Research Article

Mathematical Methods in the Applied Sciences

Received 2 January 2015

Published online 29 September 2015 in Wiley Online Library

(wileyonlinelibrary.com) DOI: 10.1002/mma.3715
MOS subject classification: 65H10

An adaptive learning rate backpropagation-type neural network for solving $n \times n$ systems on nonlinear algebraic equations

K. Goulianas[a], A. Margaris[b*†], I. Refanidis[b] and K. Diamantaras[a]

Communicated by T. Monovasilis

This paper presents an MLP-type neural network with some fixed connections and a backpropagation-type training algorithm that identifies the full set of solutions of a complete system of nonlinear algebraic equations with *n* equations and *n* unknowns. The proposed structure is based on a backpropagation-type algorithm with bias units in output neurons layer. Its novelty and innovation with respect to similar structures is the use of the hyperbolic tangent output function associated with an interesting feature, the use of adaptive learning rate for the neurons of the second hidden layer, a feature that adds a high degree of flexibility and parameter tuning during the network training stage. The paper presents the theoretical aspects for this approach as well as a set of experimental results that justify the necessity of such an architecture and evaluate its performance. Copyright © 2015 John Wiley & Sons, Ltd.

Keywords: nonlinear algebraic systems; backpropagation; numerical analysis

1. Introduction

The identification of the full set of roots of systems of nonlinear algebraic equations has been paid with serious attention in the last years in many disciplines and fields of human knowledge such as physics and chemistry, mechanics and engineering, and generally, applied mathematics. The method described and applied in this paper is another attempt towards this goal, based on the use of neural networks. The following sections describe the problem formulation, namely, the structure of the neural network and its mapping to the corresponding structure of the $n \times n$ system of nonlinear algebraic equation, a short review of previous related work that is based on the neural approach as well as other approaches (e.g., simulated annealing and genetic algorithms), the experimental results emerged for solving example systems found in the literature, and most importantly, the comparison of the proposed method against other methods in terms of the number of the identified roots and the simulation accuracy.

2. Problem formulation

The system of nonlinear algebraic equations to be solved is composed of n equations, each one with n unknowns, namely,

$$\begin{aligned} f_1(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ f_n(x_1, x_2, x_3, \ldots, x_n) &= 0 \end{aligned} \tag{1}$$

or in vector form $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$. In this notation, $\boldsymbol{F} = (f_1, f_2, f_3, \ldots, f_n)^T$ is a vector of the nonlinear functions $f_i(\boldsymbol{x}) = f_i(x_1, x_2, x_3, \ldots, x_n)$, each one of them being defined in the vector space of all real valued continuous functions, and $\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_n)^T$ is the vector of

[a] Department of Computer Science and Engineering, TEI of Thessaloniki, Thessaloniki, Greece
[b] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
* Correspondence to: A. Margaris, Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece.
† E-mail: amarg@uom.gr

unknowns. A special case of the aforementioned general system is associated with the use of constant coefficients with each function $f_i(\boldsymbol{x}) = f_i(x_1, x_2, x_3, \ldots, x_n)$ being expressed as [1]

$$f_i(\boldsymbol{x}) = \sum_{j_1, j_2, \ldots, j_{s_i}}^{n} A_i^{j_1 j_2 \ldots j_{s_i}} x_{j_1} x_{j_2} \ldots x_{j_{s_i}} = 0 \tag{2}$$

where $j_1, j_2, \ldots, j_{s_i}$ $(i = 1, 2, \ldots, n)$ are the indices of the x variables. In complete accordance with the well-known linear algebraic systems, this system has one non-vanishing solution (that is, at least one $x_j \neq 0$) if and only if the resultant $\Re$ of the system is equal to zero, or in mathematical form

$$\Re_{s_1, s_2, \ldots s_n} \left\{ A_i^{j_1 j_2 \ldots j_{s_i}} \right\} = 0 \tag{3}$$

where $(s_1, s_2, \ldots, s_n)$ are the degrees of equations.

The classical and most known method for solving the aforementioned system of nonlinear algebraic equations is the Newton's method [2] that approximates the function $\boldsymbol{F}(\boldsymbol{x})$ by its first-order Taylor expansion in the neighborhood of a specific point $\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_n)^T \in R^n$. This iterative method starts from an initial guess $\boldsymbol{x}_0$ and generates a sequence of consecutive points towards to the solution as $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} - \boldsymbol{J}(\boldsymbol{x}_{k-1})^{-1} \boldsymbol{F}(\boldsymbol{x}_{k-1})$. Even though this method is characterized by fast convergence (provided that the initial guess is a good one), it requires in each step the evaluation of a Jacobian matrix (namely, the estimation of n^2 partial derivatives) as well as the solution of an $n \times n$ linear system and, for this reason, is impractical for large-scale problems. An improvement to this approach can be found in the Broyden's method [3] that also suffers from the good initial guess, the secant method [4], as well as the steepest descent method, where this guess is not an issue but a rapidly convergence sequence of vectors can not be achieved.

3. Review of previous work

The approaches that have been developed so far in an attempt to overcome the limitations of the classical algorithms and the related methods described earlier can be grouped into two classes: the interval methods that are robust but suffer from large execution time and the continuation methods that are well suited for problems where the total degree is not too high [5]. An interesting family of such methods includes the ABS methods [6] that use a generalization of the projection matrix concept known as Abaffian [7, 8] and their variants (for example, [9], where these methods are used in conjunction with the quasi-Newton method). A completely different approach is the solution of systems of nonlinear algebraic equations via genetic algorithms (for example [10, 11], and [12]), where a population of candidate solutions to an optimization problem is evolved towards better solutions until a maximum number of generations has been produced or a satisfactory fitness level has been reached. Another class of solution methods is based on invasive weed optimization [13], allowing the identification of all real and complex roots as well as the detection of multiplicity. There are four stages in these methods, namely, an initialization stage, where a finite number of seeds are dispread randomly over the n-dimensional search area, a reproduction stage, where each plant is allowed to reproduce seeds based on their fitness, a spatial dispersal stage, where the produced seeds are randomly distributed in the search space, and a competitive exclusion stage, where undesirable plants with poor fitness are eliminated, whereas fitter plants are allowed to reproduce more seeds. In these methods, the root-finding process is composed of two phases, namely, a global search where plants abandon non-minimum points vacant and settle down around minima, and an exact search where the exact locations of roots are determined via a clustering procedure that clusters plants around each roots.

On the other hand, Oliveira and Petraglia [14] solve systems of nonlinear algebraic equations using stochastic global optimization, while Effati and Nazemi [15] apply to this problem the so-called measure theory, a tool capable of dealing with optimal control problems. The first approach includes the reformulation of the original problem as a global optimization one and the application of a fuzzy adaptive simulated annealing stochastic process, while in the second approach, the problem is transformed to an optimal control problem associated with the minimization of a linear functional over a set of Radon measures, and the system is solved using finite dimensional nonlinear programming techniques. Grosan and Abraham [16] deal the system of nonlinear equations as a multi-objective optimization problem solved via an evolutionary computational technique, while Liu *et al.* [17] used the population migration algorithm [18] in conjunction with the well-known quasi-Newton method [19].

The last family of methods reported here is based on neural network techniques and include among others the use of recurrent neural networks [20–32] for the neural-based implementation of the Newton's method, the approximation of the inverse system function $\boldsymbol{F}^{-1}(\boldsymbol{x})$ via backpropagation networks [33], and the neural computation method of Meng and Zeng [34] that minimizes the energy function $J = \left(\sum_{i=1}^{n} e^2(i)\right)/2$ using an iterative algorithm.

4. The structure of the proposed neural nonlinear system solver

The proposed neural network architecture for solving systems of nonlinear algebraic equations is the generalization of an idea that appears for the first time in [35] and describes a four-layered feed forward backpropagation neural network that uses a fixed learning rate. The number of the neurons in the output layer was equal to the number of equations of the system of nonlinear algebraic equations, and the network had been designed in such a way that the total input to the i_{th} output neuron to coincide with the left-hand side of the i_{th} system equation. The components of the system roots to be identified were associated with the variable weights of synapses between the single input neuron and the n neurons of the second layer; the fixed coefficients of the terms for each equation

are assigned to the fixed weights of synapses that join with full connections the neurons of the third and the fourth layer (which is the output layer), whereas the fixed term of each equation has been embedded to the corresponding output neuron as a bias unit. Regarding the synapses between the second and the third layer, they have also fixed weights with values 0 or 1 according to the linear or nonlinear term associated with their target neuron. In this first attempt for using such a system, the activation function of all neurons was the identity function; however, the network always converged to the same root, a problem that was solved using the nonlinear hyperbolic tangent function. The network was trained with a backpropagation-type algorithm and with the zero vector as the desired output vector. In this way, the network was capable of estimating all the roots of the system, one root per training, according to the set of initial conditions. This network has been tested successfully in solving 2×2 [36] as well as 3×3 nonlinear systems [37].

