Swarm and Evolutionary Computation 4 (2012) 33–43

Contents lists available at SciVerse ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Solving nonlinear equations systems with a new approach based on invasive weed optimization algorithm and clustering

Ebrahim Pourjafari, Hamed Mojallali [*]

Electrical Engineering Department, Faculty of Engineering, University of Guilan, Rasht, P.O. Box 3756, Iran

ARTICLE INFO

Article history:
Received 4 September 2011
Received in revised form
25 November 2011
Accepted 8 December 2011
Available online 22 December 2011

Keywords:
Solving nonlinear equations
Evolutionary optimization
Invasive weed optimization
Clustering

ABSTRACT

A novel optimization-based method for solving systems of nonlinear equations is proposed. The method employs Invasive Weed Optimization (IWO) for solving nonlinear systems and can find all real and complex roots of a system and also detect multiplicity. The proposed solver consists of two parts: a two-phase root-finder that detects the solutions of a system using IWO algorithm, and an alteration technique which creates repulsion areas around previously found roots. The proposed solver determines all solutions using successive runs of the two-phase root-finder along with the alteration technique. Several illustrative examples together with three examples of engineering applications are provided to demonstrate merits of our proposed algorithm in solving nonlinear equations systems.

1. Introduction

Nonlinear equations are ubiquitous problems in mathematics as in engineering fields such as electrical engineering, robotics, mechanical and chemical engineering. Therefore, solving either single nonlinear equations or systems of nonlinear equations has great importance in both mathematics and engineering.

There are numerous traditional approaches such as Muller's method, the secant method and the Newton method [1] for solving nonlinear equations. But, these methods show many defects in solving nonlinear equations. They are very sensitive to the choice of initial values and may show oscillatory behavior or may even diverge in case when a good initial value close to the root is not chosen. Furthermore, most of these methods require continuously differentiable nonlinear equations, a condition that may not be fulfilled at least in few points of a nonlinear function. However, the major deficiency of traditional root-finders is that most of them are only able to find a single root of a system of nonlinear equations at one run of the algorithm.

To get rid of negative aspects associated with traditional methods, some approaches based on metaheuristic optimization algorithms have been proposed. Metaheuristic optimization methods such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Simulated Annealing (SA), etc. are widely used in optimization problems with no assumptions about the function being optimized such as smoothness, convexity or differentiability. Finding a single root of a nonlinear equation by means of evolutionary algorithms is a simple task. It only needs to minimize the absolute or square value of an equation with these methods to find one of its roots. Plenty of papers have utilized this approach to solve systems of nonlinear equations and have studied its applications in the engineering fields. For example, Wu et al. [2] utilized the hybrid social emotional optimization algorithm with the metropolis rule for solving nonlinear equations and Jaberipour et al. [3] used PSO to solve systems of nonlinear equations. Wu and Kang in [4] solved systems of nonlinear equations applying a parallel elite-subspace evolutionary algorithm. Dai et al. [5] mixed genetic algorithm and quasi-Newton method for solving a system of nonlinear equations needed for trimming helicopter.

Though the literature has employed evolutionary methods for finding a single root of nonlinear equations, research on designing a root-finder capable of locating all roots of nonlinear equations systems based on evolutionary methods is relatively scarce. Brits et al. [6] presented a method for finding all roots of systems of nonlinear equations based on a new PSO method called *nbest* PSO. Modifying the way of evaluating the fitness of particles and using a specified neighborhood architecture called *neighborhood best (nbest)* for updating the velocity vector of particles allows this method to search for more than one root. The main idea behind *nbest* PSO is to reward a particle for being close to a solution, not to penalize it for being away from a solution. But, the *nbest* method needs developing. Albeit particles are absorbed to more than one root. There is no mechanism to determine which particles represent the final answers of the root-finding problem. Furthermore, the method is highly sensitive to the proper selection of its neighborhood number and the number of particles.

[*] Corresponding author. Tel.: +98 131 6690276 8; fax: +98 131 6690271.
E-mail addresses: e.pourjafari@yahoo.com (E. Pourjafari), mojallali@guilan.ac.ir (H. Mojallali).

There are no obvious rules for selecting a proper neighborhood number. Also, some roots may remain undetected in case the number of roots is high and a large value is not assigned to the number of particles. Grosan and Abraham [7] described a root-finder with the ability to find all real solutions to systems of nonlinear equations. It considers a system of nonlinear equations as a multi-objective optimization algorithm and computes the pareto optimal solutions to this system using genetic algorithm. Then using genetic operators such as crossover and mutation, pareto optimal solutions are updated and this procedure will continue until a termination criterion is reached.

Hirsh et al. [8] utilized a modified metaheuristic GRASP algorithm to find all real solutions to systems of nonlinear equations. In the mentioned paper, all roots are found through multiple minimizations of an objective function which is the sum of the squared values of each equation associated with the system being investigated, using GRASP method. In each iteration, the objective function is altered; that is, a penalty term is assigned to each root that has already been found to create a repulsion area around it and avoid multiple locating of the earlier located solutions. Also Henderson et al. [9] proposed a root-finder with the ability to find all roots of systems of nonlinear equations using multiple optimizations. After locating one root in each iteration employing the metaheuristic Simulated Annealing, the objective function is modified with a method called polarization technique that creates a repulsion area around the newly found solution, to prevent multiple locating of a root. Though the basic idea is similar to [8], methods of altering the objective function are different. The polarization technique in [9] enables the algorithm to detect the multiplicity of roots—the polarization technique in [9] has the potential of detecting multiplicity, although the authors have not noticed it—but the algorithm in [8] does not have such ability.

Invasive Weed Optimization (IWO) is a stochastic optimization algorithm inspired from colonizing weeds which was first introduced by Mehrabian and Locus in [10]. Since its invention, IWO has been used in many applications like the design and optimization of antenna array [11–13], solving optimal control problems [14], training neural networks [15], analysis of pareto improvement models in electricity markets [16] and tuning of auto-disturbance rejection controller [17]. At the present paper, we aim to propose a novel method for finding all roots of nonlinear equations using IWO algorithm. The proposed root-finder is capable of finding all roots of nonlinear equations via successive runs of a two-phase root-finder along with modifying the objective function. The two-phase root search consists of the global and exact search phases. At the global search phase, the approximate locations of solutions are estimated through an IWO algorithm with some modifications. Then, at the exact search phase, after separating the estimated locations of solutions from each other using a clustering method, an IWO algorithm is applied to each cluster to determine the exact location of the root the cluster contains. After finishing the second phase, the objective function is modified in order to discard previously found roots from the objective function. The two-phase root searching and modifying the objective function run successively until all solutions are found.

Our proposed algorithm do not suffer drawbacks of the traditional root-finding methods such as sensitivity to initial guess, need to a continuous and differentiable function, finding only one solution, etc. Furthermore, employing IWO algorithm, finding more than one root in each iteration of the two-phase root-finder and using an effective method for modifying the objective function are three differences of the proposed method in this paper with other optimization-based root-finders. The mixture of these three features enables the proposed algorithm to locate solutions to systems of nonlinear equations much faster and more effective than its optimization-based counterparts. The proposed root-finder can locate all real solutions of nonlinear equations as well as their complex roots, the ability that the algorithms in [6–9] do not offer. Furthermore, the proposed root-finder is capable of detecting multiplicity, in contrast with methods in [6–8] which do not have this ability. Also the proposed solver can distinguish the very closely positioned roots much better than Brits et al. [6] and Henderson et al. [9]. In fact finding all real and complex roots, detecting multiplicity, the ability to solve hard problems such as equations with closely located roots, very good computational performance and high precision of the calculated roots are the excellent features of the proposed root-finder. The collection of these features cannot be entirely found in the other optimization-based root-finders have been proposed so far.

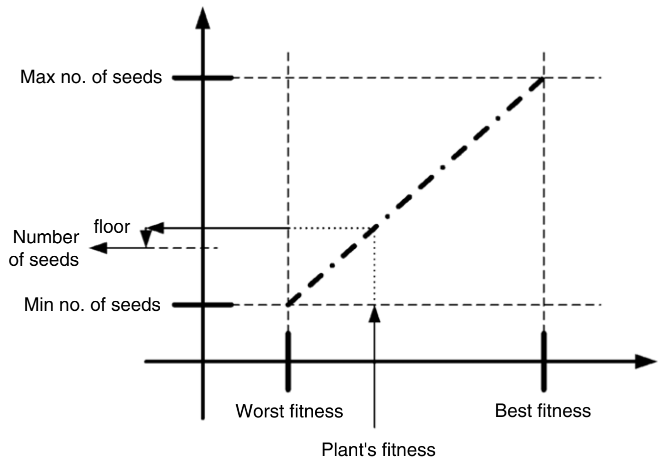

Fig. 1. Seed production procedure in a colony of weeds.

