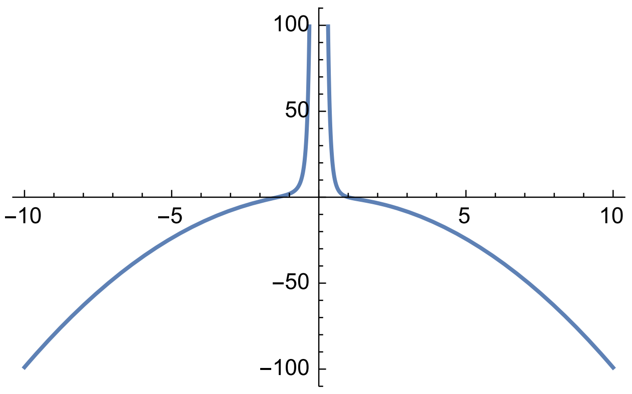

Fig. 3 $f_3(t), t \in [-10, 10]$

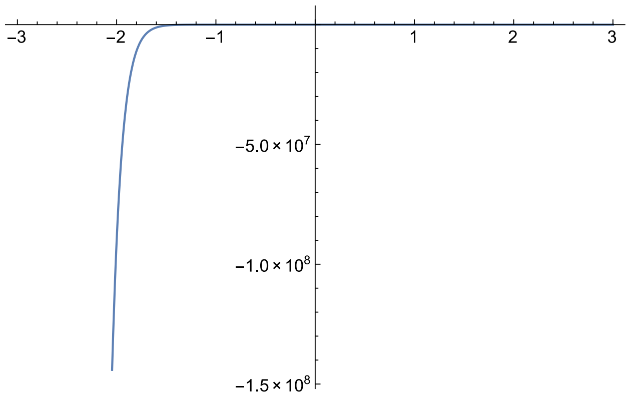

Fig. 4 $f_4(t), t \in [-3, 3]$

Numerical examples

In this section, the proposed derivative-free adaptive methods are applied to solve smooth as well as nonsmooth nonlinear equations and compared with the existing without memory and with memory methods. The iterative methods without memory and with memory are listed in Tables 1 and 2, respectively. Table 3 lists the exact roots α and initial approximations x_0, which are computed using the *FindRoot* command of Mathematica [23]. Table 4 compares evaluation function and efficiency index of the proposed method by with and without memory schemes. Table 5 compares improvement percent with memory and homogeneous without memory. Constructed iteration adaptive method, with the given function f having a simple zero is mentioned in Table 6. Tables 7, 8 and 9 compare our proposed method forty one with and without memory. In recent years, since in practice high-precision computations are applied, the higher-efficiency index schemes have become important. Due to this reason all the computations reported have been performed in the programming package Mathematica 10 using 2000 digits floating-point arithmetic using "SetAccuraccy"command. The errors $|x_k - \alpha|$ of approximations to the sought zeros, produced by the different methods at the first three iterations, are given in Table 6 where $m(-n)$

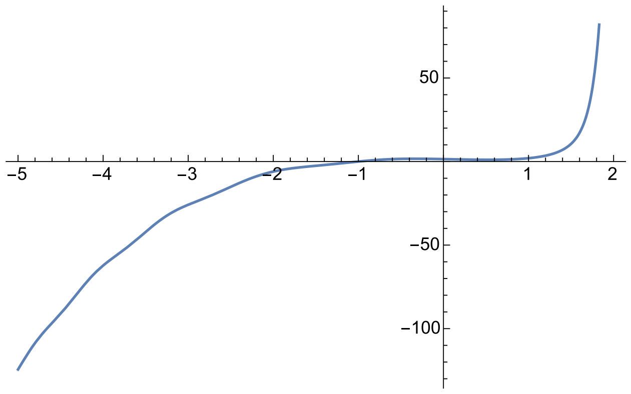
Fig. 5 $f_5(t), t \in [-5, 2]$

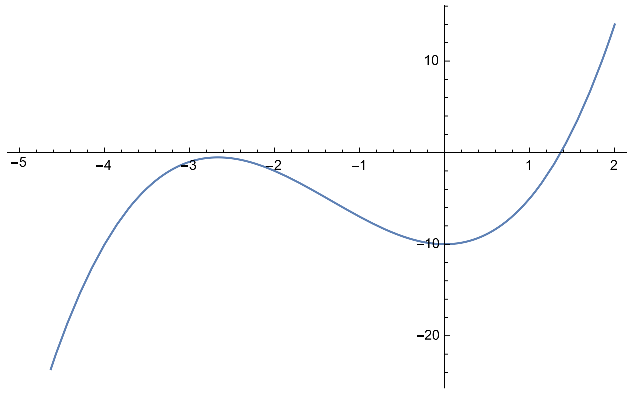
Fig. 8 $f_8(t), t \in [-5, 2]$

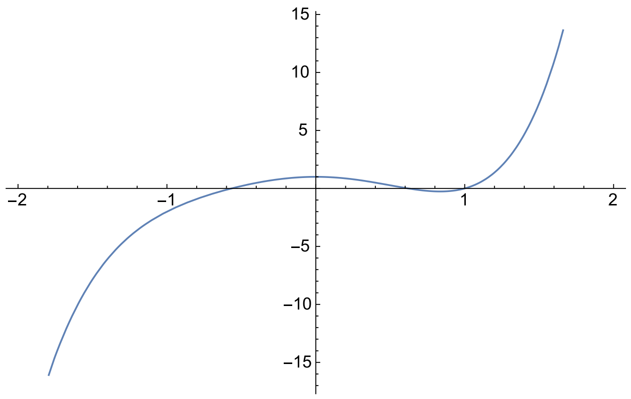
Fig. 6 $f_6(t), t \in [-2, 2]$

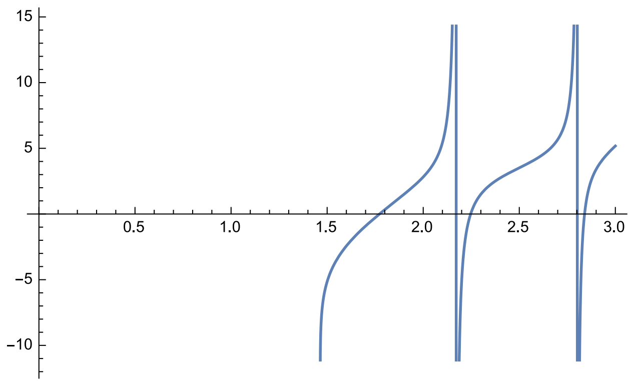
Fig. 9 $f_9(t), t \in [0, 3]$

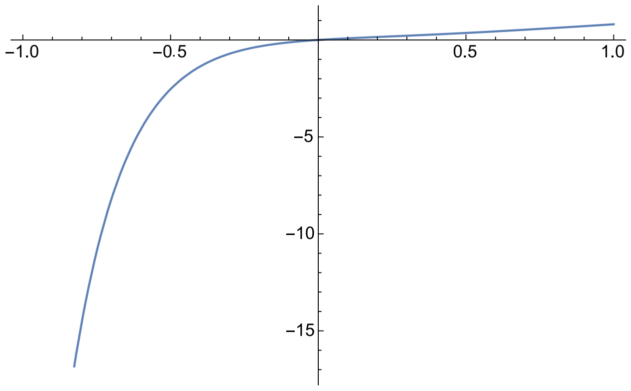
Fig. 7 $f_7(t), t \in [-1, 1]$

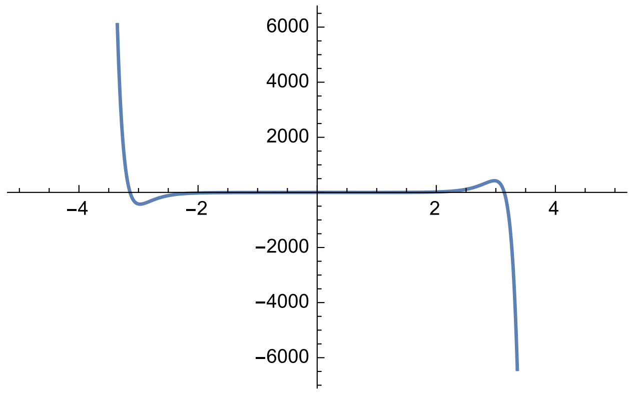
Fig. 10 $f_{10}(t), t \in [-4, 4]$

stands for $m \times 10^{-n}$. These tables also include, for each test function, the initial estimation values and the last value of the computational order of convergence COC [44] computed by the expression (if it is stable)

$$\mathrm{COC} = \frac{\log |f(x_n)/f(x_{n-1})|}{\log |f(x_{n-1})/f(x_{n-2})|} \approx p, \tag{35}$$

where p is the order of convergence. At least 40 iterative methods with and without memory, for comparing with our