Continuing this line of research, the present article generalizes the solvers presented in [36] and [37] for solving 2×2 [36] as well as 3×3 nonlinear systems [37] for the general case of $n \times n$ nonlinear systems. This generalization is not restricted only to the dimension of the neural solver, but it is extended to the form of the system because it has been enhanced with some new features that do not appear in [36] and [37]. The most important feature is that the activation functions of the third layer neurons can be any continuous and differentiable function, a feature that adds another degree of nonlinearity and allows the network to work correctly and with success, even in cases where the activation function of the output neurons is a simple linear function such as the identity functions. Furthermore, in the present work, the identity output function is also substituted by the nonlinear hyperbolic tangent function with a fast adaptive learning rate.

The complete system of nonlinear algebraic equations of n equations with n unknowns is defined as

$$\begin{aligned}
\boldsymbol{F}_1(\boldsymbol{x}) &= \boldsymbol{F}_1(x_1, x_2, \ldots, x_n) = \alpha_{11} f_{11}(\boldsymbol{x}) + \alpha_{12} f_{12}(\boldsymbol{x}) + \cdots + \alpha_{1,k_1} f_{1,k_1}(\boldsymbol{x}) - \beta_1 = 0 \\
\boldsymbol{F}_2(\boldsymbol{x}) &= \boldsymbol{F}_2(x_1, x_2, \ldots, x_n) = \alpha_{21} f_{21}(\boldsymbol{x}) + \alpha_{22} f_{22}(\boldsymbol{x}) + \cdots + \alpha_{2,k_2} f_{2,k_2}(\boldsymbol{x}) - \beta_2 = 0 \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\boldsymbol{F}_n(\boldsymbol{x}) &= \boldsymbol{F}_n(x_1, x_2, \ldots, x_n) = \alpha_{n1} f_{n1}(\boldsymbol{x}) + \alpha_{n2} f_{n2}(\boldsymbol{x}) + \cdots + \alpha_{n,k_n} f_{n,k_n}(\boldsymbol{x}) - \beta_n = 0
\end{aligned} \tag{4}$$

and its solution is the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in R^n$. In the aforementioned description, the nonlinear function $F_i(x)$ associated with the i_{th} nonlinear equation ($i = 1, 2, \ldots, n$) is treated as a vector function in the form

$$\boldsymbol{F}_i(\boldsymbol{x}) = [f_{i1}(\boldsymbol{x}), f_{i2}(\boldsymbol{x}), f_{i3}(\boldsymbol{x}), \ldots, f_{ik_i}(\boldsymbol{x})] \quad (i = 1, 2, 3, \ldots, n) \tag{5}$$

with the parameter k_i describing the number of the function components f_{ij} ($j = 1, 2, 3, \ldots, k_i$) associated with the vector function $\boldsymbol{F}_i(x)$ ($i = 1, 2, \ldots, n$). The problem is defined in the field R of real numbers; however, its generalization to the field C of complex numbers is straightforward.

The structure of the neural network that can solve a complete $n \times n$ system of nonlinear algebraic equations, is shown in Figure 1. It is characterized by four layers with the following structure:

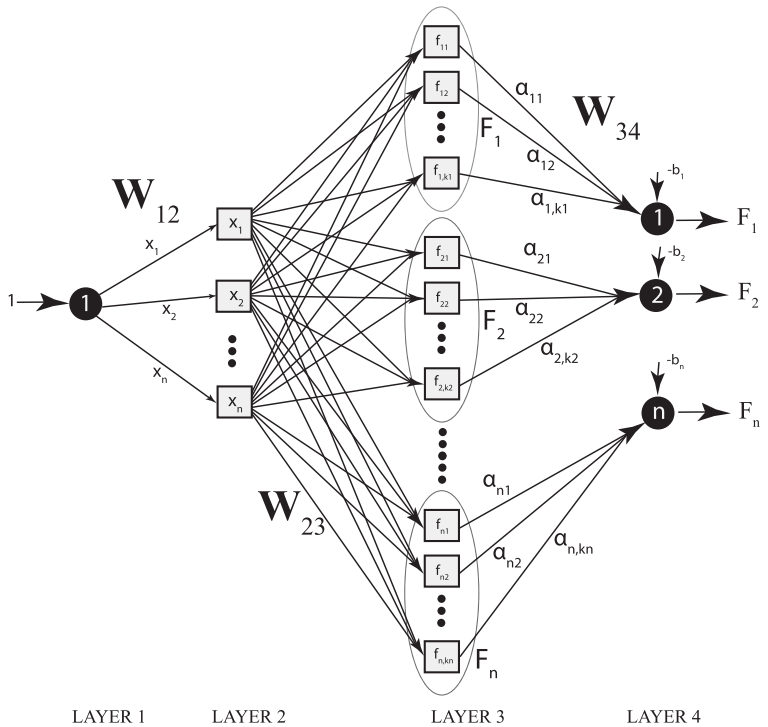

Figure 1. The structure of the neural-based solver for systems of nonlinear algebraic equations.

- Layer 1 is the single input layer. This layer does not participate to the backpropagation training, and it simply participates to the variable weight synapses whose values after training are the components of the system roots. In other words, in this procedure, there is not a training set because the simple input is always the value of unity, while the associated desired output is the zero vector of n elements.
- Layer 2 contains n neurons each one of them is connected to the single input unit of the first layer. As it has been mentioned, the weights of the n synapses defined in this way are the only variable weights of the network. During network training, their values are updated according to the equations of the backpropagation algorithm, and after a successful training, these weights contain the n components of a system root $(x_1, x_2, x_3, \ldots, x_n)$. Because the second layer makes multiple copies of the inputs, its activation function will be the identity function.
- Layer 3 is composed of n blocks of neurons with the ℓ_{th} block containing k_ℓ neurons, namely, one neuron for each one of the k_ℓ functions associated with the ℓ_{th} equation of the nonlinear system. The neurons of this layer, as well as the activation functions associated with them, are therefore described using the double index notation (ℓ, j) [for values $(\ell = 1, 2, \ldots, n)$ and $(j = 1, 2, \ldots, k_\ell)$]. The auxiliary functions f_{pq} would have to be prepared in the previous layer, where they would become the activation functions.
- Layer 4 contains an output neuron for each equation, namely, a total number of n neurons that use the hyperbolic tangent function $y = f(x) = \tanh(x)$ as the activation function. The output neurons will have to realize a weighted summation of auxiliary functions, and the output of the neuron should be the constant 0 on the other side.

Regarding the matrices of the synaptic weights, they are defined in a similar fashion as in [36] and [37] and more specifically.

The matrix $\boldsymbol{W}_{12} = \left[\boldsymbol{W}_{12}^{(1)}, \boldsymbol{W}_{12}^{(2)}, \ldots, \boldsymbol{W}_{12}^{(n)}\right] = [x_1, x_2, \ldots, x_n]$ is the only variable weight matrix, whose elements (after the successful training of the network) are the components of one of the system roots, or in a mathematical notation $\boldsymbol{W}_{12}^{(i)} = x_i$ $(i = 1, 2, \ldots, n)$.

The matrix $\boldsymbol{W}_{23}$ is composed of n rows with the i_{th} row to be associated with the variable x_i $(i = 1, 2, \ldots, n)$. The values of this row are the weights of the synapses joining the i_{th} neuron of the second layer with the complete set of neurons of the third layer. There is a total number of $k = k_1 + k_2 + \cdots + k_n$ neurons in this layer, and therefore, the dimensions of the matrix $\boldsymbol{W}_{23}$ are $n \times k = n \times (k_1 + k_2 + \cdots + k_n)$. Regarding the values of these weights, they have a value of unity if the function $f_{\ell j}$ is a function of x_i; otherwise, they have a zero value. Therefore, we have

$$\boldsymbol{W}_{23}^{(ij\ell)} = \begin{cases} 1, & \text{if } f_{\ell j} = f_{\ell j}(x_i) \\ 0, & \text{otherwise} \end{cases} \qquad \begin{matrix} i = 1, 2, \ldots, n \\ \ell = 1, 2, \ldots, n \\ j = 1, 2, \ldots, k_\ell \end{matrix} \tag{6}$$

Because the second and third layers are fully connected, then, it is simple to conclude that if some x_i is an argument of a function f_{ij}, then it should be weighted by 1 and if f_{ij} is independent of x_i with a weight equal to 0.