The paper is organized as follows: Section 2 describes IWO method and also shows how to transfer a root-finding problem to an optimization problem. The proposed root-finding algorithm is declared and discussed in Section 3. Section 4 provides several examples for showing how the algorithm works, demonstrating its capabilities and comparing it to other optimization-based solvers. Section 5 discusses about tuning the parameters of the algorithm. Section 6 provides three examples of real applications of the proposed solver and finally, Section 7 concludes the paper.

2. The optimization method and problem formulation

2.1. Invasive weed optimization

Invasive weed optimization is an evolutionary optimization algorithm inspired from invasive and robust nature of weeds in growth and colonizing. The algorithm can be stated in the following steps [10]:

- *Initialization*: A finite number of seeds are being dispread randomly on the d-dimensional search area as the initial solutions.
- *Reproduction*: Every plant is allowed to produce seeds based on its fitness and the colony's best and worst fitness: the number of seeds each plant produces increases linearly from minimum possible seed production for the worst fitness to the maximum number of seeds corresponding to the maximum fitness in the colony. Fig. 1 shows this procedure [10].
- *Spatial dispersal*: After reproduction step, the produced seeds are randomly distributed in the search space such that they abode near to the parent plant by normal distribution with mean equal to zero and varying variance. The standard deviation (SD), σ, of the normal distribution is reduced in each

iteration from an initial value $\sigma_{initial}$ to its final value σ_{final} by means of a nonlinear equation presented in Eq. (1):

$$\sigma_{iter} = \frac{(iter_{\max} - iter)^n}{(iter_{\max})^n}\left(\sigma_{initial} - \sigma_{final}\right) + \sigma_{final} \tag{1}$$

where $iter_{\max}$ is the maximum number of iterations, σ_{iter} is the standard deviation in iteration *iter* and n is the nonlinear modulation index.

- *Competitive exclusion:* Due to fast reproduction, after passing some iteration the number of produced plants in a colony reaches to its maximum $P_{\max}$. In this step, a competitive mechanism is activated for eliminating undesirable plants with poor fitness and allowing fitter plants to reproduce more seeds as expected.

The above process continues until the maximum number of iterations reached.

2.2. Transferring to an optimization problem

Consider the system of nonlinear equations expressed in Eq. (2):

$$\begin{cases} f_1(x_1, \ldots, x_i, \ldots, x_d) = 0 \\ \vdots \\ f_i(x_1, \ldots, x_i, \ldots, x_d) = 0 \\ \vdots \\ f_d(x_1, \ldots, x_i, \ldots, x_d) = 0 \end{cases} \tag{2}$$

where d is the dimension of the problem. x_i is the i-th independent variable and $f_i(.)$ is the i-th equation. The problem of computing one solution to Eq. (2) can be transferred to a global optimization problem as below:

$$\min_{x_i} y(x) = \sum_{i=1}^{d} |f_i(x_1, \ldots, x_i, \ldots, x_d)| \,. \tag{3}$$

It is obvious that with this alteration, the solutions to Eq. (2) are mapped onto the global minima of Eq. (3). Therefore, applying an optimization method to Eq. (3), one root of Eq. (2) can be found.

3. The proposed algorithm

After determining the objective function – or fitness function – in the previous section, it is time to locate the solutions of Eq. (2) based on the fitness function (3). This paper utilizes successive runs of a two-phase root-finder along with objective function alteration to find all roots of the problem. The two-phase root-finder is the core of the proposed algorithm which consists of two phases: The global search and the exact search. The global search seeks for approximate locations of solutions, followed by the exact search that finds the exact locations of roots. After finishing the two-phase root-searching, some solutions to the system of equations are determined, but some other ones might remain undetected. Therefore, the two-phase root-finder needs running more than one time to detect all solutions. Before another run of the two-phase root-finder, the fitness function is altered in order to preclude the solver from multiple locating of the previously found roots. After some iterations of the two-phase root-finder, all solutions to the system will be eventually determined. The procedure of the proposed solver is described in the following.

3.1. The two-phase root-finder

The two-phase root-finder involves two parts: The global search and exact search. At the global search, IWO algorithm is employed to find the approximate locations of roots. After finishing the global search, it is turn to the exact search. During this phase, two actions are done: first, regions where indicated as the approximate locations of the solutions during global search are separated from each other by means of a clustering method. Then, an IWO algorithm is applied to each cluster to determine the exact location of each root.

3.1.1. Global search

The value of $y(x)$ in Eq. (3) per each of its minima is equal to zero. Hence, if an optimization algorithm such as IWO is utilized for optimizing Eq. (3), all minimum points will have an equal chance to absorb weeds. After some iterations of IWO, plants abandon non-minimum points vacant and settle down around minima. Each minimum point, thus, has a chance to absorb some weeds. So, one can interpret these congested areas as the approximate locations of solutions to the system. It is mandatory for the global search to choose a small value as the maximum iterations of IWO, because IWO essentially is biased toward locating one global minimum. As long as IWO is considered to optimize Eq. (3) and allowed to run with a large value as its maximum number of iterations, it will be only able to find one global minimum. Eq. (3) may have more than global minimum. Therefore, it is essential to choose a small value as the maximum iterations during global search phase to prevent plants from assembling around only one minimum. In other words, with a small maximum of iterations, plants only have enough time to leave non-minimum points and settle down around minima, not to leave different minimums to randomly assemble around one minimum as the global minimum.

Since the exact locations of roots are not sought during global search, the standard deviation of IWO does not need reducing from a large initial value to a small final value. Thus, Eq. (1) is not used during global search and replaced with the constant standard deviation σ_{global}. After finishing the global search, the exact search runs.

3.1.2. Exact search

At this phase, the exact locations of roots are determined. Three actions are done during this phase. Clustering is the first one that clusters plants around each root and therefore, separates the area around each solution from other regions. The second action is to employ the IWO algorithm in each cluster to seek the exact coordination of the root the cluster contains. At the end of the exact search and as the third action, another clustering is applied to the centers of clusters that the necessity of this action is explained later. The three parts of the exact search are illustrated in the following.

(a) *Clustering*

Clustering helps plants around each approximate location of a root group together and become separated from other plants. Prior to clustering, it will be useful to eliminate plants with inappropriate fitness. This helps the algorithm to run faster, because the total number of clusters decreases. Since the fitness function value of a solution to Eq. (3) is zero, closeness to zero can be a suitable criterion to judge about a plant's fitness. Hence, the value *Fzero* is defined which is the maximum acceptable fitness of plants for clustering. Plants with fitness larger than *Fzero* are not allowed to get clustered.

After discarding undesirable ones, clustering of the remained plants is performed based on a modified version of the nearest neighborhood clustering method as follows.

- Create a cluster and set the plant with the best fitness as the center of this cluster. If no plant is remained due to having fitness value larger than *Fzero*, the clustering is terminated. Otherwise, do the followings.

- Define *Rclust* which is the clustering radius.
- Suppose P_i is the i-th plant in the colony and C_j is the center of the j-th cluster. Calculate the distance between P_i and all of the centers of clusters from Eq. (4):

$$dis(i,j) = \sqrt[2]{\frac{\left\|P_i - C_j\right\|^2}{d}} \tag{4}$$

where $\|.\|$ denotes the Euclidean distance between P_i and C_j and d shows dimension of x.
- If the distance between P_i and some centers of clusters is equal or less than *Rclust*, P_i belongs to that cluster which has the lowest distance with its center. Calculate the fitness of both P_i and the center of the cluster which contains P_i. If the fitness of P_i is better than the fitness of whose cluster center, set P_i as the new center of its cluster.
- If the distance between P_i and all of the centers of clusters is greater than *Rclust*, establish a new cluster and set P_i as its center.

(b) *Parallel search*

After finishing the clustering, plants around each approximate location of a root group together and become separated from other clusters. At this time, an IWO is applied to each cluster and the parallel search of IWOs in clusters, eventually will detect the exact coordination of each root in each cluster. These parallel optimizers are similar to the original IWO algorithm and their standard deviations vary in accordance with Eq. (1). Optimization parameters of IWOs such as maximum and minimum number of seeds and maximum number of plants are the same, but differ from the parameters of IWO during the global search. The parallel search continues until the maximum iterations $iter_{\max}^{exact}$ is reached.