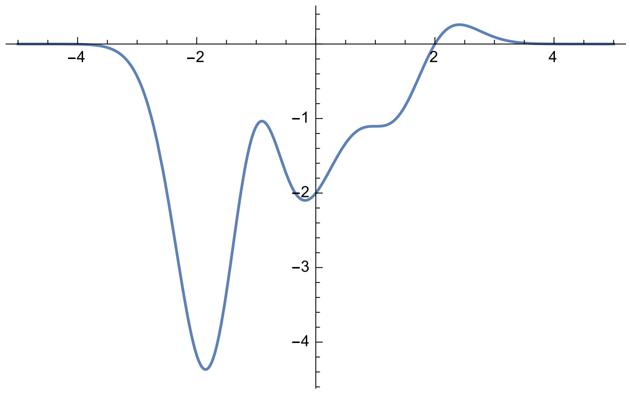

Fig. 11 $f_{11}(t), t \in [-5, 5]$

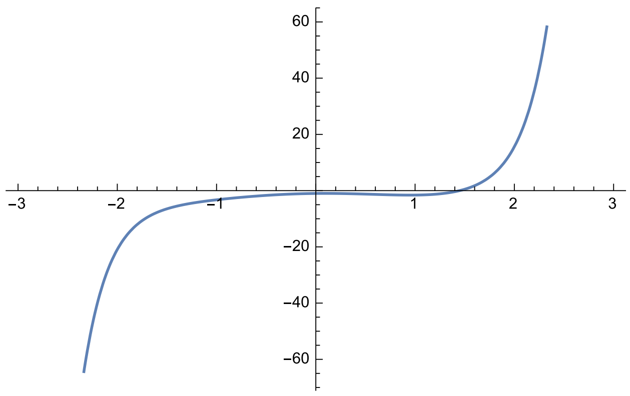

Fig. 12 $f_{12}(t), t \in [-3, 3]$

proposed methods, have been chosen as comes next. Test functions used in many papers concerning nonlinear equations. For example, the functions $f_i(x), i = 1, 2, 3, \dots, 12$ are displayed in Figs. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12, respectively. Figure 13 compares of methods without memory, with memory (25%, 50% and 75% of improvements) and recursive adaptive (100% of improvements) in terms of highest possible efficiency index. Complex test function f_{10} used to show that the proposed method is applicable to the complex domain too. In these tables symbols *In*, *div* have demonstrator infinity and divergence, respectively. It can be observed our proposed method has minimum evaluation function and maximum efficiency index. Tables 4 and 5 show that the method (23) competes the previous methods. In additional its efficiency index is better than all the previous works. In other words, it has efficiency index $4^{1/2} = 2$. The same results can be observed in the second and third columns of Table 5 and at least has evaluation function inter iterative methods existent methods with- and without memory. Some of iterative methods in the some examples are divergent. We also incorporated and applied the developed adaptive method with memory (34) for different test examples and obtained results with the same behavior as above. We can see that the self-accelerating parameters and the consequently adapting method play a key role in increasing the order of convergence of the iterative method.

Algorithms to find an initial approximation

1

An important aspect in implementing the iterative methods for the solution of nonlinear equations and systems relies on the choice of the initial approximation. There are a few known ways in the literature [24] to extract a starting point for the solutions of nonlinear functions. In practice, users need to find out robust approximations for all the zeros in an interval. Thus, to remedy this and to respond on this need, we provide a way to extract all the real zeros of nonlinear function in the interval $D = [a, b]$. We use the command *Reduce* in Mathematica 10 [23]. Hence, we give a hybrid algorithm including two main steps, a predictor and a corrector. In the predictor step, we extract initial approximations for all the zeros in an interval up to 8 decimal places. Then the corrector step will be used to boost up the accuracy of the starting points up to any tolerance. We also give some significant cautions for applying on different test

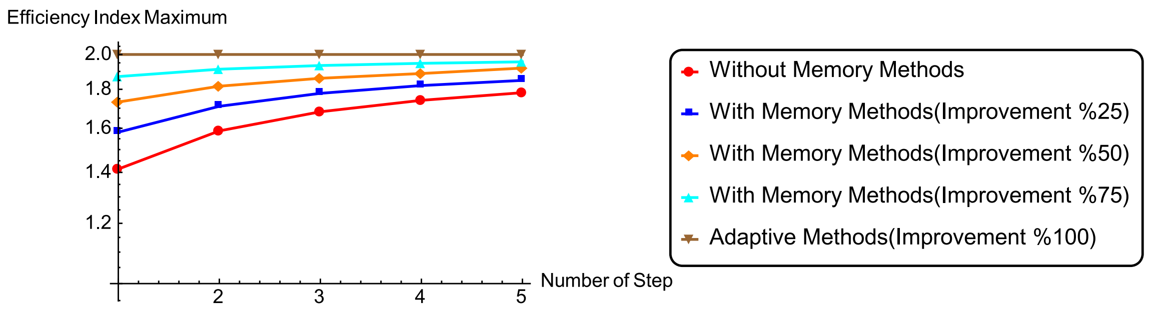

Fig. 13 Comparison of methods without memory, with memory (25%, 50%, and 75% of improvements) and recursive adaptive (100% of improvement) in terms of highest possible efficiency index

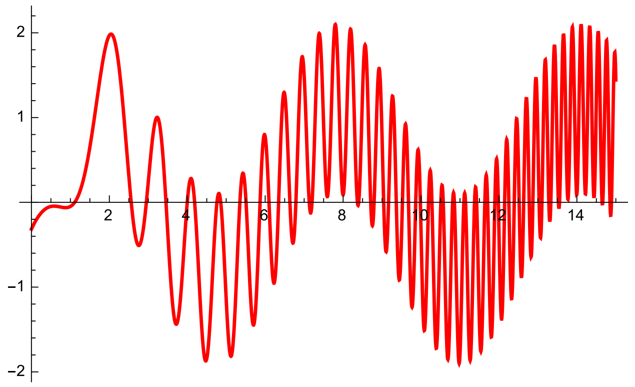

Fig. 14 The graph of the function f with finitely many zeros in an interval

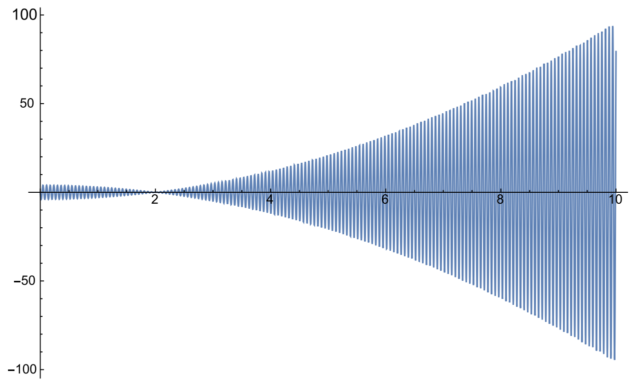

Fig. 15 The graph of the function g with finitely many zeros in an interval

functions. In what follows, we keep going by choosing an oscillatory function $f(x) = \frac{1}{10} + \cos(2 + x^2) + \sin(x)$ in the domain $D = [0., 15.]$.

Let us define the function and the domain for imposing the *Reduce*[ ] command as in Algorithm 1.

One may note that *Reduce*[ ] works with function of exact arithmetic. Hence, if a nonlinear function is the floating-point arithmetic, that is, has inexact coefficients, thus we should write it in the exact arithmetic when we enter it into the above piece of code. Now we store the list of initial approximations in initialValues, by the following piece of code, which also sort the initial points. The tol will specify that the accuracy of each member of the provided sequence to be correct up to utmost tol, decimal places (Algorithm 2).

It is obvious that f is so oscillatory, and by the above predictor piece of Mathematica code, we attain that it has 59 real solutions. Note that the graph of the function f has been drawn in Fig. 14.

Note that if a user needs much more accuracy, thus higher number of steps should be taken. It should be remarked that in order to work with such a high accuracy, we must then choose more than 2000 decimal places arithmetic in our calculations.

However, running the above algorithm could capture all the real zeros of the nonlinear functions. One is that for many oscillatory function or for nonsmooth functions, the best way is to first divide the whole interval into some subintervals and then find all the zeros of the function on the subintervals. And second, in case of having a root cluster, that is, when the zeros are concentrated on a very small area, then it would be better to increase the *first tolerance* of our algorithm in the predictor step, to find reliable starting points and then start the process.