Finally, the matrix $\boldsymbol{W}_{34}$ has dimensions $k \times n = (k_1 + k_2 + \cdots + k_n) \times n$ with elements

$$\boldsymbol{W}_{34}^{(\ell j, \ell)} = \alpha_{\ell j} \qquad \begin{matrix} \ell = 1, 2, \ldots, n \\ j = 1, 2, \ldots, k_\ell \end{matrix} \tag{7}$$

The weighted summation indicates that the activation function of the output neurons should be (or behave as) linear, and the weights of the incoming edges should correspond the α_{ij} coefficients of Equation (4). The figure shows only the synapses with a nonzero weight value.

Because the unique neuron of the first layer does not participate in the calculations, it is not included in the index notation. Therefore, wherever we use the symbol u to describe the neuron input and the symbol v to describe the neuron output, the symbols u_1 and v_1 are associated with the n neurons of the second layer, the symbols u_2 and v_2 are associated with the k neurons of the third layer and the symbols u_3 and v_3 are associated with the n neurons of the third (output) layer. These symbols are accompanied by additional indices that identify a specific neuron inside a layer, and this notation is used throughout the remaining part of the article.

5. Building the backpropagation equations

5.1. *Forward pass*

The inputs and the outputs of the network neurons during the forward pass stage are computed as follows:

LAYER 2. $u_1^\ell = \boldsymbol{W}_{12}^\ell = x_\ell$ and $v_1^\ell = u_1^\ell = x_\ell$ $(\ell = 1, 2, \ldots, n)$.

LAYER 3. $u_2^{(\ell j_\ell)} = x_\ell$ and $v_2^{(\ell j_\ell)} = f_{\ell, j_\ell}(\boldsymbol{x})$.

LAYER 4. The input and the output of the fourth layer neurons are computed as

$$u_3^\ell = \sum_{j=1}^{k_\ell} v_2^{(\ell j)} \boldsymbol{W}_{34}^{(\ell j_\ell, \ell)} - \beta_\ell = \sum_{j=1}^{k_\ell} v_2^{(\ell j)} \alpha_{ell,j} - \beta_\ell = \boldsymbol{F}_\ell(\boldsymbol{x}) = \boldsymbol{F}_\ell(x_1, x_2, \ldots, x_n) \tag{8}$$

and $v_3^\ell = \tanh\left(u_3^\ell\right) = \tanh[\boldsymbol{F}_\ell(\boldsymbol{x})]$ $(\ell = 1, 2, \ldots, n)$, where the activation function of the output neurons is the hyperbolic tangent function $f(\xi) = \tanh(\xi)$.

5.2. *Backward pass – estimation of the adaptive errors*

Working in the same way and keeping in mind that $v_3^\ell = \tanh[\boldsymbol{F}_\ell(\boldsymbol{x})]$ $(\ell = 1, 2, \ldots, n)$ and $\tanh'(x) = -\tanh(x)[1 - \tanh^2 x]$, we get the following results.

$$\delta_3^\ell = -v_3^\ell\left[1 - \left(v_3^\ell\right)^2\right] = -\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})]\left(1 - \tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})]\right)$$

$$\delta_{2(x_k)}^{\ell j_\ell} = \delta_3^\ell \frac{\partial v_2^{(\ell j_\ell)}(\boldsymbol{x})}{\partial x_k} = \delta_3^\ell \frac{\partial f_{\ell j_\ell}(\boldsymbol{x})}{\partial x_k} = -\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})](1 - \tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})])\frac{\partial f_{\ell j_\ell}}{\partial x_k}$$

$(k, \ell = 1, 2, \ldots, n$ and $j_\ell = 1, 2, \ldots, k_\ell)$. Finally, the parameter δ_1^j $(j = 1, 2, \ldots, n)$ is estimated as

$$\begin{aligned}\delta_1^j &= \sum_{\ell=1}^{n}\sum_{j_\ell=1}^{k_\ell}\delta_{2(x_k)}^{\ell j_\ell} = -\sum_{\ell=1}^{n}\sum_{j_\ell=1}^{k_\ell}\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})](1 - \tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})])\frac{\partial f_{\ell j_\ell}}{\partial x_k}\\ &= \sum_{\ell=1}^{n}\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})](1 - \tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})])\sum_{j_\ell=1}^{k_\ell}\frac{\partial f_{\ell j_\ell}}{\partial x_k}\\ &= -\sum_{\ell=1}^{n}\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})](1 - \tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})])\frac{\partial \boldsymbol{F}_\ell(\boldsymbol{x})}{\partial x_j} \qquad (j = 1, 2, \ldots, n)\end{aligned}$$

5.3. *Weight adaptation and proof of convergence*

The weight update equation for the case of the function $f(x) = \tanh(x)$ is performed again via the equation

$$x_k^{m+1} = x_k^m + \beta\delta_1^k \tag{9}$$

If we take into account that the function $E(\boldsymbol{x})$ has the form

$$E(\boldsymbol{x}) = \frac{1}{2}\sum_{\ell=1}^{n}\left(d_\ell - v_3^\ell\right)^2 = \frac{1}{2}\sum_{\ell=1}^{n}\left(0 - v_3^\ell\right)^2 = \frac{1}{2}\sum_{\ell=1}^{n}\left(v_3^\ell\right)^2 = \frac{1}{2}\sum_{\ell=1}^{n}\tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})]$$

we can easily see that

$$\frac{\partial E(\boldsymbol{x})}{\partial x_k} = \sum_{\ell=1}^{n}\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})]\frac{\partial\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})]}{\partial x_k} = \sum_{\ell=1}^{n}\tanh[\boldsymbol{F}_\ell(\boldsymbol{x})](1 - \tanh^2[\boldsymbol{F}_\ell(\boldsymbol{x})])\frac{\partial \boldsymbol{F}_\ell(\boldsymbol{x})}{\partial x_k} = -\delta_1^k$$

and therefore, Equation (9) becomes

$$x_k^{m+1} = x_k^m + \beta\delta_1^k = x_k^m - \beta\left(-\delta_1^k\right) = x_k^m - \beta\frac{\partial E(\boldsymbol{x})}{\partial x_k} \quad (k = 1, 2, \ldots, n)$$

As it can be seen, the calculation of δ_1 parameter implements the gradient descent backpropagation algorithm. Regarding the convergence analysis, it is discussed in Section 6.

5.4. *The case of adaptive learning rate*

In this case, each neuron in the first layer has its own learning rate value $\beta(j)$ $(j = 1, 2, \ldots, n)$.

The energy function associated with the m_{th} iteration is defined as

$$E^m(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{n}\left(0 - v_3^{i,m}\right)^2 = \frac{1}{2}\sum_{i=1}^{n}\left[v_3^{i,m}\right]^2 = \frac{1}{2}\sum_{i=1}^{n}\tanh^2\left[F_i^m(\boldsymbol{x})\right] \tag{10}$$

If the energy function for the $(m+1)_{\text{th}}$ iteration is denoted as $E^{m+1}(\boldsymbol{x})$, then the energy difference is

$$\begin{aligned}\Delta E^m(\boldsymbol{x}) &= E^{m+1}(\boldsymbol{x}) - E^m(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{n}\tanh^2\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right] - \frac{1}{2}\sum_{i=1}^{n}\tanh^2\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\\ &= \frac{1}{2}\sum_{i=1}^{n}\left\{\tanh^2\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right] - \tanh^2\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\right\}\\ &= \frac{1}{2}\sum_{i=1}^{n}\left\{\left[\tanh\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right] - \tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\right]\left[\tanh\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right] + \tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\right]\right\}\\ &= \frac{1}{2}\sum_{i=1}^{n}\left\{\left[\tanh\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right] - \tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\right]\left[\tanh\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right] - \tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right] + 2\tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\right]\right\}\\ &= \frac{1}{2}\sum_{i=1}^{n}\left\{\Delta\tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\left[\Delta\tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right] + 2\tanh\left[\boldsymbol{F}_i^{m}(\boldsymbol{x})\right]\right]\right\}\end{aligned}$$

From the weight update equation of the backpropagation algorithm,

$$\Delta x_j = -\beta(j)\frac{\partial E^m(\boldsymbol{x})}{\partial x_j} = -\beta(j)\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\left(1-\tanh^2\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\right)\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_j} \quad (j=1,2,\ldots,n) \tag{11}$$

Therefore, we get

$$\begin{aligned}\Delta\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right] &= \frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\Delta x_j\\ &= -\beta(j)\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\left(1-\tanh^2\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\right)\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_j}\\ &= -\beta(j)\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}\end{aligned}$$

because according to the chain rule of the derivation,

$$\left(1-\tanh^2\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\right)\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_j} = \frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_j} = \frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}$$