It is worth noting that in case a large value is assigned to *Rclust* or some roots are located very closely, some clusters are likely to contain more than one root. The two-phase root-finder can only detect one root in each cluster in each run. Though, it is not a problem. The remained roots will be found at the next runs of the two-phase root-finder.

(c) *Final clustering*

If *Rclust* is selected too small, it is possible that plants around a specific solution are clustered in more than one group. In this case, if the centers of clusters, i.e. plants with the best fitness in clusters, were directly represented as the final solutions, the solver would find more than one solution for a specific root. Hence, after fulfilling the parallel search, the centers of clusters are calculated and then, another clustering is performed just between the centers. The possibility of multiple locating of a root due to multiple clustering around it completely vanishes after clustering the centers of clusters. After this clustering, the centers of clusters which their fitness is less than a small value e.g. $y(x) < 1e-4$ can be considered as the solutions to the system of nonlinear equations. The center of clusters with poorer fitness must be rejected, because they may be just local minima, not genuine solutions.

3.2. Objective function alteration

The two-phase root-finder can find a part of solutions to a system in its first run and needs other runs to locate all roots. Before another run of the two-phase root-finder, the objective function needs modifying to avoid multiple locating of the earlier found roots. The key idea is to create a repulsion area around the earlier located roots to exclude them from the set of global minimums of Eq. (3). Suppose x_0 is a solution to Eq. (3) found in the first run of the two-phase root-finder. One can modify the objective function for the next run of the two-phase root-finder using Eq. (5):

$$\min_x y_{mod}(x) = |\coth(\alpha\,\|x - x_0\|)| \times (y(x) + \varepsilon) \tag{5}$$

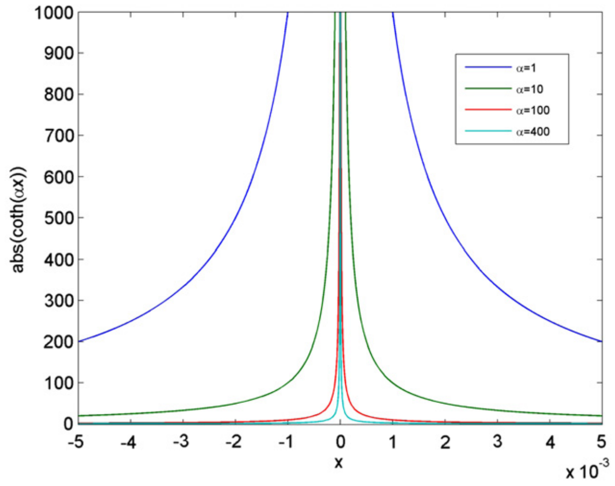

Fig. 2. The relation between α and the radius of repulsion area generated by $|\coth(\alpha x)|$.

where $\|.\|$ is the Euclidean distance like that in Eq. (4), ε is a very small positive constant or equal to zero and α is a positive coefficient equal or greater than 1 called density factor and used for adjusting the radius of the repulsion area.

$\lim_{x\to x_0} |\coth(\alpha\,\|x - x_0\|)| = \infty$ and $\lim_{\substack{x\to x_0\\0<\varepsilon\ll 1}} y(x) + \varepsilon = \varepsilon$. Therefore, if x_0 is not a root of multiplicity of $y(x)$, then $\lim_{x\to x_0} y_{mod}(x) = \infty$. Also if $\varepsilon = 0$, but $\lim_{x\to x_0} |\coth(\alpha\,\|x - x_0\|)| \to \infty$ faster than $\lim_{x\to x_0} y(x) \to 0$, then $\lim_{x\to x_0} y_{mod}(x) \gg 0$.

This means that in both cases, if the objective function is altered according to Eq. (5), x_0 is no longer a global minimum of $y_{mod}(x)$. That is a repulsion area is created around x_0 so that does not let the optimization algorithm find x_0 another time as a solution to Eq. (3). Albeit Eq. (5) creates a repulsion area around x_0, it does not affect other parts of the fitness function, because $\lim_{x\to\infty} |\coth(\alpha\,\|x - x_0\|)| = 1$.

The objective function for the j-th run of the two-phase root-finder can be altered according to Eq. (6):

$$\min_x y_{mod}^j(x) = |\coth(\alpha\,\|x - x_1\|)| \times \cdots \times |\coth(\alpha\,\|x - x_i\|)| \times (y(x) + \varepsilon) \tag{6}$$

where i is the number of detected roots until the j-th modification.

Though the basic idea for objective function modification is similar to the polarization technique in Henderson et al. [9], the alteration technique proposed in this paper is much more effective than its counterpart in [9]. The first superiority is because of the density factor α added to the repulsion generator. If some roots of the system are located very close together, the polarization technique in Henderson et al. [9] is likely to fail finding all of the closely located roots, as will be shown in the examples of the next section. The repulsion function in Eq. (6) does not undergo such a problem, because if there are some closely placed roots, one can easily raise α to avoid this problem. If α goes up, the radius of the repulsion area generated by $|\coth(\alpha x)|$ shrinks, as can be seen in Fig. 2. In other words, the greater α, the slenderer repulsion area, closely placed roots can be effectively found.

The other advantage of the repulsion generator used in this paper is due to multiplying the fitness function by *coth*, not dividing it by *atan*. $\lim_{x\to\infty} |\coth(x)| = 1$, so it does not disturb the objective function in point away from the formerly detected roots. $\lim_{x\to\infty} atan(x) = \frac{\pi}{2}$, thus, dividing the objective function by *atan* disposes most of the regions of the objective function $\frac{\pi}{2}$ times closer to zero. Multiple dividing the objective function by $\frac{\pi}{2}$ increases the risk of detecting some spurious solutions, because the fitness value of many points is now closer to zero than before.

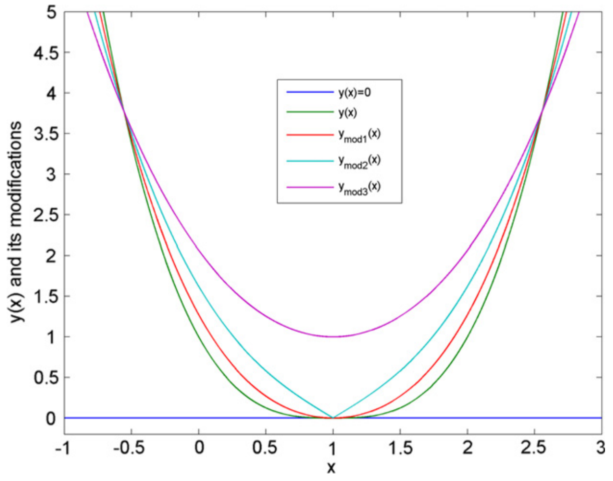

Fig. 3. The effects of successive fitness function alterations on a root of multiplicity 3.

Surprisingly, the algorithm is able to detect multiplicity. It is a direct result of the analytical alteration of the fitness function, not like the modification method has been used in [8] where a large positive constant is added to the objective function in the vicinity of the already detected roots, as if a tall wall is built around each root to physically preclude the optimizer from accessing to it.

x_0 is said to be a root of multiplicity i of $y(x)$ if $y(x0) = 0$ and $\frac{dy}{dx}|_{x_0} = \frac{d^2y}{dx^2}|_{x_0} = \cdots = \frac{d^{i-1}y}{dx^{i-1}}|_{x_0} = 0$. Suppose $\varepsilon = 0$ and the objective function $y(x)$ is being altered for the j-th time where $1 \le j < i$. Using L'ôpital's rule we have:

$$\lim_{\substack{x\to x_0\\1\le j<i}} y^j_{mod}(x) = \lim_{\substack{x\to x_0\\1\le j<i}} \left|\coth^j(\|x-x_0\|)\right| \times y(x)$$
$$= \lim_{\substack{x\to x_0\\1\le j<i}} \frac{y(x)}{\left|\tanh^j(\|x-x_0\|)\right|} = 0. \tag{7}$$

But in case $j = i$, again using L'Hôpital's rule:

$$\lim_{\substack{x\to x_0\\i=j}} y^j_{mod}(x) = \lim_{\substack{x\to x_0\\i=j}} \left|\coth^j(\|x-x_0\|)\right| \times y(x) = C \tag{8}$$

where C is a positive finite value. This means that if x_0 is a root of multiplicity i of $y(x)$, x_0 remains as the global minimum of Eq. (3) until $y(x)$ multiplied i times by $\coth(\|x - x_0\|)$. In other words, x_0 is detected i times as the solution to the system of nonlinear equation. Fig. 3 shows the effects of objective function modification on $y(x) = \left|(x-1)^3\right|$ which has a root of multiplicity 3 in $x = 1$. It can be seen in Fig. 3 until the third modification, $x = 1$ is still a root of $y(x) = 0$.