And last, if the nonlinear function has an exact solution, that is to say, an integer be the solution of a nonlinear function, then the first step of our algorithm finds this exact solution, and an error-like message would be generated by applying our second step. For instance, the function $g(x) = (x^2 - 4)\sin(100x)$ on the interval $D = [0, 10]$ has 319 real solutions in which one of them (its plot is given in Fig. 15), that is. 2, is an exact one. Thus, the first step of the mentioned Algorithm 1 finds the following very efficient list of starting points in which 2, is the exact solution:

```
{0.031416, 0.0628318, 0.0942478,
0.125664, 0.15708, 0.188496, 0.219912,
0.251327, 0.282743, 0.314159, 0.345575,
0.376991, 0.408407, 0.439823, 0.471239,
0.502655, 0.534071, 0.565487,
0.596903, 0.628319, 0.659734,
0.69115, 0.722566, 0.753982, 0.785398,
0.816814, 0.84823, 0.879646, 0.911062,
0.942478, 0.973894, 1.00531, 1.03673,
1.06814, 1.09956,1.13097, 1.16239,
1.19381, 1.22522, 1.25664, 1.28805,
1.31947, 1.35088,1.3823, 1.41372,
1.44513, 1.47655, 1.50796, 1.53938,
1.5708, 1.60221, 1.63363, 1.66504,
1.69646,1.72788, 1.75929, 1.79071,
1.82212, 1.85354, 1.88496, 1.91637,
1.94779, 1.9792, 2., 2.01062, 2.04203,
2.07345, 2.10487, 2.13628, 2.1677,
2.19911, 2.23053, 2.26195, 2.29336,
2.32478, 2.35619, 2.38761, 2.41903,
2.45044,2.48186, 2.51327, 2.54469,
2.57611,2.60752,2.63894, 2.67035,
2.70177, 2.73319, 2.7646, 2.79602,
2.82743, 2.85885, 2.89027, 2.92168,
2.9531, 2.98451, 3.01593, 3.04734,
3.07876, 3.11018, 3.14159, 3.17301,
3.20442, 3.23584, 3.26726, 3.29867,
3.33009, 3.3615, 3.39292, 3.42434,
3.45575, 3.48717, 3.51858,3.55,
3.58142, 3.61283, 3.64425, 3.67566,
3.70708, 3.7385, 3.76991, 3.80133,
3.83274, 3.86416, 3.89557, 3.92699,
3.95841, 3.98982, 4.02124, 4.05265,
4.08407, 4.11549, 4.1469, 4.17832,
4.20973, 4.24115, 4.27257, 4.30398,
4.3354, 4.36681, 4.39823, 4.42965,
4.46106, 4.49248, 4.52389,4.55531,
4.58673,4.61814, 4.64956, 4.68097,
4.71239, 4.7438, 4.77522, 4.80664,
4.83805,4.86947, 4.90088, 4.9323,
4.96372, 4.99513, 5.02655, 5.05796,
5.08938, 5.1208, 5.15221, 5.18363,
5.21504, 5.24646, 5.27788, 5.30929,
5.34071, 5.37212, 5.40354, 5.43496,
5.46637, 5.49779, 5.5292, 5.56062,
5.59203, 5.62345, 5.65487, 5.68628,
5.7177, 5.74911, 5.78053, 5.81195,
5.84336, 5.87478, 5.90619, 5.93761,
5.96903, 6.00044, 6.03186, 6.06327,
6.09469, 6.12611, 6.15752, 6.18894,
6.22035, 6.25177, 6.28319, 6.3146,
6.34602, 6.37743, 6.40885, 6.44026,
6.47168, 6.5031, 6.53451, 6.56593,
6.59734, 6.62876, 6.66018, 6.69159,
6.72301, 6.75442, 6.78584, 6.81726,
6.84867, 6.88009, 6.9115, 6.94292,
6.97434, 7.00575, 7.03717, 7.06858,
7.1, 7.13142, 7.16283, 7.19425, 7.22566,
7.25708, 7.28849, 7.31991, 7.35133,
7.38274, 7.41416, 7.44557, 7.47699,
7.50841, 7.53982, 7.57124, 7.60265,
7.63407, 7.66549, 7.6969, 7.72832,
7.75973, 7.79115, 7.82257, 7.85398,
7.8854, 7.91681, 7.94823, 7.97965,
8.01106, 8.04248, 8.07389, 8.13672,
8.16814, 8.19956, 8.23097, 8.26239,
8.2938, 8.32522, 8.35664, 8.38805,
8.41947, 8.45088, 8.4823, 8.51372,
8.54513, 8.57655, 8.60796, 8.63938,
8.6708, 8.70221, 8.73363, 8.76504,
8.79646, 8.82788, 8.85929, 8.89071,
8.92212, 8.95354, 8.98495, 9.01637,
9.04779, 9.0792, 9.11062, 9.14203,
9.17345, 9.20487, 9.23628, 9.2677,
9.29911, 9.33053, 9.36195, 9.39336,
9.42478, 9.45619, 9.48761, 9.51903,
9.55044,9.58186,9.61327, 9.64469,
9.67611, 9.70752, 9.73894, 9.77035,
9.80177, 9.83319, 9.8646, 9.89602,
9.92743, 9.95885, 9.99026}
```

Now we are able to solve nonlinear equations with finitely many roots in an interval and find all the real zeros in a short piece of time. Finding robust ways, to capture the complex solutions along working with complex nonlinear functions, can be taken into account as future works.

$$\begin{gathered}
f[x] := \frac{1}{10} + \cos[2 + x^2] + \sin[x]; a = 0.; b = 15.; \\
zeros = Reap[soln = y[x]/.First[NDSolve[\{y'[x] == Evaluate[D[f[x], x]], \\
y[b] == (f[b])\}, y[x], \{x, a, b\}, Method_{>}\{"EventLocator", \\
"event"_{>}y[x], "EventAction" :> Sow[\{x, y[x]\}]\}]]][[2, 1]]; \\
initialPoints = Sort[Flatten[Take[zeros, Length[zeros], 1]]];
\end{gathered}$$

Alghorithm 1

$$\begin{gathered}
Length[initialPoints] \\
Plot[f[x], \{x, a, b\},\ Epilog_{>}\{PointSize[Medium], Red, Point[zeros]\}, \\
PlotRange_{>}All, PerformanceGoal_{>}"Quality", PlotStyle_{>}\{Thick, Brown\}];
\end{gathered}$$

Alghorithm 2

2

An important aspect of implementing high-order nonlinear solvers is in finding very robust initial guesses to start the process, when high-precision computing is needed. As discussed in "Introduction and preliminaries" section, the convergence of our iterative methods is local. To resolve this shortcoming, the best way is to rely on hybrid algorithms, in which the first item produces a robust initial point and the second item employs the new iterative methods when high precision is required. There are some ways in the literature to find robust starting points, mostly based on interval mathematics see, for example, [3]. But herein we take into consideration the programming package Mathematica 10 [23] which could be efficiently applied on lists for high-precision computing. In fact using [24], we could build a list of initial guesses close enough with good accuracy to start the procedure of our optimal derivative-free fourth-order methods. The procedure of finding such a robust list is based on the powerful command of *NDSolve* for the nonlinear function $f(x) = \frac{1}{10} + \cos(2 + x^2) + \sin(x)$ on the interval $D = [a, b]$. Such a way can be written in the following piece of Mathematica code by considering an oscillatory function as the input test function on the domain $D = [0., 15.]$. See Algorithm 1. The output of Algorithm 3 is to plot the function graph $f(x)$.

Thus now, we have an efficient list of initial approximations for the zeros of a nonlinear once differentiable function with finitely many zeros in an interval. The number of zeros and the graph of the function including the positions of the zeros can be given by the following commands (see Fig. 14); see Algorithm 4.

For this test, there are 59 zeros in the considered interval which can easily be used as the starting points for our proposed high-order derivative-free methods. Note that the output of the vector "initialPoints" contains the initial approximations. Note that we end this section by mentioning that for very oscillatory functions, it is better to first divide the interval into some smaller subintervals and then obtain the solutions. The command *NDSolve* uses Maximum number of 10,000 steps, if it is needed this could be changed. In cases when *NDSolve* fails, this algorithm might fail too. The output of Algorithm 4 is as follows:

```
 {1.1103225,  2.5611445,  2.9496729,
3.4537697,  3.9993453,  4.1889818,
4.7622341,  4.8587772,  5.3502573,
5.5085282,  5.8682448,  6.0980068,
6.3442691,  6.6342307,  6.7876268,
7.1310609,  7.2020570,  8.3675131,
8.3999413,  8.7079140,  8.7949106,
9.0413573,  9.1668305,  9.3646249,
9.5223085,  9.6781865,  9.8636235,
9.9828725,  10.192138,  10.279661,
10.508430,  10.569895,  10.811671,
10.856029,  11.099569,  11.141751,
11.373401,  11.426999,  11.637691,
11.708314,  11.895029,  11.984045,
12.146536,  12.253911,  12.392804,
12.518074,  12.634217,  12.776841,
12.871042,  13.030597,  13.103445,
13.279866,  13.331413,  13.525843,
13.554222,  14.647052,  14.664168,
14.849621, 14.887657 }
 59
 6.82717
```

```
ClearAll["*"]
$f[x] := \frac{1}{10} + \cos[2 + x^2] + \sin[x]; a = 0.; b = 15.;$
$Plot[f[x], \{x, a, b\}, Background \to LightBlue, PlotStyle \to \{Magenta, Thick\},$
$PlotRange \to All, PerformanceGoal \to "Quality"]$
$rts = Reduce[f[x] == 0,\ a \le x \le b,\ x];$
```

