Applied Mathematics and Computation 219 (2013) 4444–4464

Contents lists available at SciVerse ScienceDirect

Applied Mathematics and Computation

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

Finding all real roots of 3×3 nonlinear algebraic systems using neural networks

Konstantinos Goulianas [a], Athanasios Margaris [b,*], Miltiades Adamopoulos [c]

[a] TEI of Thessaloniki, Department of Informatics, Thessaloniki, Greece
[b] TEI of Larissa, Department of Computer Science and Telecommunications, Greece
[c] University of Macedonia, Thessaloniki, Greece

ARTICLE INFO

Keywords:
Nonlinear algebraic systems
Neural networks
Numerical analysis
Computational methods

ABSTRACT

The objective of this research is the description of a feed-forward neural network capable of solving nonlinear algebraic systems with polynomials equations. The basic features of the proposed structure, include among other things, product units trained by the back-propagation algorithm and a fixed input unit with a constant input of unity. The presented theory is demonstrated by solving complete 3×3 nonlinear algebraic system paradigms, and the accuracy of the method is tested by comparing the experimental results produced by the network, with the theoretical values of the systems roots.

1. Introduction

A typical nonlinear algebraic system is defined as $F(\vec{z}) = 0$ with the mapping function $F : R^n \to R^n$ $(n > 1)$ to be described as an n-dimensional vector

$$F = [f_1, f_2, \ldots, f_n]^T, \tag{1}$$

where $f_i : R^n \to R\,(i = 1, 2, \ldots, n)$. Generally speaking, there are no good methods for solving such systems: even in the simple case of only two equations in the form $f_1(z_1, z_2) = 0$ and $f_2(z_1, z_2) = 0$, the estimation of the system roots is reduced to the identification of the common points of the zero contours of the functions $f_1(z_1, z_2)$ and $f_2(z_1, z_2)$. But this is a very difficult task, since in general, these two functions have no relation to each other at all. In the general case of N nonlinear equations, solving the system requires the identification of points that are mutually common to N unrelated zero-contour hypersurfaces each of dimension $N - 1$ [28].

2. Nonlinear algebraic systems

According to the basic principles of the nonlinear algebra [26], a complete nonlinear algebraic system of n polynomial equations with n unknowns $\vec{z} = (z_1, z_2, \ldots, z_n)$ is identified completely by the number of equations n, and their degrees $(s_1, s_2, \ldots, s_n)$, it is expressed mathematically as

$$F_i(\vec{z}) = \sum_{j_1 j_2 \cdots j_{s_i}}^{n} A_i^{j_1 j_2 \ldots j_{s_i}} z_{j_1} z_{j_2} \ldots z_{j_{s_i}} = 0 \tag{2}$$

* Corresponding author.
E-mail addresses: gouliana@it.teithe.gr (K. Goulianas), amarg@teilar.gr, amarg@uom.gr (A. Margaris), miltos@uom.gr (M. Adamopoulos).

0096-3003/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.amc.2012.10.049

$(i = 1, 2, \ldots, n)$, and it has one non-vanishing solution (i.e. at least one $z_j \neq 0$) if and only if the equation

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

holds. In this equation, the function $\mathfrak{R}$ is called the resultant and it is a straightforward generalization of the determinant of a linear system. The resultant $\mathfrak{R}$ is a polynomial of the coefficients of A of degree

$$d_{s_1,s_2,\ldots,s_n} = deg_A \mathfrak{R}_{s_1,s_2,\ldots,s_n} = \sum_{i=1}^{n}\left(\prod_{j\neq i} s_j\right) \tag{4}$$

When all degrees coincide, i.e. $s_1 = s_2 = \cdots = s_n = s$, the resultant $\mathfrak{R}_{n|s}$ is reduced to a simple polynomial of degree $d_{n|s} = deg_A \mathfrak{R}_{n|s} = ns^{n-1}$ and it is described completely by the values of the parameters n and s. It can be proven that the coefficients of the matrix A, which is actually a tensor for $n > 2$, are not all independent each other. More specifically, for the simple case $s_1 = s_2 = \cdots = s_n = s$, the matrix A is symmetric in the last s contra-variant indices and it contains only $nM_{n|s}$ independent coefficients, with

$$M_{n|s} = \frac{(n+s-1)}{(n-1)!\,s!}. \tag{5}$$

An interesting description concerning the existence of solution for a nonlinear algebraic system can be found in [30].