In applications where the solver is not supposed to detect multiplicity, just a little modification is needed to avoid the solver from discovering multiplicity. Two approaches can be considered. The first approach says if the Euclidean distance between the plant x and the previously found solution x_0 is less than a small value, e.g. 1e−3, instead of just one time multiplying $y(x)$ by $\coth(\|x - x_0\|)$ in Eq. (6), multiply $y(x)$ by $\coth(\|x - x_0\|)$ more than one time, until the value of the fitness function for the plant x becomes more than a pre-specified positive value, e.g. 1. This approach can create a repulsion area around x_0 in the first fitness function modification, just after the first detection of x_0. This is due to the fact that if x_0 is a root of multiplicity i of $y(x)$ and $j \ge i$, then $\lim_{x\to 0} \coth^j(\alpha\|x - x_0\|) \times y(x) = C$, Where C can be a positive value or infinity. That is if $y(x)$ is multiplied by $\coth(\|x - x_0\|)$ for $j \ge i$ times, x_0 is no longer a global minimum of $y(x)$.

The second approach adds a positive value, e.g. 1, to the fitness function of the plant x, if its Euclidean distance with the previously located root x_0 is less than a small value, e.g. 1e−3. The second approach is somewhat similar to the method has been proposed in [8] where adds a large gain to the fitness of the points in the vicinity of the detected roots.

After the every run of the two-phase root-finder, the fitness function is altered. This procedure continues until a termination criterion is met. The termination criterion for this algorithm can be finding a pre-determined number of roots—in case the number of roots is known in advance or finding just a pre-specified number of roots is desirable—or reaching to the maximum runs of the two-phase root-finder without finding any new solution.

The global search needs a population-based optimizer. Also the method used in the exact search phase implies this phase needs an optimizer to locally search around the approximate locations of roots found during global search. Among different metaheuristic optimizers, IWO seems to be the best choice for our proposed methodology, because it can completely fulfill the two requirements of the global and exact search phases mentioned above. The flowchart of the proposed solver is represented in Fig. 4.

Table 1
Parameters used for solving the first example.

Symbol	Quantity	Value
$iter^{glob}_{\max}$	max no. of iterations at global search	5
σ_{glob}	SD at global search	3
$iter^{exact}_{\max}$	Max no. of iterations at exact search	10
$\sigma_{initial}$	Initial value of SD at exact search	0.1
σ_{final}	Final value of SD at exact search	0.00001
$P^{exact}_{\max}$	Max no. of plants in each cluster	20
$Fzero$	Acceptable fitness for clustering	0.2

4. Illustrative examples

Several examples have been provided in this section in order to show the procedure of finding solutions with the proposed solver, demonstrate its ability of dealing with different types of problems, give its computational performance and compare it with other optimization-based root-finders.

4.1. The first example

The first example aims to show how the root-finder works and gives statistical data about the algorithm. The goal is to find the real roots of the nonlinear equation given by Eq. (9):

$$f(x) = \begin{cases} -\tan\left(x - \frac{\pi}{4}\right) & -6 \le x \le -2 \\ e^x \sin(10x) & -2 < x \le 2 \\ (x-5.5)(x-3)^3 & 2 < x \le 6. \end{cases} \tag{9}$$

Obviously, Eq. (9) is non-continuous and non-differentiable, so most of the analytical methods such as Newton–Raphson and bisection method are not applicable to this equation. Table 1 shows the parameters of the algorithm used for solving this equation and Table 2 lists the obtained roots. It can be seen from Table 2 that $x = 3$ is a root of multiplicity 3 of $f(x)$. Fig. 5 describes the procedure of solving Eq. (9) for a typical run of the proposed algorithm, where all of the 19 roots are found during three iterations. In Fig. 5, α is set to 5, and it can be seen that after each iteration of the two-phase root-finder the objective function $y(x)$ is altered whereby the previously found solutions are discarded. After the third detection of the root $x = 3$, $\lim_{x\to 3} \left|\coth^3(5\|x-3\|)\right| \times y(x) = 0.02$. Therefore, $x = 3$ is not any longer a global minimum of $y(x)$.

Fig. 6 compares the fitness function after the last alteration with dividing by $atan(x)$ and multiplying by $\coth(3x)$. $\lim_{x\to\infty} atan(x) = \frac{\pi}{2}$. As a result, if $atan(x)$ is used for modification, some points of $y(x)$ which are away from the solutions will be

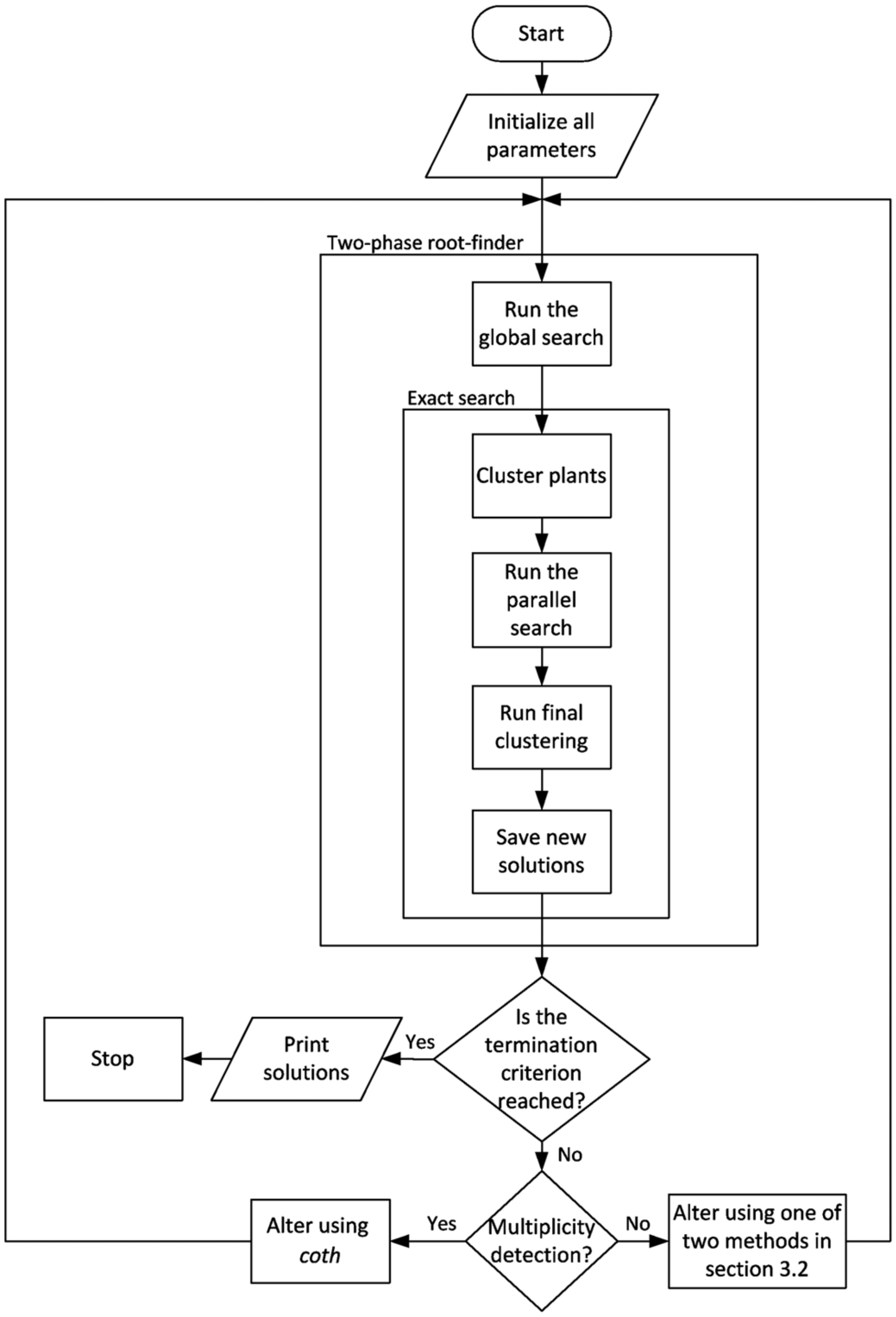

Fig. 4. The flowchart of the proposed solver.