Alghorithm 3

```
tol=8
$initialValues = Sort[N[x/.\{ToRules[rts]\}, tol]];$
Length[initialValues]
Accuracy[initialValues]
```

Alghorithm 4

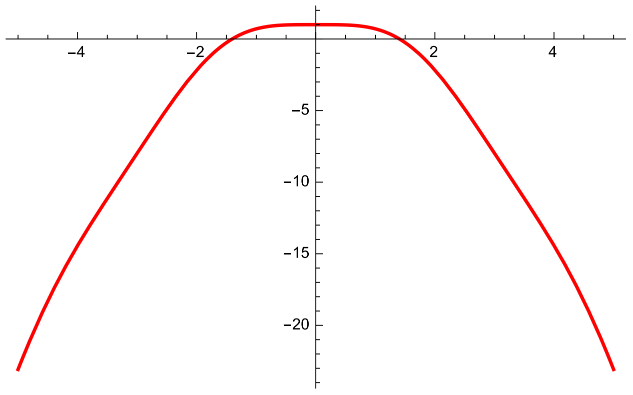

Fig. 16 The graph of the function h with finitely many zeros in an interval

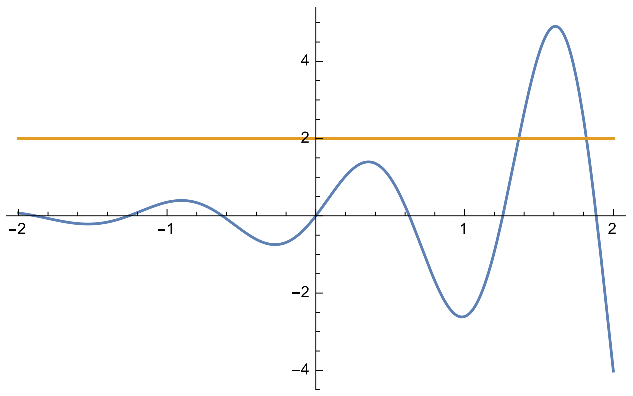

Fig. 17 The graph of the function $f_{20}(x) = \sin(5x)e^x, f_{21}(x) = 2$ with finitely many zeros in an interval

Although the choice of good initial approximations is of great importance in the application of iterative methods, including multipoint methods, this task is very seldom considered in the literature. Recall that Steffensen-like methods of the second order have been most frequently used as predictors in the first step of multipoint methods. These methods are of tangent type, and therefore, they are locally convergent, which means that a reasonably close initial approximation to the sought zero should be found. Otherwise, if the chosen initial approximation is too far from the sought zero (say, if it is chosen randomly), then the applied methods, either the ones proposed in this paper or some others with local convergence developed during the last two centuries, will probably find some other (often unwanted) zero or they will diverge.

Therefore, the determination of a reasonably good approximation x_0 that guarantees the convergence of the sequence of approximations $\{x_k\}_{k\in N}$ to the zero of f is a significant task. It is interesting to note that initial approximations, chosen randomly in a suitable way, give acceptable results when simultaneous methods for finding all roots of polynomial equations are applied, e.g., employing Aberth's approach [2].

There are many methods (mainly of non-iterative nature) and strategies for finding sufficiently good initial approximations. The well-known bisection method and its modifications belong to the simplest but not always sufficiently efficient techniques. There is a vast literature on this subject so that we omit details here. We only note that complete root-finding algorithms often consist of two parts: (1) slowly convergent search algorithm to isolate distinct real or complex

Table 1 Considered methods without memory

One-step	Two-step	Three-step	Four-step
Abbasbandy (AM) [1]	Chun (CM) [7]	Bi et al. (BWRM) [6]	Geum-Kim (GKM) [19]
Hansen-Patrick (HPM) [22]	Dehghan-Hajarian (DHM) [14]	Chun-Neta (CNM) [8]	Kreetee et al. (KBTM) [30]
Newton (NM) [43]	Ezzati-Saleki (ESM) [18]	Cordero et al. (CLMTM) [12]	Li et al. (LMMWM) [32]
Chebyshev (ChM) [47]	Kung–Traub (KTM) [31]	Kanwar et al. (KBKM) [27]	Sharifi et al. (SSSLM) [52]
Halley (HM) [21]	Mahehwari (MM) [39]	Matinfar et al. (MAAM) [40]	Thukral (TM) [60]
Steffensen (SM) [58]	Ren et al. (RWBM) [50]	Singh-Jaiswal (SJM) [54]	Zheng et al. (ZLHM) [66]
Zheng et al. (ZLHM) [66]	Soleymani-Mousavi (SMM) [57]	Taher-Khani (TkM) [59]	Guo-Qian (GQM) [20]

Table 2 Studied methods with memory

One-step	Two-step	Three-step
Dzunic (DM) [15]	Bassiri et al. (BBAM) [4]	Dzunic et al. (DPPM) [17]
Dzunic-Petkovic (DPM) [16]	Cordero et al. (CLBTM) [10]	Lotfi-Assari (LAM) [33]
Khaksar (KM) [28]	Cordero et al. (CLKTM) [11]	Lotfi et al. (LMNKSM) [34]
Lui-Zhang (LZM) [38]	Kansal et al. (KKBM) [26]	Lotfi et al. (LSGAM) [35]
Secant (SecM) [47]	Lotfi-Tavakoli (LTM) [37]	Lotfi et al. (LSSAKM) [36]
Traub (TrM) [61]	Wang et al. (WZQM) [64]	Sharifi et al. (SSSM) [51]

Table 3 The test functions

Nonlinear function	Root	Initial guess
$f_1(x) = x\log(1 + x\sin(x)) + e^{-1+x^2+x\cos(x)}\sin(\pi x)$	$\alpha = 0$	$x_0 = 0.3$
$f_2(x) = \sin(5x)e^x - 2$	$\alpha = 1.36$	$x_0 = 1$
$f_3(x) = 1 + \frac{1}{x^4} - \frac{1}{x} - x^2$	$\alpha = 1$	$x_0 = 1.4$
$f_4(x) = (x - 2)(x^{10} + x + 2)e^{-5x}$	$\alpha = 2$	$x_0 = 2.3$
$f_5(x) = e^{x^3-x} - \cos(x^2 - 1) + x^3 + 1$	$\alpha = -1$	$x_0 = -1.3$
$f_6(x) = \frac{-5x^2}{2} + x^4 + x^5 + \frac{1}{1+x^2}$	$\alpha = 1$	$x_0 = 1.3$
$f_7(x) = \log(1 + x^2) + e^{-3x+x^2}\sin(x)$	$\alpha = 0$	$x_0 = 0.3$
$f_8(x) = x^3 + 4x^2 - 10$	$\alpha = 1.3652$	$x_0 = 1$
$f_9(x) = x\log(1 - \pi + x^2) - \frac{1+x^2}{1+x^3}\sin(x^2) + \tan(x^2)$	$\alpha = \sqrt{\pi}$	$x_0 = 1.7$
$f_{10}(x) = (-1 + 2i) + \frac{1}{x} + x + \sin(x)$	$\alpha = 0.28860 - 0.2422i$	$x_0 = \frac{-i}{2}$
$f_{11}(x) = (x - 2)(x^6 + x^3 + 1)e^{-x^2}$	$\alpha = 2$	$x_0 = 1.8$
$f_{12}(x) = e^{x^2-1}\sin(x) + \cos(2x) - 2$	$\alpha = 1.44$	$x_0 = 1.1$
$f_{13}(x) = e^x\sin(x) + log(x^4 - 3x + 1)$	$\alpha = 0$	$x_0 = -0.5$
$f_{14}(x) = (x - 1)(x^{10} + x^3 + 1)\sin(x)$	$\alpha = 1$	$x_0 = 1.5$
$f_{15}(x) = x^2\sin(x^2) + e^{x\cos(x)\sin(x)} - 18$	$\alpha = 9.98$	$x_0 = 9.6$
$f_{16}(x) = x^4 + \sin(\frac{\pi}{x^2}) - 5$	$\alpha = 1.41$	$x_0 = 1$
$f_{17}(x) = \arcsin(x^2 - 1) - x/2 + 1$	$\alpha = 0.59$	$x_0 = 1$
$f_{18}(x) = \sqrt{x^4 + 8}\sin(\frac{\pi}{x^2+2}) + \frac{x^3}{x^4+1} - \sqrt{6} + \frac{8}{17}$	$\alpha = -2$	$x_0 = -2.3$
$f_{19}(x) = e^{\sin(x)} - 1 - x/5$	$\alpha = 0$	$x_0 = 0.5$
$f_{20}(x) = \arcsin(e^{x+2} + 1) + \tanh(e^{-x\cos(x)}) - \sin(\pi x)$	$\alpha = -3.98$	$x_0 = -4.3$

Table 4 Numerical results for the test functions $f_i(x), i = 1, 2, 3, \dots, 20$ the proposed method (34)