Even though the notion of the resultant has been defined for homogenous nonlinear equations, it can also describe non-homogenous algebraic equations as well. In the general case, the resultant $\mathfrak{R}$, satisfies the nonlinear Cramer rule

$$\mathfrak{R}_{s_1,s_2,\ldots,s_n}\{A^{(k)}(Z_k)\} = 0, \tag{6}$$

where Z_k is the k_{th} component of the solution of the no homogenous system, and $A^{(k)}$ is the k_{th} column of the coefficient matrix A.

3. Review of previous work

The solution of nonlinear algebraic systems is generally possible by using not analytical, but numerical algorithms. Besides the well known fixed-point based methods, (quasi)-Newton and gradient descent methods, a well known class of such algorithms is the ABS algorithms introduced in 1984 by Abaffy, Broyden, and Spedicato [10] to solve linear systems as well as nonlinear equations and system of equations [11,4]. The basic function of the initial ABS algorithms is to solve a determined or under-determined $n \times m$ linear system $Az = b$ $(z \in R^n, b \in R^m, m \leqslant n)$ by using special matrices, known as Abaffians. The choice of those matrices as well as the quantities used in their defining equations, determine particular sub-classes of the ABS algorithms, the most important of them are the conjugate direction subclass, the orthogonality scaled subclass as well as, the optimally stable subclass. The extension of the ABS methods for solving nonlinear algebraic systems is straightforward and it can be found in many sources such as [9,8]. It can be proven, that under appropriate conditions, the ABS methods are locally convergent with a speed of Q-order two, while, the computational cost of one iteration is $O(n^3)$ flops plus one function and one Jacobian matrix evaluation. To save the cost of Jacobian matrix evaluations, Huang [5] introduced quasi-Newton based ABS methods known as row update methods. These methods, do not require the a priori computation of the Jacobian matrix, and therefore its computational cost is $O(n^3)$.

Galantai and Jeney [1] have proposed alternative methods for solving nonlinear systems of equations that are combinations of the nonlinear ABS methods and quasi-Newton methods. Another class of methods has been proposed by Kublanovskaya and Simonova [27] for estimating the roots of m nonlinear coupled algebraic equations with two unknowns λ and μ. In their work, the nonlinear system under consideration is described by the algebraic equation $F(\lambda,\mu) = [f_1(\lambda,\mu), f_2(\lambda,\mu),\ldots,f_m(\lambda,\mu)]^T = 0$ with the function $f_k(\lambda,\mu)$ $(k = 1,\ldots,m)$ to be a polynomial in the form

$$f_k(\lambda,\mu) = [\alpha_{ts}^{(k)}\mu^t + \cdots + \alpha_{0s}^{(k)}]\lambda^s + \cdots + [\alpha_{t0}^{(k)}\mu^t + \cdots + \alpha_{00}^{(k)}]. \tag{7}$$

In this equation, the coefficients α_{ij} $(i = 0, 1, \ldots, t$ and $j = 0, 1, \ldots, s)$ are, in general, complex numbers, while s and t are the maximum degrees of polynomials in λ and μ respectively, found in $F(\lambda,\mu) = 0$. The algorithms proposed by Kublanovskaya and Simonova are capable of finding the zero-dimensional roots (λ^*,μ^*), i.e. the pairs of fixed numbers satisfying the nonlinear system, as well as the one-dimensional roots defined as $(\lambda,\mu) = [\varphi(\mu),\mu]$ and $(\lambda,\mu) = [\lambda,\tilde{\varphi}(\lambda)]$ whose components are functionally related.

The first method of Kublanovskaya and Simonova consists of two stages. At the first stage, the process passes from the system $F(\lambda,\mu) = 0$ to the spectral problem for a pencil $D(\lambda,\mu) = A(\mu) - \lambda B(\mu)$ of polynomial matrices $A(\mu)$ and $B(\mu)$, whose zero-dimensional and one-dimensional eigenvalues coincide with the zero dimensional and one dimensional roots of the nonlinear system under consideration. On the other hand, at the second stage, the spectral problem for $D(\lambda,\mu)$ is solved, i.e. all zero-dimensional eigenvalues of $D(\lambda,\mu)$ as well as a regular polynomial matrix pencil whose spectrum gives all one-dimensional eigenvalues of $D(\lambda,\mu)$ are found. Regarding the second method, it is based to the factorization of $F(\lambda,\mu)$ into irreducible factors and to the estimation of the roots (μ,λ) one after the other, since the resulting polynomials produced by this factorization are polynomials of only one variable.