19 times divided by $\frac{\pi}{2}$. Fig. 6 shows this adverse effect on $y_{mod}(x)$ where some points of $y_{mod}(x)$ become very close to zero. For example, for the points in $x \in [-6, -5]$ the fitness value of $y_{mod}(x)$ is less than 4e−3 that obviously increases the probability of finding spurious roots in this region. In contrast, since $\lim_{x\to\infty} |\coth(\alpha x)| = 1$, using this function for modifying the objective function does not tend to this problem, as can be seen in Fig. 6.

Table 3 shows the computational time and other performance obtained for the first example using MATLAB® 7.7 on a PC with Intel® core™ 2 Duo 2.26 GHz Processor and 2 GB of RAM. The termination criterion is to find all 19 roots of the equation and α is set to 10. The solver ran with different values of clustering radius *Rclust* and the maximum number of plants during the global search $P_{\max}^{global}$. There are four rows in Table 3 that respectively show the total computational time, the average number of iterations, the average time the solver spent on discovering each root and finally, the average number of solutions obtained during the first three iterations of the two-phase root-finder. It can be inferred that the best clustering radius for this example is $Rclust = 0.5$, although it is not a general rule and depends on the nonlinear

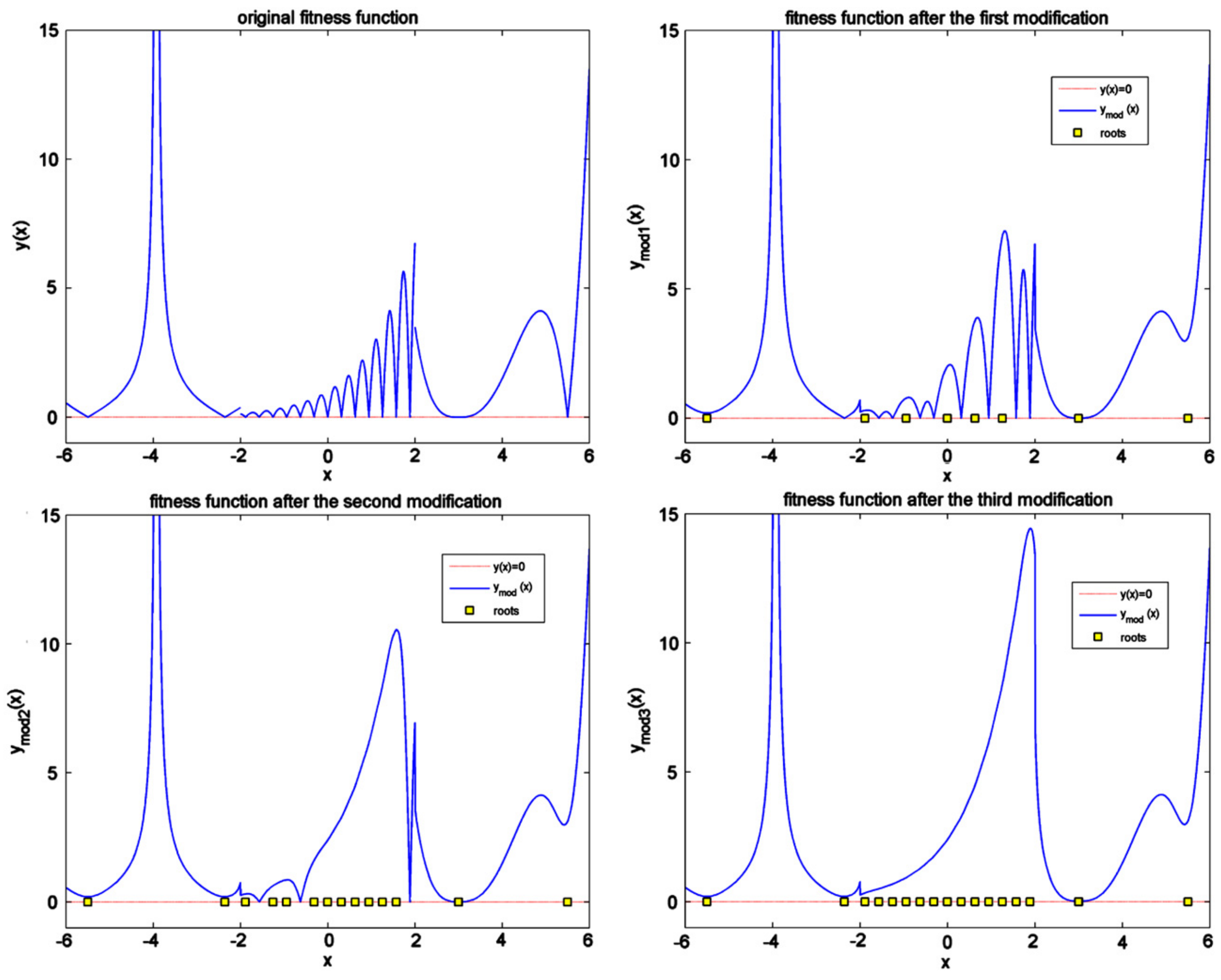

Fig. 5. The procedure of finding solutions to the first example through the proposed solver.

Table 2
Solutions obtained for the first example.

Solution	x	Value of the fitness function
1	−5.497787136599994	9.727274600200703e−24
2	−2.356194487701345	7.182144013531960e−09
3	−1.884955600064493	7.434653105541026e−07
4	−1.570796346117686	1.201114890307217e−08
5	−1.256637055923195	8.005175513396992e−09
6	−0.942477798131332	2.518412057464591e−07
7	−0.628318538391336	2.028107907994209e−07
8	−0.314159237048665	5.232697537039000e−07
9	−2.028107949126441e−8	1.013384567985147e−24
10	0.314159256341041	2.067793035448866e−07
11	0.628318570380946	1.568973418139226e−08
12	0.942477793072216	2.490999973299576e−09
13	1.256637054268276	1.234652931684284e−07
14	1.570796210561094	5.591400139591277e−06
15	1.884955592760783	7.711115432978450e−08
16	2.999999984271630	1.243737515634461e−23
17	3.000000007400790	4.016813186737704e−08
18	3.000000017071155	4.093655416950115e−08
19	5.499999965510734	3.997128177617975e−08

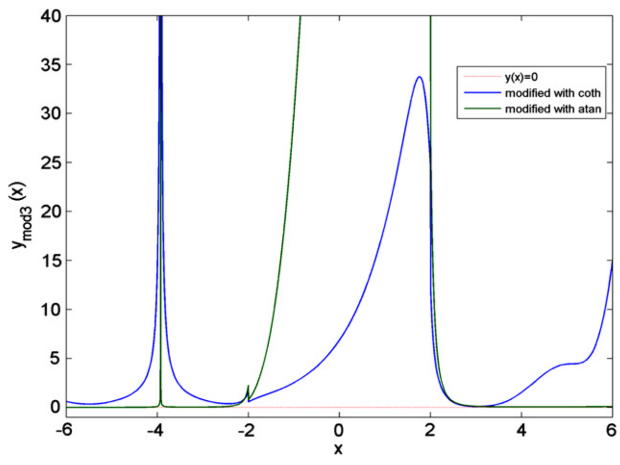

Fig. 6. Comparing the fitness function of the first example after modification by coth and atan.

equation being investigated. In addition, a general rule about $P_{\max}^{global}$ can be observed: as $P_{\max}^{global}$ increases, more solutions are detected in each iteration and consequently, the computational time dwindles. Setting $Rclust = 0.5$ and $P_{\max}^{global} = 200$, the best result, i.e. $T = 1.5941$ s, as the total computational time spent on finding 19 roots is achieved.

Table 4 shows the number of solutions detected with the proposed algorithm, if the termination criterion is to pass three iterations without finding any new solution. Data is available for different $P_{\max}^{global}$ values in two modes: with multiplicity detecting and without it. Both approaches that were presented in Section 3.2 for not discovering multiplicity worked well and there was not a significant difference between their computational performance. Results of Table 4 reinforce a general rule about the solver: setting

Table 3
Results obtained for the first example.