Function	$\lvert x_1 - \alpha\rvert$	$\lvert x_2 - \alpha\rvert$	$\lvert x_3 - \alpha\rvert$	$\lvert x_4 - \alpha\rvert$	COC	EI
$f_1(x), \beta_0 = \xi_0 = 0.1$	0.189 (−4)	0.152 (−19)	0.478 (−79)	0.222 (−317)	4.0054	2.00135
$f_2(x), \beta_0 = \xi_0 = 0.1$	0.396 (−2)	0.397 (−2)	0.397 (−2)	397 (−2)	3.9988	1.99970
$f_3(x), \beta_0 = \xi_0 = 0.1$	0.492 (−3)	0.992 (−13)	0.473 (−51)	0.283 (−204)	3.9984	1.99960
$f_4(x), \beta_0 = \xi_0 = 0.1$	0.983 (−4)	0.548 (−16)	0.100 (−66)	0.214 (−270)	4.0142	2.00355
$f_5(x), \beta_0 = \xi_0 = 0.1$	0.906 (−8)	0.117 (−31)	0.160 (−126)	0.981 (−506)	3.9975	1.99937
$f_6(x), \beta_0 = \xi_0 = 0.1$	0.127 (−5)	0.412 (−22)	0.451 (−88)	0.651 (−352)	4.0000	2.00000
$f_7(x), \beta_0 = \xi_0 = 0.1$	0.187 (−3)	0.837 (−14)	0.503 (−55)	0.679 (−220)	3.9996	1.99990
$f_8(x), \beta_0 = \xi_0 = 0.1$	0.301 (−4)	0.301 (−4)	0.301 (−4)	.301 (−4)	4.0000	2.00000
$f_9(x), \beta_0 = \xi_0 = 0.1$	0.129 (−10)	0.544 (−42)	0.221 (−167)	0.665 (−669)	3.9996	1.99990
$f_{10}(x), \beta_0 = \xi_0 = 0.1$	0.116 (1)	0.115 (1)	0.115 (1)	0.115 (1)	3.9992	1.99980
$f_{11}(x), \beta_0 = \xi_0 = 0.1$	0.194 (−5)	0.293 (−22)	0.861 (−90)	0.534 (−360)	4.0012	2.00300
$f_{12}(x), \beta_0 = \xi_0 = 0.1$	0.121 (−1)	0.779 (−2)	0.779 (−2)	0.779 (−2)	3.9998	1.99995
$f_{13}(x), \beta_0 = -0.001, \xi_0 = -0.01$	0.142 (−2)	0.112 (−10)	0.530 (−43)	0.361 (−172)	3.9882	1.99705
$f_{14}(x), \beta_0 = -0.001, \xi_0 = -0.01$	0.222 (−2)	0.640 (−2)	0.115 (−2)	0.161 (1)	3.9302	1.98247
$f_{15}(x), \beta_0 = \xi_0 = 0.1$	0.290 (0)	0.290 (0)	0.290 (0)	0.290 (0)	3.9976	1.99940
$f_{16}(x), \beta_0 = \xi_0 = 0.1$	0.421 (−2)	0.421 (−2)	0.421 (−2)	0.421 (−2)	4.0057	2.00142
$f_{17}(x), \beta_0 = \xi_0 = 0.1$	0.480 (−2)	0.481 (−2)	0.481 (−2)	0.481 (−2)	3.9983	1.99957
$f_{18}(x), \beta_0 = \xi_0 = 0.1$	0.249 (−6)	0.190 (−26)	0.377 (−107)	0.554 (−430)	4.0003	2.00007
$f_{19}(x), \beta_0 = \xi_0 = 0.1$	0.405 (−6)	0.154 (−27)	0.231 (−110)	0.377 (−443)	4.0179	2.00447
$f_{20}(x), \beta_0 = \xi_0 = 0.1$	0.615 (0)	0.598 (0)	0.598 (0)	0.598 (0)	4.0010	2.00025

Table 5 Comparison evaluation function and efficiency index of the proposed method with other schemes

Without memory methods	EF	EFD	COC	EI	With memory methods	EF	EFD	COC	EI
AM [1]	1	2	3.000	1.4423	DM [15]	2	0	3.550	1.8841
HPM [22]	1	2	4.000	1.5874	DPM [16]	2	0	3.000	1.7321
NM [43]	1	1	2.000	1.4142	KM [28]	2	0	3.550	1.8841
ChM [29]	1	2	3.000	1.4423	LZM [38]	2	0	3.380	1.8385
HM [21]	1	2	3.000	1.4423	SecM [47]	1	0	1.680	1.6800
CM [7]	2	1	4.000	1.5874	TrM [61]	2	0	2.410	1.5524
ESM [18]	3	0	4.000	1.5874	BBAM [4]	3	0	7.220	1.9328
RWBM [50]	3	0	4.000	1.5874	CLBTM [10]	3	0	7.000	1.9129
SMM [57]	2	1	4.000	1.5874	KKBM [26]	3	0	7.000	1.9129
ZLHM [66]	3	1	4.000	1.5874	LTM [37]	4	0	12.000	1.8612
BWRM [6]	3	1	8.000	1.6818	WZQM [64]	3	0	7.530	1.9601
CNM [8]	3	1	8.000	1.6818	DPPM [17]	4	0	11.000	1.8212
CLMTM [12]	3	1	8.000	1.6818	LAM [33]	4	0	15.500	1.9842
KBKM [27]	4	0	8.000	1.6818	LMNKSM [34]	4	0	12.000	1.8612
MAAM [40]	3	1	8.000	1.6818	LSGAM [35]	4	0	12.000	1.8612
SJM [54]	4	0	8.000	1.6818	LSSAKM [36]	4	0	14.000	1.9343
TM [60]	5	0	16.000	1.7411	SSSM [51]	4	0	12.000	1.8612
KBTM [30]	4	1	16.000	1.7411	LSSAKM [36]	4	0	12.000	1.8612
DHM [14]	3	0	3.000	1.4423	LSGAM [35]	3	0	7.238	1.9344
TkM [59]	3	1	8.000	1.6818	LAM [33]	4	0	15.000	1.9680
GQM [20]	5	0	16.000	1.7411	TLAM (34)	2	0	4.000	2.0000

Table 6 Comparison improvement of convergence order the proposed method with other schemes

With memory methods	Number of steps	Optimal order	p	Percentage increase
DM [15]	1	2.000	3.560	78
DPM [16]	1	2.000	3.000	50
KM [28]	1	2.000	3.560	78
LZM [38]	1	2.000	3.380	69
TrM [61]	1	2.000	2.410	20.5
BBAM [4]	2	4.000	7.220	80.5
CLBTM [10]	2	4.000	7.000	75
KKBM [26]	2	4.000	7.000	75
LSGAM [35]	2	4.000	7.238	80.95
WZQM [64]	2	4.000	7.530	88.25
LTM [37]	3	8.000	12.000	50
DPPM [17]	3	8.000	11.000	37.5
LAM [33]	3	8.000	15.500	93.75
LMNKSM [34]	3	8.000	12.000	50
LSGAM [35]	3	8.000	12.000	50
LSSAKM [36]	3	8.000	14.000	75
SSSM [51]	3	8.000	12.000	50
LSSAKM [36]	3	8.000	12.000	50
LAM [33]	3	8.000	15.000	87.5
TLAM (34)	1	2.000	4.000	100

interval containing single root and (2) rapidly convergent iterative method for finding sufficiently close approximation of the isolated root to the required accuracy. In this paper we are concentrating on the part (2). Applying computer algebra systems, a typical statement for solving nonlinear equations reads *FindRoot*[*equation*, $\{x, x_0\}$]; see, e.g., Wolfram's computational software package Mathematica, that is, an initial approximation x_0 is required. In finding good initial approximations, a great advance was recently achieved by developing an efficient non-iterative method of significant practical importance, originally proposed by Yun [65]. Yun's method is based on numerical integration briefly referred to as NIM, where tanh, arctan and signum functions are involved. The NIM requires neither any knowledge of the derivative $f(x)$ nor any iterative process. Handling non-pathological cases it is not necessary to have a close approximation to the zero; instead, a real interval (not necessarily tight) that contains the root (so-called inclusion interval) is sufficient. For illustration, to find an initial approximation x_0 of the zeros $\alpha = -1.4044916, 1.4044916$ of the function $h(x) = \sin(x)^2 - x^2 + 1$ isolated in the interval $[-5, 5]$, we employed Yun's algorithm with the statement taking $m = 250, a = -1, b = 2$, and found very good approximation $x_0 = 1.40449$. The graph of function h is plotted in Fig. 16.