Finally, using this expression in the equation of $\Delta E^m(\boldsymbol{x})$, it gets the form

$$\begin{aligned}\Delta E^m(\boldsymbol{x}) &= \frac{1}{2}\sum_{i=1}^{n}\left\{\left(-\beta(j)\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}\right)\times\right.\\ &\quad\left.\times\left(-\beta(j)\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j} + 2\tanh\boldsymbol{F}_i^m(\boldsymbol{x})\right)\right\}\\ &= \frac{1}{2}\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}\times\\ &\quad\times\left[\sum_{i=1}^{n}\beta(j)^2\left(\frac{\partial\tanh[\boldsymbol{F}_i^m(\boldsymbol{x})]}{\partial x_j}\right)^2\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j} -\right.\\ &\quad\left.-2\beta(j)\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\right]\\ &= \frac{1}{2}\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}\times\\ &\quad\times\left[\sum_{i=1}^{n}\beta(j)^2\left(\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\right)^2\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j} -\right.\\ &\quad\left.-2\beta(j)\sum_{i=1}^{n}\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\right]\\ &= \frac{1}{2}\beta(j)\left(\sum_{\ell=1}^{n}\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]\frac{\partial\tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}\right)^2\left[\beta(j)\sum_{i=1}^{n}\left(\frac{\partial\tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\right)^2 - 2\right]\end{aligned}$$

The condition of convergence for the backpropagation algorithm is $\Delta E^m(\boldsymbol{x}) < 0$. Because by definition the learning rate is a positive number, namely, $\beta(j) > 0$ $(j = 1, 2, \ldots, n)$ and furthermore

$$\left(\sum_{\ell=1}^{n} \tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right] \frac{\partial \tanh\left[\boldsymbol{F}_\ell^m(\boldsymbol{x})\right]}{\partial x_j}\right)^2 > 0$$

it should be

$$\beta(j) \sum_{i=1}^{n} \left(\frac{\partial \tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\right)^2 < 2$$

or

$$\beta(j) < \frac{2}{\sum_{i=1}^{n} \left(\frac{\partial \tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial x_j}\right)^2} = \frac{2}{\sum_{i=1}^{n} \left(\frac{\partial \tanh\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]}{\partial \boldsymbol{F}_i^m(\boldsymbol{x})} \times \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_j}\right)^2}$$

$$= \frac{2}{\sum_{i=1}^{n} \left[\left(1 - \tanh^2\left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]\right) \times \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_j}\right]^2} = \frac{2}{\sum_{i=1}^{n} \left[\left(1 - \left[v_3^\ell\right]^2\right) \times \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_j}\right]^2}$$

5.5. *A modified adaptive learning rate (MALR)*

Because $v_3 = \tanh(u_3)$ and $\tanh(0) = 0$, the energy function to be minimized can be defined as

$$E^m(\boldsymbol{x}) = \frac{1}{2} \sum_{i=1}^{n} \left(0 - v_3^{i,m}\right)^2 = \frac{1}{2} \sum_{i=1}^{n} \left(v_3^{i,m}\right)^2 = \frac{1}{2} \sum_{i=1}^{n} \left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]^2 \tag{12}$$

and the associated energy difference is

$$\begin{aligned}
\Delta E^m(\boldsymbol{x}) &= E^{m+1}(\boldsymbol{x}) - E^m(\boldsymbol{x}) = \frac{1}{2} \sum_{i=1}^{n} \left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right]^2 - \frac{1}{2} \sum_{i=1}^{n} \left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} \left\{\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x})\right]^2 - \left[\boldsymbol{F}_i^m(\boldsymbol{x})\right]^2\right\} \\
&= \frac{1}{2} \sum_{i=1}^{n} \left\{\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x}) - \boldsymbol{F}_i^m(\boldsymbol{x})\right]\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x}) + \boldsymbol{F}_i^m(\boldsymbol{x})\right]\right\} \\
&= \frac{1}{2} \sum_{i=1}^{n} \left\{\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x}) - \boldsymbol{F}_i^m(\boldsymbol{x})\right]\left[\boldsymbol{F}_i^{m+1}(\boldsymbol{x}) - \boldsymbol{F}_i^m(\boldsymbol{x}) + 2\boldsymbol{F}_i^m(\boldsymbol{x})\right]\right\} \\
&= \frac{1}{2} \sum_{i=1}^{n} \left\{\Delta \boldsymbol{F}_i^m(\boldsymbol{x})\left[\Delta \boldsymbol{F}_i^m(\boldsymbol{x}) + 2\boldsymbol{F}_i^m(\boldsymbol{x})\right]\right\}
\end{aligned} \tag{13}$$

From the weight update equation of the backpropagation algorithm, we have

$$\Delta x_k = -\beta(k) \frac{\partial E^m(\boldsymbol{x})}{\partial x_k} = -\beta(k) \sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x}) \frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k} \tag{14}$$

and therefore,

$$\Delta \boldsymbol{F}_i^m(\boldsymbol{x}) = \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k} \Delta x_k = -\beta(k) \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k} \sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x}) \frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k} \quad (k = 1, 2, \ldots, n) \tag{15}$$

Using this expression in the equation of $\Delta E^m(\boldsymbol{x})$, it gets the form

$$\begin{aligned}
\Delta E^m(\boldsymbol{x}) &= \frac{1}{2} \sum_{i=1}^{n} \left\{\left(-\beta(k) \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k} \sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x}) \frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k}\right) \times \right. \\
&\quad \left. \times \left(-\beta(k) \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k} \sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x}) \frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k} + 2\boldsymbol{F}_i^m(\boldsymbol{x})\right)\right\} \\
&= \frac{1}{2} \sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x}) \frac{\partial F_\ell^m(\boldsymbol{x})}{\partial x_k} \times \\
&\quad \times \sum_{i=1}^{n} \left\{\beta(k)^2 \left(\frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right)^2 \sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x}) \frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k} - 2\beta(k) \boldsymbol{F}_i^m(\boldsymbol{x}) \frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right\}
\end{aligned}$$

or equivalently,

$$\begin{aligned}\Delta E^m(\boldsymbol{x}) &= \frac{1}{2}\sum_{\ell=1}^{n} \boldsymbol{F}_\ell^m(\boldsymbol{x})\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k}\times \\ &\times\left[\sum_{i=1}^{n}\beta(k)^2\left(\frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right)^2\sum_{\ell=1}^{n}\boldsymbol{F}_\ell^m(\boldsymbol{x})\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k} - 2\beta(k)\sum_{i=1}^{n}\boldsymbol{F}_i^m(\boldsymbol{x})\frac{\partial \boldsymbol{F}_i(\boldsymbol{x})}{\partial x_k}\right] \\ &= \frac{1}{2}\left(\sum_{\ell=1}^{n}\boldsymbol{F}_\ell^m(\boldsymbol{x})\frac{\partial \boldsymbol{F}_\ell^m}{\partial x_k}\right)^2\times\left[\sum_{i=1}^{n}\beta(k)^2\left(\frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right)^2 - 2\beta(k)\right] \\ &= \frac{1}{2}\beta(k)\left(\sum_{\ell=1}^{n}\boldsymbol{F}_\ell^m(\boldsymbol{x})\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k}\right)^2\times\left[\beta(k)\sum_{i=1}^{n}\left(\frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right)^2 - 2\right]\end{aligned}$$

The convergence condition of the backpropagation algorithm is expressed as $\Delta E^m(\boldsymbol{x}) < 0$. Because $\beta(k) > 0$ (the learning value is obviously a positive number) and of course it holds that

$$\left(\sum_{\ell=1}^{n}\boldsymbol{F}_\ell^m(\boldsymbol{x})\frac{\partial \boldsymbol{F}_\ell^m(\boldsymbol{x})}{\partial x_k}\right)^2 > 0 \tag{16}$$

it has to be

$$\beta(k)\sum_{i=1}^{n}\left(\frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right)^2 - 2 < 0 \tag{17}$$

or equivalently

$$\beta(k) < \frac{2}{\sum_{i=1}^{n}\left(\frac{\partial \boldsymbol{F}_i^m(\boldsymbol{x})}{\partial x_k}\right)^2} \tag{18}$$

Defining the adaptive learning rate parameter (ALRP) μ, the aforementioned equation can be expressed as

$$\beta(k) = \frac{\mu}{\|\boldsymbol{C}_k^m(\boldsymbol{J})\|^2} \tag{19}$$

where $\boldsymbol{C}_k^m(\boldsymbol{J})$ is the k_{th} column of the Jacobian matrix

$$\boldsymbol{J} = \begin{pmatrix} \partial F_1/\partial x_1 & \partial F_1/\partial x_2 & \dots & \partial F_1/\partial x_n \\ \partial F_2/\partial x_1 & \partial F_2/\partial x_2 & \dots & \partial F_2/\partial x_n \\ \vdots & \ddots & \ddots & \vdots \\ \partial F_n/\partial x_1 & \partial F_n/\partial x_2 & \dots & \partial F_n/\partial x_n \end{pmatrix} \tag{20}$$

for the m_{th} iteration. Using this notation, the backpropagation algorithm converges for ALRP values $\mu < 2$.