Performance[a]	$P_{\max}^{global} = 50$			$P_{\max}^{global} = 100$			$P_{\max}^{global} = 200$		
	$Rclust = 0.2$	$Rclust = 0.5$	$Rclust = 1$	$Rclust = 0.2$	$Rclust = 0.5$	$Rclust = 1$	$Rclust = 0.2$	$Rclust = 0.5$	$Rclust = 1$
Total CPU time (s)	2.5623	1.9957	2.6650	2.3710	1.7607	1.9617	1.6824	1.5941	2.3544
Average number of iterations	7.5	7.1	10.06	4.52	6.00	7.56	3.26	4.10	6.84
Average time per each solution (s)	0.1349	0.1050	0.1403	0.1248	0.0927	0.1032	0.0885	0.0839	0.1239
Solutions found during the first three iterations	16.3	15.12	11.42	17.92	16.86	12.22	18.86	17.92	12.52

[a] Results have been calculated over 50 executions of the proposed algorithm.

Table 4
Total number of solutions of the first example found by algorithm when the termination criterion is passing three iterations without finding a new solution.

Total number of solutions[a]	$P_{\max}^{global}$			
	50	100	200	400
With multiplicity detection (out of 19)	18.78	18.91	18.99	19
Without multiplicity detection (out of 17)	16.98	16.98	17	17

[a] Obtained from over 100 runs of the algorithm.

Table 5
Solutions to the truncated Weierstrass function ($N = 20$).

Solution	x	$f(x)$
1	0	0
2	1.888678276844590	−4.991051055311543e−08
3	3.731766320147167	1.241870996219455e−06
4	3.734889916912982	2.359739780655851e−07
5	3.737910519572671	−5.569230770020688e−08
6	3.739910524728320	2.238740843261197e−07
7	3.740847252602398	1.710304298001546e−08
8	4.549749746292466	1.527870271906054e−07
9	5.020090372428666	−3.572015453497286e−09

a large value for $P_{\max}^{global}$, the number of solutions discovered during the first iterations of the two-phase root-finder goes up.

4.2. The Weierstrass function

The second example is an interesting pathological function called the Weierstrass function [18] which is defined as:

$$f(t) = \sum_{k=1}^{N\to\infty} \lambda^{(s-2)k} \sin(\lambda^k t) \qquad (10)$$

where $1 < s < 2$ and $\lambda > 1$. The Weierstrass function is continuous everywhere, but surprisingly differentiable nowhere. This is because every point of this function oscillates more and more on small scales, but with higher-frequency oscillations. In other words, it is a fractal. Therefore the equation $f(t) = 0$ has infinite solutions. In this example, we aim to solve the truncated form of this equation with $N = 20$, $s = 1.1$ and $\lambda = 1.5$. This function is interesting for root-finding because as Fig. 7 shows, it has several local minima and five closely positioned roots laid in $t \in [3.7318, 3.7408]$. However it is a truncated version of the original function, the fractal behavior of the Weierstrass function is apparent. The diagram of this function in the interval [3.7318, 3.7408] as shown in Fig. 7(b) resembles that in the interval [0,6] in Fig. 7(a), although with less details and oscillations as a result of truncation. Fig. 7 also represents the positions of 9 roots obtained by the proposed solver in the range [0, 6] for the truncated Weierstrass function.

Nine obtained solutions are listed in Table 5 with $Rclust = 0.5$, $Fzero = 0.04$ and $P_{\max}^{global} = 50$, where other parameters are like Table 1. The computational time and average number of iterations spent on finding 9 solutions are expressed in Table 6 where α varies between 200 and 400. One can infer from Table 6 that there is an inverse relation between α and the computational time: as α goes up, the computational time shrinks. This relation, in turn, demonstrates merits of the density factor α in solving equations which have some closely located roots. In fact if α was set to a small value, the algorithm would not find all 9 solutions. A large density factor reduces the repulsion area around the already discovered roots and helps the algorithm find other solutions more easily.

Other root-finders were employed to solve this equation. Except Hirsh et al. [8] which found all roots with $\rho = 5e-4$ and $\beta = 2000$ as its repulsion radius and penalty term respectively, two other optimization-based root-finders Brits et al. [6] and Henderson et al. [9] did not represent satisfactory results. The main problem with these two methods was disability of finding all 5 solutions in the range [3.7318, 3.7408]. The root-finder in [9] could only detect two out of five solutions in this range, as a direct result of lack of the density factor. *nbest* PSO in [6] even went further and found some spurious solutions in points such as $t = 1.44$ or $t = 2.76$, which were originally local minima.

Analytical Newton–Raphson and Muller's method were applied to solve the truncated Weierstrass function. The aim was finding just one root in $t \in [0, 6]$. The failure rate of the Newton–Raphson method was 25% —differentiation was performed using numerical methods—while the failure rate of Muller's method was 27% for over 100 executions of each of these two algorithms.

4.3. Trigonometric equation

Consider the trigonometric function expressed in Eq. (11):

$$f(x) = \sin(0.2x)\cos(0.5x), \quad -50 \le x \le 50. \qquad (11)$$

The roots of this function lie in $x_0 = 5k\pi$ for $\sin(0.2x) = 0$ and $x_0 = (2k+1)\pi$ for $\cos(0.5x) = 0$. The objective is to compare the computational performance of our proposed solver based on IWO with three other optimization-based root-finders in [6,8,9] and also with our proposed method in case the genetic algorithm is used in it instead of IWO. Root-finders are supposed to locate the solutions to Eq. (11) in the interval [−50, 50]. Eq. (11) has 23 roots that four of them are roots of multiplicity 2 in $x = \pm 5\pi$ and $x = \pm 15\pi$. Hence, root-finders Brits et al. [6] and Hirsh et al. [8] are only able to detect 19 solutions. Parameters of the proposed solver used for solving Eq. (11) is similar to those were used for solving the Weierstrass function, with the exception of $\alpha = 1$.

Table 7 represents the computational performance of the proposed solver in this paper based on IWO, our proposed method with genetic algorithm instead of IWO, Brits et al. [6] with *nbest* PSO, Hirsh et al. [8] with C-GRASP optimizer and Henderson et al. [9] with the polarization technique and the standard Simulated Annealing. Columns one and two of Table 7 show the total computational time and time spent for calculating each root respectively. For calculating this performance the termination

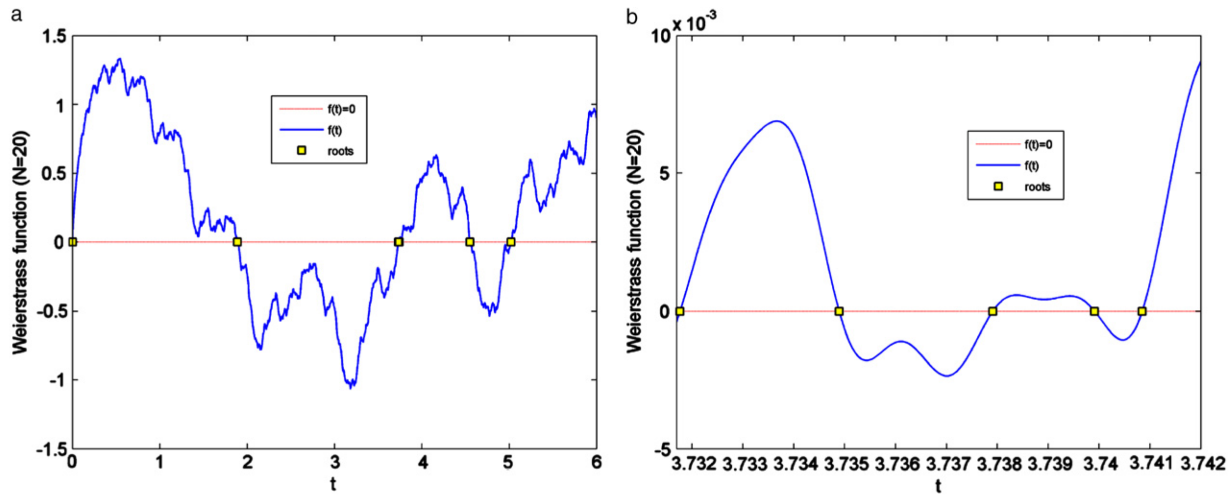

Fig. 7. The diagram and roots of the truncated Weierstrass function.

Table 6
Performance of the algorithm in solving the Weierstrass function with different values of α.

Performance[a]	α (Density factor)				
	200	250	300	350	400
Total CPU time (s)	1.3791	1.0083	0.8563	0.8367	0.7991
Average number of iterations	10.02	7.52	6.46	6.36	6.10
Average time per each solution (s)	0.1532	0.1120	0.0951	0.0930	0.0888

[a] Obtained from 50 runs of the algorithm.