Table 7 Comparison evaluation function and efficiency index of the proposed method with other schemes for f_1, f_2, f_3 and f_4

Without memory methods	EF	Iter	COC	EI	With memory methods	EF	Iter	COC	EI
$f_1(x) = x\log(1 + x\sin(x)) + e^{-1+x^2+x\cos(x)}\sin(\pi x)$,					$\alpha = 0, \quad x_0 = 0.3$				
AM	3	5	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	3	3.5252	1.87755
NM	2	7	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4282	1.55827
HM	3	5	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	4	3.0000	1.44225	SecM, $x_1 = 0.6$	2	15	1.6181	1.61808
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5681	1.88894
RWBM, $\beta_0 = 1$	3	4	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61803
MM	3	5	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	3	7.5214	1.95929
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	4	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	3	1.0000	1.00000
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = -1, \lambda_0 = 0.1, \beta_0 = 5$	4	3	15.5250	1.98499
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0054	2.00135
$f_2(x) = \sin(5x)e^x - 2$,					$\alpha = 1.36, \quad x_0 = 1$				
AM	3	1000	1.0000	1.00000	KM, $\beta_0 = \xi_0 = 0.1$	2	3	3.5149	1.87451
NM	2	15	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	5	0.0000	0.00000
HM	3	7	3.0000	1.44225	DPM	2	4	0.0000	0.00000
ChM	3	3	3.0000	1.44225	SecM, $x_1 = 1.2$	2	17	1.6181	1.61808
SMM	3	4	0.0000	0.00000	DM, $\gamma_0 = p_0 = 0.1$	3	4	1.0205	1.00679
RWBM, $\beta_0 = 1$	3	4	0.0000	0.00000	WM, $\lambda_0 = 0.1$	3	5	4.2361	1.61803
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	5	7.4405	1.95224
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	6	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	6	13.8020	1.92746
CLMTM, H_1, G_1	4	4	0.0000	0.00000	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	4	12.0000	1.86121
LSSSM	4	In	div	div	DPPM, $\gamma_0 = -0.1$	4	3	1.0000	1.00000
GKM	5	3	0.000	0.00000	LAM, $\gamma_0 = 0.01, p_0 = -1, \lambda_0 = 0.1, \beta_0 = 5$	4	5	15.9080	1.99712
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	3.9988	1.99970
$f_3(x) = 1 + \frac{1}{x^4} - \frac{1}{x} - x^2$,					$\alpha = 1, \quad x_0 = 1.4$				
AM	3	6	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	3	3.5862	1.89373
NM	2	5	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4062	1.55120
HM	3	5	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	5	3.0000	1.44225	SecM, $x_1 = 1.5$	2	16	1.6181	1.61808
SMM	3	3	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5592	1.88658
RWBM, $\beta_0 = 1$	3	4	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61804
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	3	7.4973	1.95720
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	168179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	3	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	3	10.0000	1.77828
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5120	1.98457
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	3.9984	1.99960
$f_4(x) = (x - 2)(x^{10} + x + 2)e^{-5x}$,					$\alpha = 2, \quad x_0 = 2.3$				
AM	3	1500	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	3	3.4123	1.84724
NM	2	5	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4681	1.57100
HM	3	4	3.0000	1.44225	DPM	2	5	3.0000	1.73205
ChM	3	5	3.0000	1.44225	SecM, $x_1 = 1.8$	2	17	1.6181	1.61808
SMM	3	3	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5877	1.89412
RWBM, $\beta_0 = 1$	3	4	4.0000	1.58740	WM, $\lambda_0 = 1$	3	6	4.2361	1.61803
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	3	7.5395	1.96086
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	5	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121

Table 7 (continued)

Without memory methods	EF	Iter	COC	EI	With memory methods	EF	Iter	COC	EI
LSSSM	4	4	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	3	1.0000	1.00000
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5130	1.98460
SSSLM, *method* 6	5	4	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0142	2.00355

$$\boxed{x_0 = 0.5 * (a + b + Sign[f[a]] * NIntegrate[\tanh[m * f[x]], \{x, a, b\}])}$$

Alghorithm Yun

Remark 9 By changing a, b, and m, different values are obtained for this description: if $a = -1, b = 2$, and $m = 6$ the output of the algorithm is 1.40143. If $a = -2, b = 0$, and $m = 16$ the output of the algorithm is -1.40408. If $a = -2, b = 0$, and $m = 16000$, the output of the algorithm is -1.40457 and so on.

4

Using the command *FindRoot* and assigning the function to the two functions, then draw both functions in a concatenated graph.

The command *WorkingPrecision* specifies the accuracy of the operation. For example, if we want to find the root of equation $f_2(x) = \sin(5x)e^x - 2$, we rewrite it like this $f_{20}(x) = \sin(5x)e^x, f_{21}(x) = 2$. Then, first, by plotting the function in interval $[-2, 2]$ and then using the code given below, in the package Mathematica, the approximate value of the root can be determined.

```
FindRoot[e^xsin[5x] == 2, t, 1, WorkingPrecision− > 50]
```

Below is the program output and its graph in Fig. 17.

```
{x− > 1.3639731802637126891832999034292974589390644240412}
```

Conclusion

In this work, we developed a new kind of with memory methods for solving nonlinear equations. Convergence analysis proves that these new derivative-free methods preserve their order of convergence. To this end, based on Newton's interpolatory polynomial of different degrees. One should note that the computational accuracy strongly depends on the structures of the iterative methods, the sought zero and the test functions as well as good initial approximations. In general, in Tables 4, 5, 6, 7, 8 and 9 we have examined some methods with different kinds of convergence order. It is observed that these methods support their theoretical aspects. The last column of tables show computational efficiency index defined by $\mathrm{EI} = \mathrm{COC}^{1/n}$, where n number of function evaluations per iteration. The numerical results show that proposed method is very useful to find an acceptable approximation of the exact solution of nonlinear equations, specially when the function is non-differentiable. In fact, we have contributed further to the development of the theory of iteration processes and propose a new accurate and efficient higher-order derivative-free method for solving nonlinear equations numerically. In other words, the efficiency index of the proposed family with memory is $\mathrm{EI} = 4^{1/2} = 2$, which is much better than optimal one until six-point optimal methods without memory having efficiency indexes $\mathrm{EI} = 2^{1/2} \simeq 1.414, \mathrm{EI} = 4^{1/3} \simeq 1.587, \mathrm{EI} = 8^{1/4} \simeq 1.681$, $\mathrm{EI} = 16^{1/5} \simeq 1.741, \mathrm{EI} = 32^{1/6} \simeq 1.781, \mathrm{EI} = 64^{1/7} \simeq 1.814$ respectively. Also, which are better than the other methods given in [1, 4–20], [22, 25–30, 32–41], [44–64, 66]. A comparison between the without memory, with memory and adaptive methods in terms of the maximum efficiency index alongside the number of steps per cycle are given in Fig 5. All algorithms are implemented using symbolic Math of MATHEMATICA [23]. Adaptive method with memory has minimum evaluation function, and not evaluation derivative, hence competes with methods existent with and without memory.

Table 8 Comparison evaluation function and efficiency index of the proposed method with other schemes for f_5, f_6, f_7 and f_8

Without memory methods	EF	Iter	COC	EI	With memory methods	EF	Iter	COC	EI
$f_5(x) = e^{x^3-x} - \cos(x^2-1) + x^3 + 1,$					$\alpha = -1, \quad x_0 = -1.3$				
AM	3	5	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5610	1.88706
NM	2	5	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4520	1.56549
HM	3	4	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	4	3.0000	1.44225	SecM, $x_1 = -1.6$	2	18	1.6181	1.61808
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5674	1.88876
RWBM, $\beta_0 = 1$	3	4	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61803
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	3	7.5223	1.95937
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	3	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	4	1.0000	1.00000
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5100	1.98451
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0000	2.00000
$f_6(x) = \frac{-5x^2}{2} + x^4 + x^5 + \frac{1}{1+x^2},$					$\alpha = 1, \quad x_0 = 1.3$				
AM	3	6	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5609	1.88703
NM	2	7	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.2545	1.50151
HM	3	5	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	5	3.0000	1.44225	SecM, $x_1 = 0.8$	2	16	1.61825	1.61825
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.4986	1.87045
RWBM, $\beta_0 = 1$	3	4	4.0000	1.58740	WM, $\lambda_0 = 1$	3	6	4.23606	1.61803
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	3	7.5090	1.95822
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	5	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	3	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	3	10.0000	1.77828
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5190	1.9848
SSSLM, *method* 6	5	4	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0000	2.00000
$f_7(x) = \log(1+x^2) + e^{-3x+x^2}\sin(x),$					$\alpha = 0, \quad x_0 = 0.3$				
AM	3	6	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5622	1.88738
NM	2	5	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.3980	1.54855
HM	3	5	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	5	3.0000	1.44225	SecM, $x_1 = 0.5$	2	16	1.61803	1.61803
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5659	1.88836
RWBM, $\beta_0 = 1$	3	4	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61808
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	3	7.5122	1.95849
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	4	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	4	10.0000	1.77828
GKM	5	2	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5110	1.98454
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	3.9996	1.99990
$f_8(x) = x^3 + 4x^2 - 10,$					$\alpha = 1.3652, \quad x_0 = 1$				
AM	3	4	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.7320	1.93184
NM	2	5	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4693	1.57140
HM	3	4	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	4	3.0000	1.44225	SecM, $x_1 = 1.1$	2	18	1.61803	1.61803
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.7320	1.93184
RWBM, $\beta_0 = 1$	3	6	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61804
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	4	9.0000	2.08008
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	4	15.0000	1.96799
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121