6. Experimental results

To examine and test the validity and the accuracy of the proposed method, sample systems of nonlinear algebraic equations were selected and solved using the neural network approach, and the results were compared against those obtained by other methods.In these simulations, the adaptive learning rate approach (ALR) with a hyperbolic tangent activation function is considered as the primary algorithm, but the network is also tested with a fixed learning rate value as well as an identity output function. Even though in the theoretical analysis and the construction of the associated equations the classical backpropagation algorithm was used, the simulations showed that the execution time can be further decreased (leading to the speedup of the simulation process) if in each cycle the synaptic weights were updated one after the other and the new output values were used as input parameters in the corrections associated with the next weight adaptation. Because the accuracy of the results found in the literature varies significantly, and because the comparison of the root values requires the same number of decimal digits, different tolerance values in the form $10^{-\text{tol}}$ were used, with a value of $\text{tol} = 12$ to give an accuracy of six decimal digits, a value of $\text{tol} = 31$ to give an accuracy of 15 decimal digits, and a value of $\text{tol} = 20$ to give an accuracy of 10 decimal digits.

According to the proposed method, the condition that ensures the convergence of the backpropagation algorithm is given by the inequality $\mu < 2$, where μ is the ALRP. Therefore, to test the validity of this statement, a lot of simulations were performed, with the value of ALRP varying between $\mu = 0.1$ and $\mu = 1.9$ with a variation step equal to 0.1. The maximum allowed number of iterations was set to $N = 1000$, and a training procedure that reaches this limit is considered to be unsuccessful. In all cases, the initial conditions is a set in the form $[x_1(0), x_2(0), \dots, x_n(0)]$, and the search region is an n-dimensional region defined as $-\alpha \le x_i \le \alpha$ $(i = 1, 2, \dots, n)$. In almost all cases, the variation step of the system variables is equal to 0.1 or 0.2 even though

the values of 0.5 and 1.0 were also used for large α values to reduce the simulation time. The main graphical representation of the results shows the variation of the minimum and the mean iteration number with respect to the value of the adaptive learning rate parameter μ.

Because in some cases the modified adaptive learning rate (MARL) method gave better results, it was also used as an alternative approach. To distinguish between these two approaches, we call them ALR and MARL according to the expression of the energy function to be minimized. Therefore, we have

$$E = \begin{cases} \frac{1}{2}\sum_{i=1}^{n} \tanh^2[F_i(\boldsymbol{x})] & \text{for ALR method} \\ \frac{1}{2}\sum_{i=1}^{n} F_i^2(\boldsymbol{x}) & \text{for MALR method} \end{cases}$$

where n is the dimensionality of the system. In some cases the energy function

$$E = \frac{1}{2}\sum_{i=1}^{n} F_i^2(\boldsymbol{x}) + \frac{1}{2}\sum_{i=1}^{n} \tanh^2[F_i(\boldsymbol{x})]$$

was also used. This hybrid approach is described as the ALR12 method.

After the description of the experimental conditions, let us now present seven example systems as well as the experimental results emerged for each one of them. In the following presentation, the roots of the example systems are identified and compared with the roots estimated by the other methods.

Example 1

Consider the following system of two nonlinear algebraic equations with two unknowns x_1, x_2 defined as

$$F_1(x_1, x_2) = -\sin(x_1)\cos(x_2) - 2\cos(x_1)\sin(x_2) = 0$$
$$F_2(x_1, x_2) = -\cos(x_1)\sin(x_2) - 2\sin(x_1)\cos(x_2) = 0$$

(this problem has been borrowed by [38], see also [39, 40], and [41]). It can be proven that this system has 13 roots in the interval $0 \leq x_1, x_2 \leq 2\pi$. To estimate those roots, the neural solver run in this search region with variation steps $\Delta x_1 = \Delta x_2 = 0.2$ and $\Delta x_1 = \Delta x_2 = 0.5$ and identified all these roots using a tolerance value tol $= 12$. All the simulation runs identified the 13 roots regardless of the initial conditions with an accuracy of 100% with minimum iteration numbers 9–13 for an ALRP value equal to 1.3. The variation of the minimum iteration number with respect to the ALRP parameter for the ALR method is shown in Figure 2, while the results associated with the MALR method are characterized by a similar variation.

Example 2

Consider the following system of two nonlinear algebraic equations with two unknowns x_1, x_2 defined as

$$F_1(x_1, x_2) = \frac{1}{2}\sin(x_1 x_2) - \frac{x_2}{4\pi} - \frac{x_1}{2}$$
$$F_2(x_1, x_2) = \left(1 - \frac{1}{4\pi}\right)(e^{2x_1} - e) + \frac{e}{\pi}x_2 - 2ex_1$$

(this problem has also been borrowed by [38], see also [42] and [43]), where $x_1 \in [0.25, 1]$ and $x_2 \in [1.5, 2\pi]$. It has been proven that the aforementioned system has two roots in this domain. The neural solver identified these two roots using a tolerance value of tol $= 12$

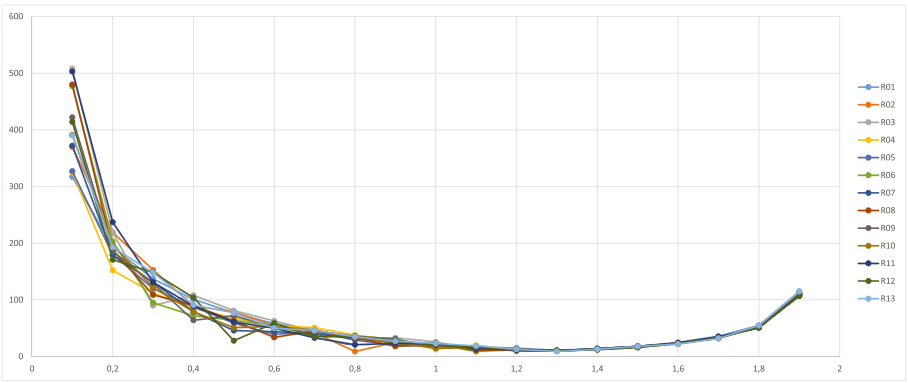

Figure 2. The variation of the minimum iteration number with respect to the value of the adaptive learning rate parameter for the example system 1 and for the adaptive learning rate method.

and variation steps $\Delta x_1 = 0.05$ and $\Delta x_2 = 0.25$. As in previous cases, all the simulation runs identified all the roots regardless of the initial conditions with a success rate greater than 64%. The best results are associated with 25 iterations and $ALRP = 1.6$ for root1 and six iterations with an $ALRP = 1.1$ for root2. The variation of the minimum iteration number with respect the ALRP for the ALR method is shown in Figure 3.

Example 3

The next example is a system of three nonlinear algebraic equations with three unknowns (x_1, x_2, x_3, x_4) defined as

$$F_1(x_1, x_2, x_3) = 15x_1 + x_2^2 - 4x_3 - 13 = 0$$
$$F_2(x_1, x_2, x_3) = x_1^2 + 10x_2 - e^{-x_3} - 11 = 0$$
$$F_3(x_1, x_2, x_3) = x_2^3 - 25x_3 + 22 = 0$$

(this is the example system 5 in [44], see also [45]). The system has been solved using the initial condition $(x_1, x_2, x_3) = (5, 4, 2)$, and the simulation results together with the results reported by Hafiz and Bahgat are shown in Table I. Note that in order to solve the system, each term in each equation was divided with the corresponding constant term, because the hyperbolic tangent function for large values of arguments tends to ± 1 and the solver does not work correctly. The variation of the minimum and the average iteration number with respect to the ALRP for the ALR method are shown in Figure 4

In a more detailed description, the use of the initial condition $(x_1, x_2, x_3) = (5, 4, 2)$ leads to an almost identical result compared with the one reported by Hafiz after 29 iterations (ALR method with $ALRP = 1.0$). Even though Hafiz's method reaches the same result after only five iterations, it evaluates four times the inverse Jacobian matrix, a process associated with high computational cost. The situation is improved in the search region $[-2, +2]$, where the MARL method for an ALRP equal to 1.0 requires only 14 iterations.