Emiris, Mourrain, and Vrahatis [7] developed a method for the counting and identification of the roots of a nonlinear algebraic system based to the concept of topological degree and by using bisection techniques (for the application of those techniques see also [22]). A method for solving such a system using linear programming techniques can be found in [29], and another interesting method that uses evolutionary algorithms is described in [6]. There are many other methods concerning this subject; some of them are described in [2,25,24,12]. In the concluding section, the proposed neural model will be compared against some of those other methods such as the trust-region-dogleg [3], the trust-region-reflective [3] and the Levenberg–Marquardt algorithm [15,20], as well as the multivariate Newton–Raphson method with partial derivatives.

4. ANNs as nonlinear system solvers

Artificial neural networks (ANNs in short) have been used among other methods for solving linear algebra problems and simple linear systems and equations [14,13] as well as nonlinear equations and systems of nonlinear equations; however, they are not used so frequent as the methods described in the previous section, especially for the case of nonlinear systems. Mathia and Saeks [21] used recurrent neural networks composed of linear Hopfield networks to solve nonlinear equations which are approximated by a multilayer perceptron. Mishra and Kalra [23] used a modified Hopfield network with a properly selected energy function to solve a nonlinear algebraic system of m equations with n unknowns. Hopfield neural networks were also used by Luo and Han [17] to solve a nonlinear system of equations which in the previous stage has been transformed to a kind of quadratic optimization. Finally, Li and Zeng [16] used the gradient descent rule with a variable step size to solve nonlinear algebraic systems at very rapid convergence and with very high accuracy.

The main idea behind the proposed approach for solving a nonlinear algebraic system of p equations with p unknowns [18], is to construct a network with p output neurons, with the desired output of the ℓ_{th} neuron $(1 \leqslant \ell \leqslant p)$ to be equal to zero; in this way, the nonlinear algebraic system under consideration is simulated completely by the neural model. An efficient approach to construct such a network is shown in Fig. 1 for the case of a complete 2×2 nonlinear algebraic system [19]. In this figure, the proposed model is composed of four layers, an input layer, with one summation unit with a constant input equal to unity, a hidden layer of linear units that generate the linear terms x and y, a hidden layer of summation and product units that generate all the terms of the left hand part of the system equations except the fixed term – namely, the terms x, y, x^2, xy and y^2 – by using the appropriate activation function (for example, the activation function of the neuron

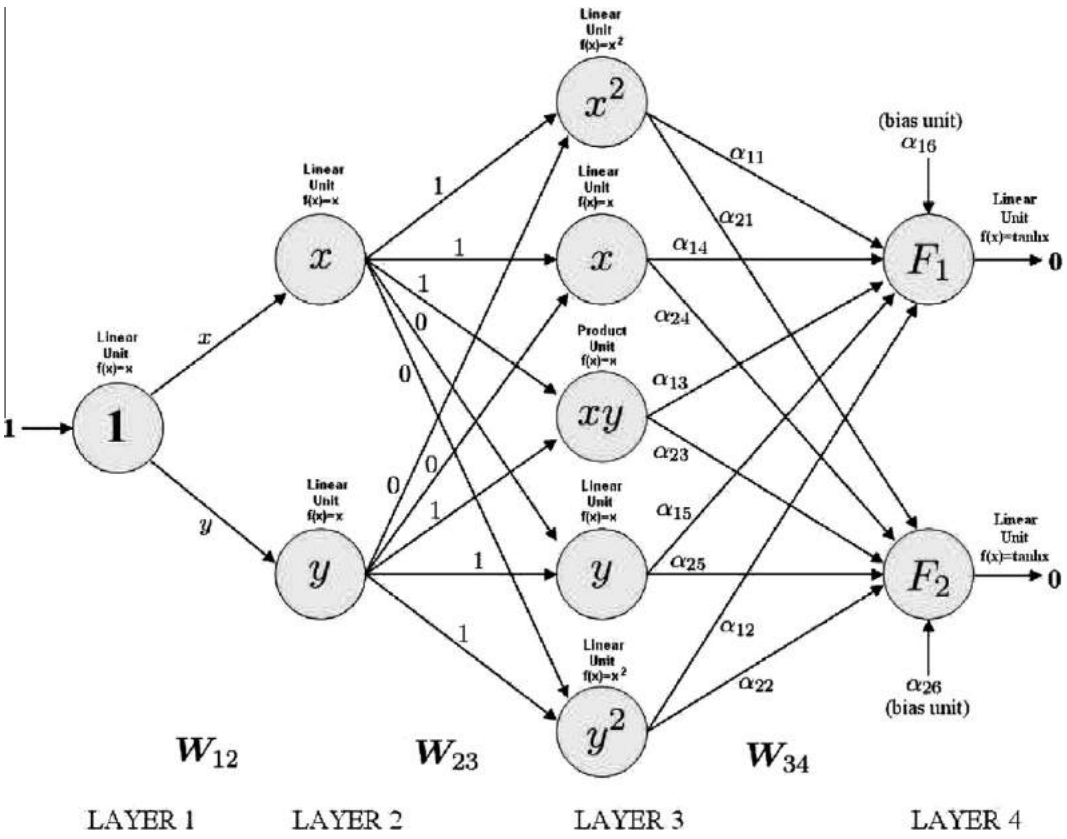

Fig. 1. The structure of the complete 2×2 nonlinear algebraic system neural solver.