Table 7
Comparison between four optimization-based solvers.

Performance[a]	Proposed solver in this paper with IWO (23 roots)	Using GA instead of IWO in the proposed method (23 roots)	Brits et al. [6] with PSO (19 roots)	Hirsh et al. [8] with GRASP (19 roots)	Henderson et al. [9] with SA (23 roots)
Total CPU time (s)	1.2267	20.2884	3.1612	6.9189	21.0839
Average time spent on finding each root (s)	0.0533	0.8821	0.1664	0.3642	0.9167
Success rate	100%	100%	92.11%	100%	94.35%
Accuracy	3.2577e−6	3.4628e−4	3.2873e−5	3.8025e−5	2.8465e−4

[a] Obtained from over 50 runs of each algorithm.

criterion for our proposed solver and method in [9] was finding 23 solutions, for [8] was detecting 19 solutions and for [6] was reaching to the maximum iterations of 200. Undoubtedly, the computational performance of our proposed method is much better than its optimization-based counterparts, as is apparent from Table 7. Our proposed root-finder has spent only $T = 0.0533$ s on detecting each root which is a very good average for optimization-based methods. Also comparison between the performance of GA and IWO shows that IWO has been an appropriate choice for our proposed root-finder.

Column 3 of Table 7 lists the success rates of the solvers defined with Eq. (12) when the termination criterion is to reach to 200 iterations for [6] and spending three iterations without finding any new root for the other root-finders:

$$Success = \frac{Solution_{total} - Solution_{detected}}{Solution_{total}} \times 100\% \tag{12}$$

where $Solution_{total}$ is the total number of solutions of the equation and $Solution_{detected}$ is the number of roots detected in the execution of a root-finder.

The last column of Table 7 shows *accuracy* of root-finders calculated as follows:

$$accuracy = \frac{1}{Root_{total}} \sum_{i=1}^{Root_{total}} |root_{exact}(i) - root_{calc}(i)| \tag{13}$$

where $Root_{total}$ is the total number of roots, $root_{exact}$ is the exact value of solutions to Eq. (11) and $Root_{calc}$ is the value of the obtained solutions by root-finders. A smaller *accuracy* value, a more precise algorithm in calculating roots and according to Table 7, our proposed method with IWO is the most precise root-finder between all methods in Table 7 with the lowest *accuracy* value 3.2577e−6.

4.4. Nonlinear equations systems and higher dimensions

Two systems of nonlinear transcendental equations are considered. The first system is given by Eq. (14):

$$f(x) = \begin{cases} x_1^2 - x_2 - 2 = 0 \\ x_1 + \sin\left(\frac{\pi}{2} x_2\right) = 0. \end{cases} \tag{14}$$

The solver finds three roots where lie in $x_0 = (0, -2), x_0 = (1, -1)$ and $x_0 = (0.7071, -1.5)$ for this system.

The second system which is non-differentiable is defined in Eq. (15):

$$f(x) = \begin{cases} 3 - x_1 x_3^2 = 0 \\ x_3 \sin\left(\frac{\pi}{x_2}\right) - x_3 - x_4 = 0 \\ -x_2 x_3 e^{(1 - x_1 x_3)} + 0.2707 = 0 \\ 2x_1^2 x_3 - x_2^4 x_3 - x_2 = 0. \end{cases} \tag{15}$$

Table 8
Complex roots of Eq. (18).

Solution	x
1	$-4.0384 - j1.3914$
2	$-4.0384 + j1.3914$
3	$-2.7649 - j1.1284$
4	$-2.7649 + j1.1284$
5	$-1.4716 - j0.8492$
6	$-1.4716 + j0.8492$
7	$-0.2296 - j0.5596$
8	$-0.2296 + j0.5596$
9	$1.0612 - j0.3980$
10	$1.0612 + j0.3980$
11	2.1250
12	2.7937
13	3.1685
14	3.9813

The proposed solver finds a unique root for this system at $x_0 = (3, 2, 1, 0)$.

The capability of the proposed solver for dealing with higher dimensions can be investigated by applying it to the well-known Schwefel function expressed below:

$$SC_n(x) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|}). \tag{16}$$

The unique solution to this function lies $x^* = (420.9687, \ldots, 420.9687)$ in the domain $[-500, 500]^n$. The proposed algorithm was applied to solve this function with a dimension of $n = 10$. It found a solution at $x_0 = (420.9688, 420.9687, 420.9689, 420.9692, 420.9686, 420.9694, 420.9695, 420.9685, 420.9688, 420.9689)$ with the fitness value $SC_{10}(x_0) = 1.2743e{-}4$ for a typical run of the solver. Proper solving of this high-dimension function shows that our proposed algorithm has the ability to solve equations with higher dimensions.

4.5. Finding complex roots

One can extend the proposed solver for finding complex root of an equation. It just needs representing each dimension of the equation with two sub-dimensions as stated in Eq. (17):

$$x_i = x_{i,Re} + jx_{i,Im} \tag{17}$$

where two sub-dimensions $x_{i,Re}$ and $x_{i,Im}$ represent the real and imaginary parts of x_i respectively, and j denotes the imaginary unit. Such modification doubles the dimension of the problem, as a result of representing each dimension with two sub-dimensions. The proposed solver is applied to Eq. (18) to find its complex roots.

$$f(x) = \sin(\pi x)\cos\left(\frac{\pi}{2}x\right)e^x + \pi \tag{18}$$

where x is a complex number represented with Eq. (18) and $x_{Re}, x_{Im} \in [-4.5, 4.5]$. The obtained roots are listed in Table 8 where 10 out of 14 solutions are complex.

5. Tuning the parameters

Most of the examples in Section 4 were solved with a similar set of parameters. This demonstrates that the sensitivity of our proposed solver to the values of its parameters is not high. Though, selecting a proper set of parameters can promote the computational performance of the algorithm.

Two issues should be paid attention. The first issue is about α: if a problem has some closely located roots, it would be better to set $\alpha > 1$. This helps the algorithm to effectively locate the solutions. The latter issue is about selecting $P_{\max}^{global}$: if the termination criterion of the algorithm is to pass a pre-determined number of iterations of the two-phase root-finder without detecting a new solution, choosing a high value for $P_{\max}^{globa}$ can increase the chance of finding all solutions to the problem, because more plants are now available to probe the search area during the global search.

6. Practical applications

Wherever it is needed to solve nonlinear equations, our proposed solver can be employed. This section presents three real world applications for the proposed solver. The first example is related to chemical engineering and finding the equilibrium point of a chemical system. In the second example the proposed algorithm solves a nonlinear equation used in robotics for describing spatial architecture of planar parallel manipulators and the last example demonstrates the ability of our presented algorithm in solving a geometry problem.

6.1. Chemical system

The third example is the model of a chemical system that presented in [19] and also used in [8,9] as a test system. The system is a model of two Continuous, Series, and non-adiabatic Tank Reactors (CSTR) at steady state with recycle and an exothermic, first order, irreversible reaction. The system is represented with two nonlinear equations:

$$\begin{cases} (1-R)\left[\dfrac{D}{10(1+\beta_1)} - \Phi_1\right]\exp\left(\dfrac{10\Phi_1}{1+\frac{10\Phi_1}{\gamma}}\right) - \Phi_1 = 0 \\ (1-R)\left[\dfrac{D}{10} - \beta_1\Phi_1 - (1+\beta_2)\Phi_2\right] \\ \quad\times\exp\left(\dfrac{10\Phi_2}{1+\frac{10\Phi_2}{\gamma}}\right) + \Phi_1 - (1+\beta_2)\Phi_2 = 0 \\ 0 \le \Phi_i \le 1, \quad i = 1, 2 \end{cases} \tag{19}$$

where Φ_1 and Φ_2 are two dimensionless variables represent temperatures in the reactors. Constant parameters β_1, β_2, γ and D are set to 2, 2, 1000 and 22 respectively. Depending on the value assigned to the recycle ratio R, the number of solutions to Eq. (19) varies between 1 and 7. Results of applying the proposed solver to the CSTR problem are provided in Table 9, as the recycle ratio R is assigned values in the set $\{0.935, 0.940, \ldots, 0.960\}$. The second column gives the number of solutions for each value of R and the last column presents the solutions found by the root-finder.