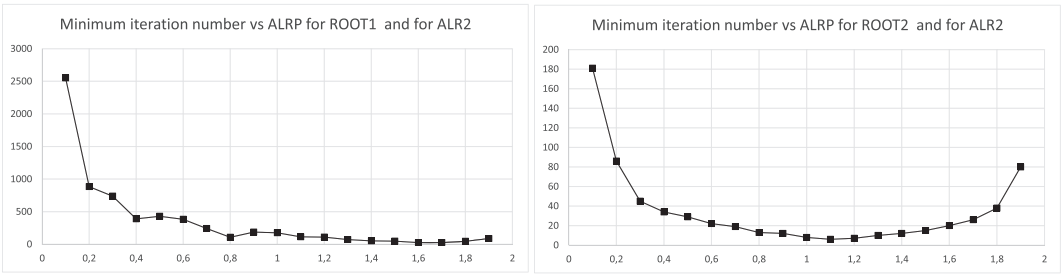

Figure 3. The variation of the minimum iteration number with respect to the value of the adaptive learning rate parameter (ALRP) for the example system 2 and for the adaptive learning rate (ALR) method.

Table I. Simulation results for the example system 2 (Example 5 of Hafiz and Bahgat).

	HAFIZ (Example 5)	MALR	ALR	ALR12
x1	1.0421495605769300	1.0421495605769660	1.0421495605768980	1.0421495605768980
x2	1.0310912718394000	1.0310912718394010	1.0310912718394180	1.0310912718394180
x3	0.9238481548793670	0.9238481548793740	0.9238481548793560	0.9238481548793560
F1	−0.0000000000001297	0.0000000000000298	−0.0000000000000404	−0.0000000000000404
F2	−0.0000000000000409	0.0000000000000042	0.0000000000000064	0.0000000000000064
F3	0.0000000000000142	−0.0000000000000071	0.0000000000000158	0.0000000000000158
ABSERR	0.0000000000001847	0.0000000000000411	0.0000000000000626	0.0000000000000626

MALR, modified adaptive learning rate; ALR, adaptive learning rate; ALRP, adaptive learning rate parameter.

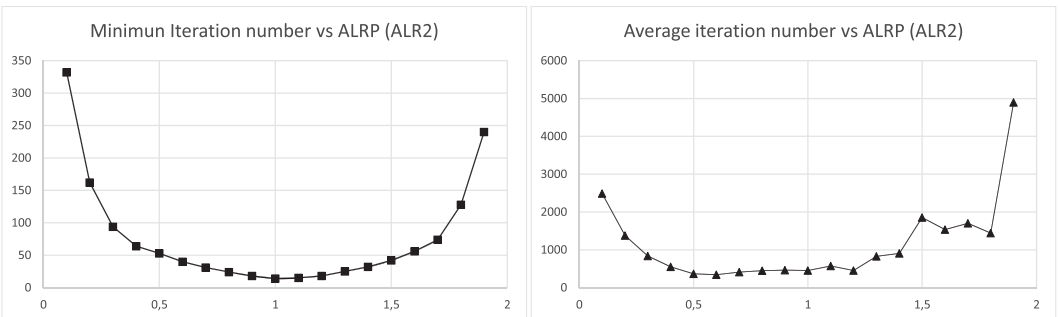

Figure 4. The variation of the minimum and the average iteration number with respect to the value of the adaptive learning rate parameter (ALRP) for the example system 3.

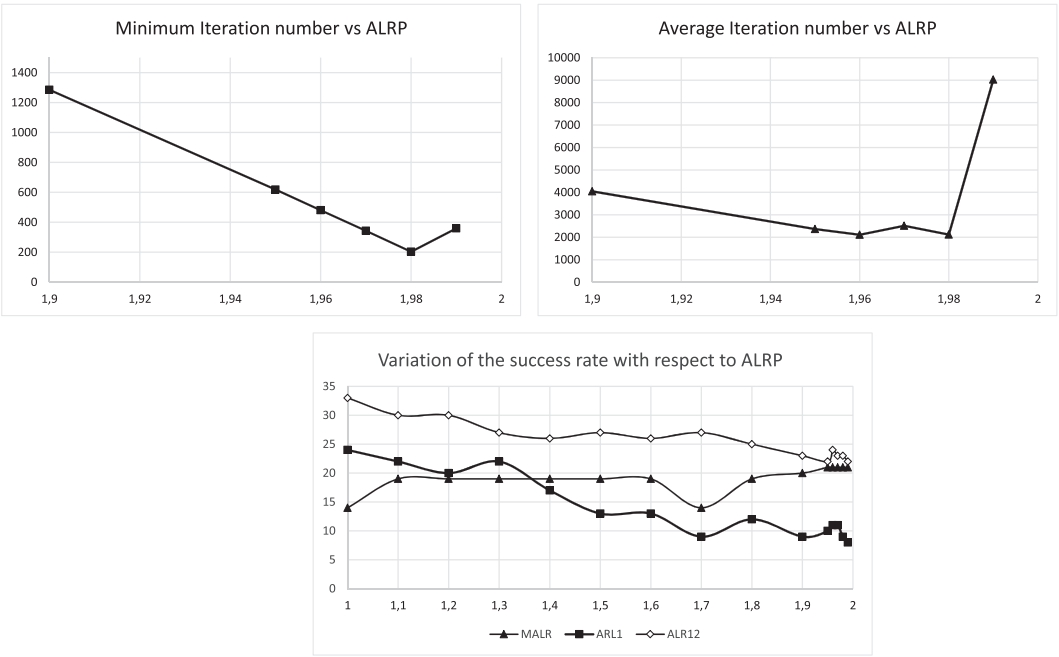

Figure 5. The variation of the minimum and the average iteration number with respect to the value of the adaptive learning rate parameter (ALRP) for the example system 4.

Example 4

The next system has four equations with four unknowns (x_1, x_2, x_3, x_4) and it is defined as

$$F_1(x_1, x_2, x_3, x_4) = 3 - x_1 x_3^2 = 0$$
$$F_2(x_1, x_2, x_3, x_4) = x_3 \sin(\pi / x_2) - x_3 - x_4 = 0$$
$$F_3(x_1, x_2, x_3, x_4) = -x_2 x_3 \exp(1 - x_1 x_3) + 0.2707 = 0$$
$$F_4(x_1, x_2, x_3, x_4) = 2x_3 x_1^2 - x_3 x_2^4 - x_2 = 0$$

(this example is described by the Equation (15) in [13]). This system, according to the literature is non-differentiable, and the method described in [13] identifies a unique solution $x_0 = (3, 2, 1, 0)$. The proposed neural solver was able to identify the same solution with an accuracy of six decimal digits, and the variation of the minimum as well as the average number of iterations with respect to the ALRP parameter are presented in Figure 5. Because in this case the network converges for an ALRP value $\mu = 1.98$, the horizontal axis has been configured accordingly. The minimum iteration number of ALR as well as MALR is equal to 204.

Example 5

The next example describes a system of seven equations with seven unknowns x_i $(i = 1, 2, \ldots, 7)$ defined as

$$f_i(\boldsymbol{x}) = e^{x_i} - 1, \quad i = 1, 2, \ldots, 7$$

where $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ (this is the example system 7 in [44], see also [45]). The unique solution of this system is $\boldsymbol{x} = (0, 0, 0, 0, 0, 0, 0)$. The system was run with 2187 combinations of initial conditions, namely, from $\boldsymbol{x}_0 = (-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5)$ to $\boldsymbol{x}_0 = (0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$ and with a variation step $\Delta x_i = 0.5$ $(i = 1, 2, 3, 4, 5, 6, 7)$ (therefore each x_i was assigned to the values -0.5, 0.0 and 0.5). In all cases, the unique root of the system was estimated with an accuracy of six decimal digits.

The best simulation run in this example is associated with the value $\mu = 1.0$, and the root for the case of the MARL method was reached after three iterations. To compare the results with those reported by Hafiz, the initial condition vector $\boldsymbol{x}_0 = (0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$ was used and the results were four iterations for ALR and five iterations for MARL. Hafiz reported three iterations for this case, but the proposed approach is associated with fewer mathematical operations and therefore smaller computational cost.

Figure 6 shows the variation of the minimum and the average iteration number with respect to the ALRP parameter and for the MALR method (the ALR method gave almost the same results).

Example 6

Finally, consider a system of eight nonlinear algebraic equations with eight unknowns x_i $(i = 1, 2, \ldots, 8)$ such as

$$f_i(\boldsymbol{x}) = x_i^2 - \cos(x_i - 1), \quad i = 1, 2, \ldots, 8$$

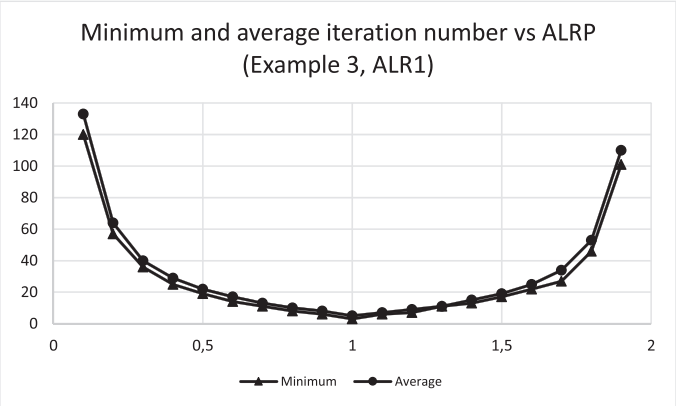

Figure 6. The variation of the minimum and the average iteration number with respect to the value of the adaptive learning rate parameter (ALRP) for the example system 5 and for the modified adaptive learning rate (MALR) method.