Table 8 (continued)

Without memory methods	EF	Iter	COC	EI	With memory methods	EF	Iter	COC	EI
LSSSM	4	4	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	4	10.0000	1.77828
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	16.0000	2.00000
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0000	2.00000

Table 9 Comparison evaluation function and efficiency index of the proposed method with other schemes for f_9, f_{10}, f_{11} and f_{12}

Without memory methods	EF	Iter	COC	EI	With memory methods	EF	Iter	COC	EI
$f_9(x) = x\log(1-\pi+x^2) - \frac{1+x^2}{1+x^3}\sin(x^2) + \tan(x^2)$,					$\alpha = \sqrt{\pi}, \quad x_0 = 1.7$				
AM	3	5	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5614	1.88717
NM	2	5	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4452	1.56637
HM	3	4	3.0000	1.44225	DPM	2	4	3.0000	1.73205
ChM	3	4	3.0000	1.44225	SecM, $x_1 = 1.5$	2	16	1.61803	1.61803
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5698	1.88939
RWBM, $\beta_0 = 1$	3	6	4.0000	1.58740	WM, $\lambda_0 = 1$	3	6	4.2361	1.61803
MM	3	3	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	4	7.5321	1.96022
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0000	1.93434
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	3	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	3	10.0000	1.77828
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5140	1.98464
SSSLM, *method* 6	5	3	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	3.9996	1.99990
$f_{10}(x) = (-1+2i) + \frac{1}{x} + x + \sin(x)$,					$\alpha = 0.28860 - 0.2422i, \quad x_0 = \frac{-i}{2}$				
AM	3	5	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5594	1.88664
NM	2	6	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.5471	1.59596
HM	3	4	3.0000	1.44225	DPM	2	5	3.0000	1.73205
ChM	3	4	3.0000	1.44225	SecM, $x_1 = -i$	2	21	1.61803	1.61803
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.6273	1.90455
RWBM, $\beta_0 = 1$	3	5	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61803
MM	3	4	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	4	7.3943	1.94819
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	3	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0020	1.93441
CLMTM, H_1, G_1	4	3	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	3	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	3	10.0000	1.77828
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5090	1.98448
SSSLM, *method* 6	5	8	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	3.9992	1.99980
$f_{11}(x) = (x-2)(x^6+x^3+1)e^{-x^2}$,					$\alpha = 2, \quad x_0 = 1.8$				
AM	3	8	3.0000	1.44225	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5616	1.88722
NM	2	7	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	2.4323	1.55958
HM	3	5	3.0000	1.44225	DPM	2	5	3.0000	1.73205
ChM	3	25	1.0000	1.00000	SecM, $x_1 = 2.2$	2	18	1.61803	1.61803
SMM	3	4	4.0000	1.58740	DM, $\gamma_0 = p_0 = 0.1$	3	4	3.5639	1.88783
RWBM, $\beta_0 = 1$	3	5	4.0000	1.58740	WM, $\lambda_0 = 1$	3	5	4.2361	1.61803
MM	3	6	4.0000	1.58740	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	4	7.5314	1.96016
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	7	8.0000	1.68179	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	3	14.0000	1.93434
CLMTM, H_1, G_1	4	8	8.0000	1.68179	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	3	12.0000	1.86121
LSSSM	4	4	8.0000	1.68179	DPPM, $\gamma_0 = -0.1$	4	4	1.0000	1.00000
GKM	5	3	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5140	1.98464
SSSLM, *method* 6	5	4	16.0000	1.74110	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0000	2.00000

Table 9 (continued)

Without memory methods	EF	Iter	COC	EI	With memory methods	EF	Iter	COC	EI
$f_{12}(x) = e^{x^2-1}\sin(x) + \cos(2x) - 2,$					$\alpha = 1.44, \quad x_0 = 1.1$				
AM	3	11	1.0000	1.00000	KM, $\beta_0 = \xi_0 = 0.1$	2	5	3.5659	1.88836
NM	2	9	2.0000	1.41421	TrM, $\lambda_0 = 0.1$	2	4	1.0357	1.01769
HM	3	5	3.0000	1.44225	DPM	2	5	3.0001	1.73208
ChM	3	7	3.0000	1.44225	SecM, $x_1 = 1.6$	2	18	1.61803	1.61803
SMM	3	14	1.0000	1.00000	WZQM, $\gamma_0 = \lambda_0 = \beta_0 = 0.1$	3	4	7.5424	1.96112
RWBM, $\beta_0 = 1$	3	5	0.0000	0.0000	WM, $\lambda_0 = 1$	3	19	4.2361	1.61803
MM	4	5	1.0000	1.00000	DM, $\gamma_0 = p_0 = 0.1$	3	In	div	div
CNM, $\alpha_0 = \gamma_0 = A_0 = 1$	4	30	1.0000	1.00000	LSSAKM, $\gamma_0 = p_0 = \lambda_0 = 1, \beta_0 = 0.01$	4	4	0.0000	0.00000
CLMTM, H_1, G_1	4	5	div	div	LSGAM, $a_0 = 1, \beta_0 = 0.01$	4	In	div	div
LSSSM	4	In	div	div	DPPM, $\gamma_0 = -0.1$	4	4	9.9999	1.77827
GKM, $\beta_0 = \lambda_0 = 1$	5	4	16.0000	1.74110	LAM, $\gamma_0 = 0.01, p_0 = \lambda_0 = \beta_0 = 0.1$	4	3	15.5009	1.98422
SSSLM, *method* 6	5	18	1.0000	1.00000	TLAM, $\beta_0 = \xi_0 = 0.1$	2	5	4.0000	2.00000

Acknowledgements We would like to deeply thank the editor, Professor Maleknejad, as well as the respected referees' for their valuable comments. This work was supported by the Islamic Azad University, Hamedan Branch and Central Tehran Branch.

Compliance with ethical standards

Conflicts of interest The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. Abbasbandy, S.: Modified homotopy perturbation method for nonlinear equations and comparison with Adomian decomposition method. Appl. Math. Comput. **172**, 431–438 (2006)
2. Aberth, O.: Iteration methods for finding all zeros of a polynomial simultaneously. Math. Comput. **27**, 339–344 (1973)
3. Alefeld, G.: Verified numerical computation for nonlinear equations. Jpn. J. Ind. Appl. Math. **26**(2–3), 297–315 (2009)
4. Bassiri, P., Bakhtiari, P., Abbasbandy, S.: A new iterative with memory class for solving nonlinear equations. Int. J. Ind. Math. **8**(3), 225–229 (2016)
5. Behl, R., Cordero, A., Motsa, S.S., Torregrosa, J.R.: Stable high-order iterative methods for solving nonlinear models. Appl. Math. Comput. **303**, 70–88 (2017)
6. Bi, W., Wu, Q., Ren, H.: A new family of eighth-order iterative methods for solving nonlinear equations. Appl. Math. Comput. **214**, 236–245 (2009)
7. Chun, C.: Some fourth-order iterative methods for solving nonlinear equations. Appl. Math. Comput. **195**, 454–459 (2008)
8. Chun, C., Neta, B.: An analysis of a new family of eighth-order optimal methods. Appl. Math. Comput. **245**, 86–107 (2014)
9. Cordero, A., Hueso, J.L., Martinez, E., Torregrosa, J.R.: Generating optimal derivative free iterative methods for nonlinear equations by using polynomial interpolation. Math. Comput. Model. **57**, 1950–1956 (2013)
10. Cordero, A., Lotfi, T., Bakhtiari, P., Torregrosa, J.R.: An efficient two-parametric family with memory for nonlinear equations. Numer. Algorithms **68**(2), 323–335 (2014)
11. Cordero, A., Lotfi, T., Khoshandi, A., Torregrosa, J.R.: An efficient Steffensen-like iterative method with memory. Bull. Math. Soc. Sci. Math. Roum Tome **58**(1), 49–58 (2015)
12. Cordero, A., Lotfi, T., Mahdiani, K., Torregrosa, J.R.: Two optimal general classes of iterative methods with eighth-order. Acta Appl. Math. **134**(1), 61–74 (2014)
13. Cordero, A., Lotfi, T., Torregrosa, J.R., Assari, P., Mahdiani, K.: Some new bi-accelerator two-point methods for solving nonlinear equations. Comput. Appl. Math. **35**, 251–267 (2016)
14. Dehghan, M., Hajarian, M.: Some derivative free quadratic and cubic convergence iterative formulas for solving nonlinear equations. Comput. Appl. Math. **29**(1), 19–30 (2010)
15. Dzunic, J.: On efficient two-parameter methods for solving nonlinear equations. Numer. Algorithms **63**, 549–569 (2013)
16. Dzunic, J., Petkovic, M.S.: A cubically convergent Steffensen-like method for solving nonlinear equations. Appl. Math. Lett. **25**, 1881–1886 (2012)
17. Dzunic, J., Petkovic, M.S., Petkovic, L.D.: Three-point methods with and without memory for solving nonlinear equations. Appl. Math. Comput. **218**, 4917–4927 (2012)
18. Ezzati, R., Saleki, F.: On the construction of new iterative methods with fourth-order convergence by combining previous methods. Int. Math. Forum. **6**(27), 1319–1326 (2011)
19. Geum, Y.H., Kim, Y.I.: A biparametric family optimally convergent sixteenth-order multipoint methods with their fourth-step weighting function as a sum of a rational and a generic two-variable function. J. Comput. Appl. Math. **235**, 3178–3188 (2011)
20. Guo, Q.W., Qian, Y.H.: New efficient optimal derivative-free method for solving nonlinear equations. Int. J. Math. Comput. Sci. **1**(3), 102–110 (2015)
21. Halley, E.: A new, exact and easy method of finding the roots of equations generally and that without any previous reduction. Philos. Trans. R. Soc. Lond. **18**, 136–148 (1694)
22. Hansen, E., Patrick, M.: A family of root finding methods. Numer. Math. **27**, 257–269 (1977)