6.2. Application in robotics

Our solver can be utilized to solve nonlinear equations with more than one root encountered in robotics. Planar parallel manipulators are parallel robots that because of their rigid design, their displacements are subject to hard constraints, which are expressed as nonlinear equations. Collins [20] presented the displacement of a type of planar parallel manipulators called 3-*RPR* with a four-component vector $P = (q_1, q_2, q_3, q_4)T$ which is known as *Planar quaternion*. In this example we do not want to explain the planar manipulators (see [20] for more explanations), but our goal is finding values of q_4 so that q_4 always satisfies the constraint expressed in Eq. (20):

$$k_0 + k_2q_4^2 + k_4q_4^4 + k_6q_4^6 + \left(k_1q_4 + k_3q_4^3 + k_5q_4^5\right) * \sqrt{1-q_4^2} = 0 \tag{20}$$

Table 9
Solutions to cstr per different values of R.

R	No. of solutions	Φ
0.935	1	(0.724987, 0.245241)
0.940	1	(0.724233, 0.245133)
0.945	3	(0.079764, 0.664381), (0.172231, 0.591346), (0.723330, 0.244987)
0.950	5	(0.062799, 0.103830), (0.062799, 0.195656), (0.206003, 0.557880), (0.722226, 0.244796), (0.062799, 0.675505)
0.955	5	(0.051212, 0.078028), (0.236329, 0.520375), (0.051212, 0.235148), (0.051192, 0.682362), (0.720846, 0.244535)
0.960	7	(0.042125, 0.061755), (0.266589, 0.327275), (0.266589, 0.178423), (0.266589, 0.461132), (0.042114, 0.686938), (0.042125, 0.268726), (0.719074, 0.244162)

Table 10
The obtained roots for example 6.2.

Solution	Value of q_4
1	−0.999628022748170
2	−0.880779518883258
3	0.483255814603585
4	0.876501865784002
5	0.958895114150097
6	0.991563469631314

where k_i are functions of the structural parameters. For the given k_i from [20], Eq. (20) can be rewritten as follows:

$$3.9852 - 10.039q_4^2 + 7.2338q_4^4 - 1.17775q_4^6 + \left(-8.8575q_4 + 20.091q_4^3 - 11.177q_4^5\right) * \sqrt{1-q_4^2} = 0. \quad (21)$$

The solver finds six real roots for q_4 in the interval $[-1, 1]$ which satisfy Eq. (21) and are listed in Table 10. Of course, if q_4 is not limited to the mentioned interval, it will have some complex roots. Though, these complex roots do not have a physical interpretation.

6.3. Geometry problem

The last example in this section is to find the solution of thin wall rectangle girder section [21]:

$$\begin{cases} f_1(b,h,t) = bh + (b-2t)(h-2t) - 165 = 0 \\ f_2(b,h,t) = \dfrac{bh^3}{12} - \dfrac{(b-2t)(h-2t)^3}{12} - 9369 = 0 \\ f_3(b,h,t) = \dfrac{2(h-t)^2(b-t)^2 t}{b+h-2t} - 6835 = 0 \end{cases} \quad (22)$$

where b is the width, h is the height and t is the thickness of the section, respectively. Since it is a real world problem, the values of all variables must be positive. Applying the proposed solver to this problem, the solution $(b, h, t) = (43.1556, 10.1289, 12.9440)$ is found. If the constraint on the variables did not exist, the algorithm would detect another solution at $(b, h, t) = (-7.6030, -24.5420, -11.5767)$ for Eq. (22).

7. Conclusion

In this paper, we introduced a new optimization-based root-finder using IWO algorithm. The proposed solver consisted of two parts: a two-phase root-finder for finding the roots and an alteration technique for modifying the objective function. IWO played the main role in the two-phase root-finder. Our proposed algorithm had interesting capabilities including detecting multiplicity, finding closely located roots, finding complex roots and good computational performance. We compared this method to other optimization-based solvers. Results were encouraging. Besides applying to several illustrative examples, we applied our proposed solver to three real world applications. In all cases, the solver was able to find all solutions to the system accurately and quickly.

References

[1] M. Goyel, Computer-Based Numerical & Statistical Techniques, Infinity Science Press LLC, Hingham, Massachusetts, 2007.
[2] J. Wu, Z. Cui, J. Liu, Using hybrid social emotional optimization algorithm with metropolis rule to solve nonlinear equations, in: 10th IEEE International Conference on Cognitive Informatics & Cognitive Computing, ICCI*CC, 18–20 August 2011, pp. 405–411.
[3] M. Jaberipour, E. Khorram, B. Karimi, Particle swarm algorithm for solving systems of nonlinear equations, Computers and Mathematics with Applications 62 (2) (2011) 566–576.
[4] Z. Wu, L. Kang, A fast and elitist parallel evolutionary algorithm for solving systems of non-linear equations, in: Proceedings of Congress on Evolutionary Computation, vol. 2, 2003, pp. 1026–1028.
[5] J. Dai, G. Wu, Y. Wu, G. Zhu, Helicopter trim research based on hybrid genetic algorithm, in: Proceedings of World Congress on Intelligent Control and Automation, 2008, pp. 2007–2011.
[6] R. Brits, A.P. Engelbrecht, F. van den Bergh, Solving systems of unconstrained equations using particle swarm optimization, in: Proceedings of International Conference on Systems, Man and Cybernetics, vol. 3, 2002, pp. 6–9.
[7] C. Grosan, A. Abraham, A new approach for solving nonlinear equations systems, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 38 (3) (2008) 698–714.
[8] M.J. Hirsch, P.M. Pardalos, M.G.C. Resende, Solving systems of nonlinear equations with continuous GRASP, Nonlinear Analysis Real World Applications 10 (4) (2009) 2000–2006.
[9] N. Henderson, W.F. Sacco, G.M. Platt, Finding more than one root of nonlinear equations via a polarization technique: an application to double retrograde vaporization, Chemical Engineering Research and Design 88 (2010) 551–561.
[10] A.R. Mehrabian, C. Lucas, A novel numerical optimization algorithm inspired from weed colonization, Ecological Informatics 1 (4) (2006) 355–366.
[11] S.H. Sedighy, A.R. Mallahzadeh, M. Soleimani, J. Rashed-Mohassel, Optimization of printed yagi antenna using invasive weed optimization (IWO), IEEE Antennas and Wireless Propagation Letters 9 (2010) 1275–1278.
[12] G.G. Roy, S. Das, P. Chakraborty, P.N. Suganthan, Design of non-uniform circular antenna arrays using a modified invasive weed optimization algorithm, IEEE Transactions on Antennas and Propagation 59 (1) (2011) 110–118.
[13] S. Karimkashi, A.A. Kishk, D. Kajfez, Antenna array optimization using dipole models for MIMO applications, IEEE Transactions on Antennas and Propagation 59 (8) (2011) 3112–3116.
[14] A. Ghosh, S. Das, A. Chowdhury, R. Giri, An ecologically inspired direct search method for solving optimal control problems with Bézier parameterization, Engineering Applications of Artificial Intelligence 24 (7) (2011) 1195–1203.
[15] R. Giri, A. Chowdhury, A. Ghosh, S. Das, A. Abraham, V. Snasel, A modified invasive weed optimization algorithm for training of feed-forward neural networks, in: IEEE International Conference on Systems Man and Cybernetics, SMC, 10–13 October 2010, pp. 3166–3173.
[16] A.H. Nikoofard, H. Hajimirsadeghi, A. Rahimi-Kian, C. Lucas, Multiobjective invasive weed optimization: application to analysis of Pareto improvement models of electricity markets, Applied Soft Computing 12 (1) (2012) 100–112.
[17] Z. Chen, S. Wang, Z. Deng, X. Zhang, Tuning of auto-disturbance rejection controller based on the invasive weed optimization, in: 6th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA, 27–29 September 2011, pp. 314–318.
[18] K. Falconer, Fractal Geometry: Mathematical Foundations and Applications, second ed., John Wiley and Sons Ltd., West Sussex, England, 2003.
[19] C.A. Floudas, Recent advances in global optimization for process synthesis, design and control: enclosure of all solutions, Computers and Chemical Engineering Supplement 23 (1999) 963–973.
[20] C.L. Collins, Forward kinematics of planar parallel manipulators in the Clifford algebra of P2, Mechanism and Machine Theory 37 (8) (2002) 799–813.
[21] Y.Z. Luo, G.J. Tang, L.N. Zhou, Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method, Applied Soft Computing 8 (2) (2008) 1068–1073.