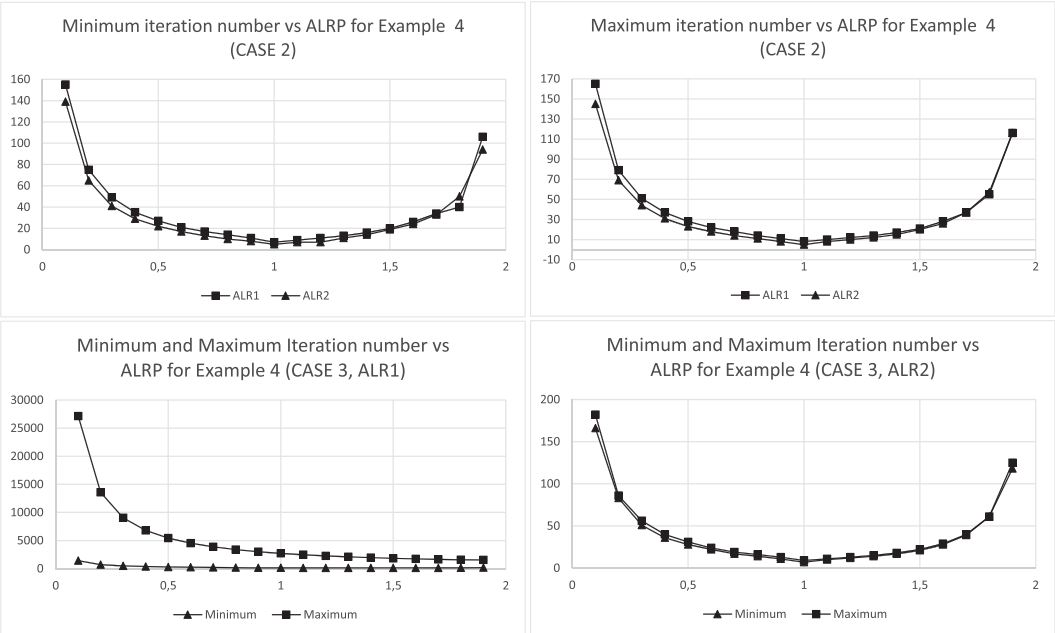

Figure 7. The variation of the minimum and the maximum iteration number with respect to the value of the adaptive learning rate parameter (ALRP) for the example system 6 (CASE 2 and CASE 3).

where $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ (this is the example system 8 in [44], see also [45]). The roots of this system in the interval $[-2, 2]$ are the $2^8 = 256$ vertices of the eight-dimensional hypercube $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ with each x_i $(i = 1, 2, \ldots, 8)$ to have either the value -0.405564 or the value 1.000000. The neural solver was used with a lot of different initial conditions, and the results in short are the following:

- CASE 1: the initial conditions are the components of the 256 roots. The solver identified all the 256 roots.
- CASE 2: the initial conditions are spread uniformly in the interval $[-1, 1]$ with a variation step equal to $h = 2$. This means that each x_i $(i = 1, 2, \ldots, 8)$ gets either the value $+1$ or the value -1. The solver identified 192 of the 256 roots (namely, a percentage of 75%).
- CASE 3: the initial conditions are spread uniformly in the interval $[-2, 2]$ with a variation step equal to $h = 4$. This means that each x_i $(i = 1, 2, \ldots, 8)$ gets either the value $+2$ or the value -2. The solver identified 128 of the 256 roots (namely, a percentage of 50%).

Note, that in almost all cases, the iterations for ALR and MALR methods are almost the same for all the initial conditions. Figure 7 depicts the variation of the minimum and the maximum iteration number with respect to the ALRP for example system 6 and for the methods ALR and MALR. Note the large variation between the minimum and the maximum iteration number for the method ALR in CASE 3.

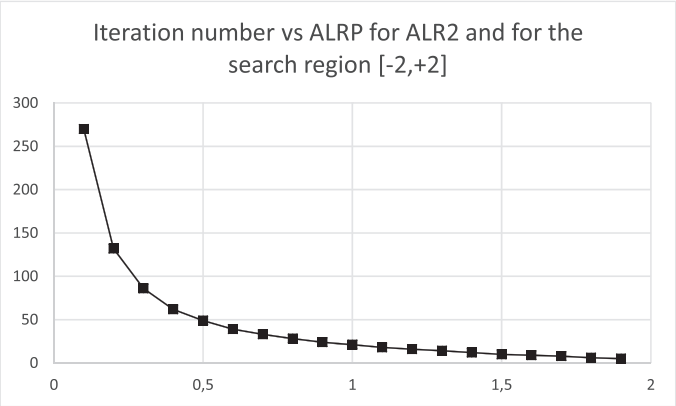

Figure 8. The variation of the iteration number with respect to the value of the adaptive learning rate parameter (ALRP) for the example system 7 and for the search region $[-2, +2]$.

To compare the results of our method with the results mentioned in Hafiz, the neural solver run with the same initial condition $\boldsymbol{x}_0 = (2, 2, 2, 2, 2, 2, 2, 2)$. Both methods converged to the same root found by Hafiz with a value of $(1, 1, 1, 1, 1, 1, 1, 1)$ and the best results emerged from ALR method for an ALRP value of $\mu = 1.0$ and a minimum iteration number equal to 7.

Example 7

Consider the following system of nine nonlinear algebraic equations with nine unknowns x_i $(i = 1, 2, \ldots, 9)$ defined as

$$f_i(\boldsymbol{x}) = \cos(x_i) - 1, \quad i = 1, 2, \ldots, 9$$

where $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$ (this is the example system 9 in [44], see also [45]). The network was run twice with 512 different initial condition combinations. In the first run, each one of the x_i $(i = 1, 2, \ldots, 9)$ was assigned to the values of -1 and $+1$ (in an equivalent description, the network run in the interval $[-1, +1]$ with a variation step $h = 2$). In the second run, each one of the x_i $(i = 1, 2, \ldots, 9)$ was assigned to the values of -2 and $+2$ (in an equivalent description the network run in the interval $[-2, +2]$ with a variation step $h = 4$). The tolerance value used in these cases was $tol = 24$, a fact that allowed the estimation of the system root with an accuracy of six decimal digits, while the values of the $F_i(\boldsymbol{x})$ functions $(i = 1, 2, \ldots, 9)$ were estimated with an accuracy of 12 decimal digits. The unique root of the system identified by the network (because of the values of the initial condition) has the value $\boldsymbol{x} = (0, 0, 0, 0, 0, 0, 0, 0, 0)$, but of course, it is well known that the system has infinite roots in the form of $x_i = 2\pi m$ $(i = 1, 2, \ldots, 9,\ m = 0, 1, 2, 3, \ldots)$. The iteration number with respect to the ALRP for the ALR method and for the search region $[-2, +2]$ is shown in Figure 8. Note that in this case, the number of iterations for each initial condition was the same.

To compare the neural based results with the results reported in the literature, the initial condition vector $\boldsymbol{x}_0 = (2, 2, 2, 2, 2, 2, 2, 2, 2)$ was also used. In this case, the minimum number of iterations was equal to 8 for the ALR method (for the value ALRP $= 1.9$) and equal to 5 for the MALR method (for the value ALRP $= 1.9$).

It is important to note that the presented approach has been tested to various systems of equations of degree from $n = 2$ to $n = 9$. The examples used in the simulations have been borrowed from the literature. Particularly, the large systems, whose equations were used here, are determined by the same function. This limitation does not have any influence to the convergence of the algorithm, because the convergence and the CPU time (or equivalently the number of iterations needed for convergence) depends not only on the dimension of the system, namely, the number of equations, but also on the complexity of each function and the value of the ALRP, which determines the value of the learning rate (because we are using a gradient descent algorithm). For instance, with an $ALRP = 1.5$, the system in Example 1 with $n = 2$ equations needs 17 to 20 iterations (best value of $ALRP = 1.3$ with 9–13 iterations), whereas the system in Example 4 with $n = 4$ equations needs 8177 iterations (best value of $ALRP = 1.98$ with 204 iterations), the system in Example 5 with $n = 7$ equations needs 15 to 17 iterations (best value of $ALRP = 1.0$ with four iterations), but the system in Example 7 with $n = 9$ equations needs 13 iterations (best value of ALRP $= 1.9$ with eight iterations). From these results, we are not able to derive any direct relationship between the dimension of the system and the number of iterations needed for convergence. On the contrary, for every one of the examples used, we always find some ALRP values for which the system converges to the exact solution with a minimum number of iterations.