23. Hazrat, R.: Mathematica a Problem-Centered Approach. Springer, London (2010)
24. http://www.mathematica.stackexchange.com/questions/5663/about-multi-root-search-inmathematica-for-transcendental-equations?lq=1
25. Jarratt, P.: Some fourth order multipoint methods for solving equations. Math. Comput. **20**, 434–437 (1966)
26. Kansal, M., Kanwar, V., Bhatia, S.: Efficient derivative-free variants of Hansen–Patrick's family with memory for solving nonlinear equations. Numer. Algorithms **73**, 1017–1036 (2016)
27. Kanwar, V., Bala, R., Kansal, M.: Some new weighted eighth-order variants of Steffensen-King's type family for solving nonlinear equations and its dynamics. SeMa J. **74**(1), 75–90 (2016)
28. Khaksar Haghani, F.: A modified Steffensen's method with memory for nonlinear equations. Int. J. Math. Mod. Comput. **5**(1), 41–48 (2015)
29. Kou, J., Li, Y.: The improvements of Chebyshev–Halley methods with fifth-order convergence. Appl. Math. Comput. **188**, 143–147 (2007)
30. Kreetee, D., Babajee, R., Thukral, R.: On a 4-point sixteenth-order King family of iterative methods for solving nonlinear equations. Int. J. Math. Sci. **2012**, 1–13 (2012)
31. Kung, H.T., Traub, J.F.: Optimal order of one-point and multipoint iteration. J. Assoc. Comput. Mach. **21**(4), 643–651 (1974)
32. Li, X., Mu, C., Ma, J., Wang, C.: Sixteenth-order method for nonlinear equations. Appl. Math. Comput. **215**, 3754–3758 (2010)
33. Lotfi, T., Assari, P.: New three-and four-parametric iterative with memory methods with efficiency index near 2. Appl. Math. Comput. **270**, 1004–1010 (2015)
34. Lotfi, T., Mahdiani, K., Noori, Z., Khaksar Haghani, F., Shateyi, S.: On a new three-step class of methods and its acceleration for nonlinear equations. Sci. World J. **2014**, 1–9 (2014)
35. Lotfi, T., Soleymani, F., Ghorbanzadeh, M., Assari, P.: On the construction of some tri-parametric iterative methods with memory. Numer. Algorithms **70**(4), 835–845 (2015)
36. Lotfi, T., Soleymani, F., Shateyi, S., Assari, P., Khaksar Haghani, F.: New mono- and biaccelerator iterative methods with memory for nonlinear equations. Abstr. Appl. Anal. **14**, 1–8 (2014)
37. Lotfi, T., Tavakoli, E.: On a new efficient Steffensen-like iterative class by applying a suitable self-accelerator parameter. Sci. World J. **2014**, 1–9 (2014)
38. Lui, Z., Zhang, H.: Steffensen-type method of super third-order convergence for solving nonlinear equations. J. Appl. Math. Phys. **2**, 581–586 (2014)
39. Maheshwari, A.K.: A fourth order iterative method for solving nonlinear equations. Appl. Math. Comput. **211**, 383–391 (2009)
40. Matinfar, M., Aminzadeh, M., Asadpour, S.: A new three-step iterative method for solving nonlinear equations. J. Math. Ext. **6**(1), 29–39 (2012)
41. Neta, B.: A sixth order family of methods for nonlinear equations. Int. J. Comput. Math. **7**, 157–161 (1979)
42. Ortega, J.M., Rheinboldt, W.C.: Iterative Solutions of Nonlinear Equations in Several Variables. Academic Press, New York (1970)
43. Ostrowski, A.M.: Solution of Equations and Systems of Equations. Academic press, New York (1960)
44. Petkovic, M.S.: On a general class of multipoint root-finding methods of high computational efficiency. SIAM J. Numer. Anal. **47**(6), 4402–4414 (2010)
45. Petkovic, M.S., Dzunic, J., Neta, B.: Interpolatory multipoint methods with memory for solving nonlinear equations. Appl. Math. Comput. **218**, 2533–2541 (2011)
46. Petkovic, M.S., Dzunic, J., Petkovic, L.D.: A family of two-point with memory for solving nonlinear equations. Appl. Anal. Discrete Math. **5**, 298–317 (2011)
47. Petkovic, M.S., Neta, B., Petkovic, L.D., Dzunic, J.: Multipoint Methods for Solving Nonlinear Equations. Elsevier, Amsterdam (2013)
48. Petkovic, M.S., Ilic, S., Dzunic, J.: Derivative free two-point methods with and without memory for solving nonlinear equations. Appl. Math. Comput. **217**, 1887–1895 (2010)
49. Petkovic, M.S., Sharma, J.R.: On some efficient derivative-free method with memory for solving system nonlinear equations. Numer. Algorithms **71**, 457–474 (2016)
50. Ren, H., Wu, Q., Bi, W.: A class of two-step Steffensen type methods with fourth-order convergence. Appl. Math. Comput. **209**, 206–210 (2009)
51. Sharifi, S., Siegmund, S., Salimi, M.: Solving nonlinear equations by a derivative-free form of the King's family with memory. Calcolo **53**(2), 201–215 (2015)
52. Sharifi, S., Salimi, M., Siegmund, S., Lotfi, T.: A new class of optimal four-point methods with convergence order 16 for solving nonlinear equations. Math. Comput. Simul. **119**(c), 69–90 (2016)
53. Sharma, J.R., Guha, R.K., Gupta, P.: Some efficient derivative free methods with memory for solving nonlinear equations. Appl. Math. Comput. **219**(2), 699–707 (2012)
54. Singh, A., Jaiswal, J.P.: A class of optimal eighth-order Steffensen-type iterative methods for solving nonlinear equations and their basins of attraction. Appl. Math. Inf. Sci. **10**(1), 251–257 (2016)
55. Soleymani, F.: Some optimal iterative methods and their with memory variants. J. Egypt. Math. Soc. **2013**, 1–9 (2013)
56. Soleymani, F., Lotfi, T., Tavakoli, E., Khaksar Haghani, F.: Several iterative methods with memory using self-accelerators. Appl. Math. Comput. **254**, 452–458 (2015)
57. Soleymani, F., Mousavi, B.S.: On novel classes of iterative methods for solving nonlinear equations. Zh. Vychisl. Mat. Mat. Fiz. **52**(2), 214–221 (2012)
58. Steffensen, J.F.: Remarks on iteration. Scand. Aktuarietidskr **16**, 64–72 (1933)
59. Taher-Khani, S.: A note on the paper "A new general eighth-order family of iterative methods for solving nonlinear equations". Math. Sci. **8**(123), 1–3 (2014)
60. Thukral, R.: New sixteenth-order derivative-free methods for solving nonlinear equations. Am. J. Comput. Appl. Math. **2**(3), 112–118 (2012)
61. Traub, J.F.: Iterative Methods for the Solution of Equations. Prentice Hall, New York (1964)
62. Ullah, M.Z., Kosari, S., Soleymani, F., Khaksar Haghani, F., Al-Fhaid, A.S.: A super-fast tri-parametric iterative method with memory. Appl. Math. Comput. **289**, 486–491 (2016)
63. Wang, X.: An Ostrowski-type method with memory using a novel self-accelerating parameter. J. Comput. Appl. Math. **330**, 1–18 (2017)
64. Wang, X., Zhang, T., Qin, Y.: Efficient two-step derivative-free iterative methods with memory and their dynamics. Int. J. Comput. Math. **93**(8), 1–27 (2015)
65. Yun, B.I.: Iterative methods for solving nonlinear equations with finitely any roots in an interval. J. Comput. Appl. Math. **236**, 3308–3318 (2012)
66. Zheng, Q., Li, J., Huang, F.: An optimal Steffensen-type family for solving nonlinear equations. Appl. Math. Comput. **217**, 9592–9597 (2011)