7. Conclusions

The objective of this research was the design and performance evaluation of a neural network architecture, capable of solving a complete system of n nonlinear algebraic equations with n unknowns. The novel attribute of this approach is the adaptive learning rate for the neurons of the first layer as well as the use of the hyperbolic tangent function as the activation function of the output neurons. The developed theory shows that the network must be used with an adaptive learning rate parameter $\mu < 2$. The network was tested for solving seven example systems with increased dimensionality and in all cases was able to identify all the available roots in the search

region with an exceptional accuracy, because the solutions we found by using our algorithm were accurate to at least six decimal digits. Challenges for future research include the use of the network for the identification of multiple real and complex roots as well as the identification of all roots in only one simulation run.

Acknowledgements

The research of K. Goulianas, A. Margaris, I. Refanidis, and K. Diamantaras has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program 'Education and Lifelong Learning' of the National Strategic Reference Framework (NSRF) – Research Funding Program: THALES, Investing in knowledge society through the European Social Fund.

References

1. Dolotin V, Morozov A. *Introduction to Nonlinear Algebra*. World Scientific Publishing Company, 2007.
2. Burden R, Faires J. *Numerical Analysis*. Brooks Cole, 2010.
3. Broyden C. A class of methods for solving nonlinear simultaneous equations. *Mathematical Computations* 1965; **19**:577–593.
4. Dennis J, Wolkowicz H. Least change secant method, sizing and shifting. *SIAM Journal on Numerical Analysis* 1993; **30**:1291–1314.
5. Hentenryck P, McAllester D, Kapur D. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis* 1997; **34**(2):797–827.
6. Abaffy J, Spedicato E. *ABS Projection Algorithms: Mathematical Techniques for Linear and Nonlinear Equations*. Ellis Horwood, 1989.
7. Abaffy J, Galantai A. Conjugate direction methods for linear and nonlinear systems of algebraic equations. In *Numerical Methods*, Vol. 50, Rezsa P, Greenspan D (eds). Colloquia Mathematica Soc: Amsterdam, 1987; 481–502.
8. Abaffy J, Galantai A, Spedicato E. The local convergence of ABS methods for nonlinear algebraic equations. *Numerische Mathematik* 1987; **51**: 429–439.
9. Galantai A, Jeney A. Quasi-Newton ABS methods for solving nonlinear algebraic systems of equations. *Journal of Optimization Theory and Applications* 1996; **89**(3):561–573.
10. Ren H, Wu L, Bi W, Argyros IK. Solving nonlinear equations system via an efficient genetic algorithm, with symmetric and harmonious individuals. *Applied Mathematics and Computations* 2013; **219**:10967–10973.
11. Kuri-Morales A. Solution of simultaneous nonlinear equations using genetic algorithms. *WSEAS Transactions on Systems* 2003; **2**:44–51.
12. Nasira G, Devi D. Solving nonlinear equations through Jacobian sparsity patterns using genetic algorithms. *International Journal of Communications and Engineering* 2012; **5**:78–82.
13. Pourjafari E, Mojallali H. Solving nonlinear equation systems with a new approach based on invasive weed optimization algorithm and clustering. *Swarm and Evolutionary Computation* 2012; **4**:33–43.
14. Oliveira H, Petraglia A. Solving nonlinear systems of functional equations with fuzzy adaptive simulated annealing. *Applied Soft Computing* 2013; **13**:4349–4357.
15. Effati S, Nazemi A. A new method for solving a system of the nonlinear equations. *Applied Mathematics and Computations* 2005; **168**:877–894.
16. Grosan C, Abraham A. A new approach for solving nonlinear equation systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans* 2008; **38**(3):698–714.
17. Liu H, Zhou Y, Li Y. A quasi-Newton population migration algorithm for solving systems of nonlinear equations. *Journal of Computers* 2011; **6**(1): 36–42.
18. Zhou Y, Mao Z. A new search algorithm for global optimization – population migration algorithm. *Journal of South China* 2003; **21**(3):1–5.
19. Broyden C, Dennis J, More J. On the local and the superlineal convergences of quasi-Newton methods. *Journal of the Institute of Mathematics and its Applications* 1993; **12**:223–246.
20. Mathia K, Saeks R. Solving nonlinear equations using recurrent neural networks. *Proceedings of World Congress on Neural Networks (WCNN'95)*, Washington, DC, USA, 1995, 1.76–1.80.
21. Anastassiou G. Rate of convergence of some neural network operators to the unit-univariate case. *Journal of Mathematical Analysis and Applications* 1997; **212**:237–262.
22. Anastassiou G. Univariate hyperbolic tangent neural network approximation. *Mathematical and Computer Modelling* 2001; **53**:1111–1132.
23. Anastassiou G. Multivariate hyperbolic tangent neural network approximation. *Computers & Mathematics with Applications* 2011; **61**:809–821.
24. Anastassiou G. Intellingent systems: approximation by artificial neural networks. *Intelligent Systems Reference Library* 2011; **19**:108.
25. Barron A. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory* 1993; **39**(3):930–945.
26. Costarelli D. Interpolation by neural network operators activated by ramp functions. *Journal of Mathematical Analysis and Application* 2014; **419**: 574–582.
27. Costarelli D, Spigler R. Approximation results for neural networks operators activated by sigmoidal functions. *Neural Networks* 2013; **44**:101–106.
28. Costarelli D, Spigler R. Multivariate neural networks operators activated with sigmoidal activation functions. *Neural Networks* 2013; **48**:72–77.
29. Costarelli D, Spigler R. Convergence of a family of neural network operators of the Kantorovich type. *Journal of Approximation Theory* 2014; **185**: 80–90.
30. Costarelli D, Spigler R. Approximation by series of sigmoidal functions with applications to neural networks. *Annali di Matematica Pura a Applicata* 2015; **194**(1):289–306.
31. Cybenko G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems* 1989; **2**:303–314.
32. Ismailov V. On the approximation by neural networks with bounded number of neurons in hidden layers. *Journal of Mathematical Analysis and Applications* 2014; **417**(2):963–969.
33. Zhao Q, Li W. An improved iterative algorithm of neural network for nonlinear equation groups. *Proceedings of IEEE 2nd International Conference on Busines Computing and Global Informatization*, Washington, DC, USA, 2012, 522–525.
34. Meng A, Zeng Z. A neural computational method to solve nonlinear equation systems. *Journal of Computational Information Systems* 2011; **7**: 3462–3469.

35. Margaris A, Adamopoulos M. Solving nonlinear algebraic systems using artificial neural networks. *Proceedings of the 10th International Conference on Engineering Applications of Artificial Neural Networks*, Thessaloniki, Greece, 2007, 107–120.
36. Margaris A, Goulianas K. Finding all roots of 2 × 2 nonlinear algebraic systems using backpropagation neural networks. *Neural Computing and Applications* 2012; **21**(5):891–904.
37. Goulianas K, Margaris A, Adamopoulos M. Finding all real roots of 3*x*3 nonlinear algebraic systems using neural networks. *Applied Mathematics and Computation* 2013; **219**(9):4444–4464.
38. Tsoulos IG, Stavrakoudis A. On locating all roots of systems of nonlinear equations inside bounded domain using global optimization methods. *Nonlinear Analysis: Real World Applications* 2010; **11**:2465–2471.
39. Merlet JP. The coprin examples page, 2006. Available from: http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html [Accessed on 14, September 2014].
40. Hirsch MJ, Resende MCG, Pardalos PM. Solving systems of nonlinear equations with continuous grasp. *Nonlinear Analysis: Real World Applications* 2009; **10**:2000–2006.
41. Silva RMA, Resende MGC, Pardalos PM. Finding multiple roots of box-constrained system of nonlinear equations with a biased random-key genetic algorithm. *Journal of Global Optimization* 2014; **60**:289–306.
42. Floudas C. Recent advances in global optimization for process synthesis and control: enclosure of all solutions. *Computers and Chemical Engineering* 1999; **963**:963–973.
43. Floudas CA, Pardalos PM, Adjiman CS, Esposito WR, Gumus ZH, Harding ST, Klepeis JL, Meyer CA, Schweiger CA. *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers: Dordrecht, 1999.
44. Hafiz MA, Bahgat MSM. An efficient two-step iterative method for solving systems of nonlinear equation. *Journal of Mathematical Research* 2012; **4**:28–34.
45. Darvishi MT, Shin B. High-order Newton–Krylov methods to solve systems of nonlinear equations. *Journal of the Korean Society for Industrial and Applied Mathematics* 2011; **15**:19–30.