producing the x^2 term is the function $f(x) = x^2$) and finally, an output layer of summation units whose total input is the expression of the left hand part of the corresponding equation. These output neurons uses the hyperbolic tangent as the activation function and have an additional input from a bias unit, whose fixed weight is the constant term of the associated nonlinear equation. To simulate the system, the synaptic weights are configured in the way shown in the figure. It is not difficult to note that the weights of synapses joining the neurons of the second layer to the neurons of the third layer, are fixed and equal to unity, while the weights joining the summation and the product units of the third layer to the neurons of the fourth (output) layer have been set to the fixed values of the coefficients of the system equations associated with the linear and nonlinear terms provided by the neurons (it is clear that if the algebraic system is not complete but there are missing terms, the corresponding synaptic weights are set to zero). The only variable-weight synapses are the two synapses that join the input neuron with the two linear neurons of the first hidden layer. Since this network structure produces as output a zero value, it is clear that a training operation with the traditional back propagation algorithm and the $[0\ \ 0]^T$ as the desired output vector, will assign to the weights of the two variable weight synapses, the components x and y of one of the system roots. The training of this network with a lot of initial conditions is capable of identifying all roots of the traditional 2×2 complete nonlinear algebraic systems with four distinct roots as well as the basin of attraction for each one of them and the basin of infinity that lead to a nonconverging behavior [19].

5. The neural model for the 3×3 complete nonlinear algebraic system

The complete 3×3 nonlinear algebraic system is defined as

$$F_i(x_1, x_2, x_3) = \alpha_{i,1}x_1^3 + \alpha_{i,2}x_2^3 + \alpha_{i,3}x_3^3 + \alpha_{i,4}x_1^2x_2 + \alpha_{i,5}x_1x_2^2 + \alpha_{i,6}x_1^2x_3 + +\alpha_{i,7}x_1x_3^2 + \alpha_{i,8}x_2^2x_3 + \alpha_{i,9}x_2x_3^2 + \alpha_{i,10}x_1x_2x_3 + \alpha_{i,11}x_1x_2 + +\alpha_{i,12}x_1x_3 + \alpha_{i,13}x_2x_3 + \alpha_{i,14}x_1^2 + \alpha_{i,15}x_2^2 + \alpha_{i,16}x_3^2 + \alpha_{i,17}x_1 + \alpha_{i,18}x_2 + \alpha_{i,19}x_3 - \alpha_{i,20} = 0$$

$(i = 1, 2, 3)$ and it is a straightforward generalization of the 2×2 complete nonlinear system studied in [19]. The neural structure that solves this type of system is based to the approach described in the previous section, it is described in Table 1 and it is shown in Fig. 2. In complete accordance with the proposed design, there are full connections between two consecutive layers; however, for the sake of clarity, the figure shows only the connections with nonzero synaptic weighs. These weights are defined as follows:

$$\boldsymbol{W}_{12} = \left[\boldsymbol{W}_{12}^{(1)}\ \ \boldsymbol{W}_{12}^{(2)}\ \ \boldsymbol{W}_{12}^{(3)}\right] = [x_1x_2x_3],$$

$$\boldsymbol{W}_{23} = \{\boldsymbol{W}_{2,3}^{(i,j)}\}_{i=1,2,3}^{j=1,2,3,\ldots,9} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_{34} = \{\boldsymbol{W}_{3,4}^{(i,j)}\}_{i=1,2,\ldots,9}^{j=1,2,3,\ldots,17,18,19} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

Table 1
The structure of the neural solver for the 3×3 complete nonlinear algebraic system.

Layer	Neurons	Neuron type	Weight table
Layer 1	01	Summation	$\boldsymbol{W}_{12}(1 \times 3)$
Layer 2	03	Summation	$\boldsymbol{W}_{23}(3 \times 9)$
Layer 3	09	Summation	$\boldsymbol{W}_{34}(9 \times 19)$
Layer 4	19	Product & Summation Neurons 1,2,3,8,9,10,17,18,19 Summation units Neurons 4,5,6,7,11,12,13,14,15,16 Product units	$\boldsymbol{W}_{45}(19 \times 3)$
Layer 5	03	Summation	Not available

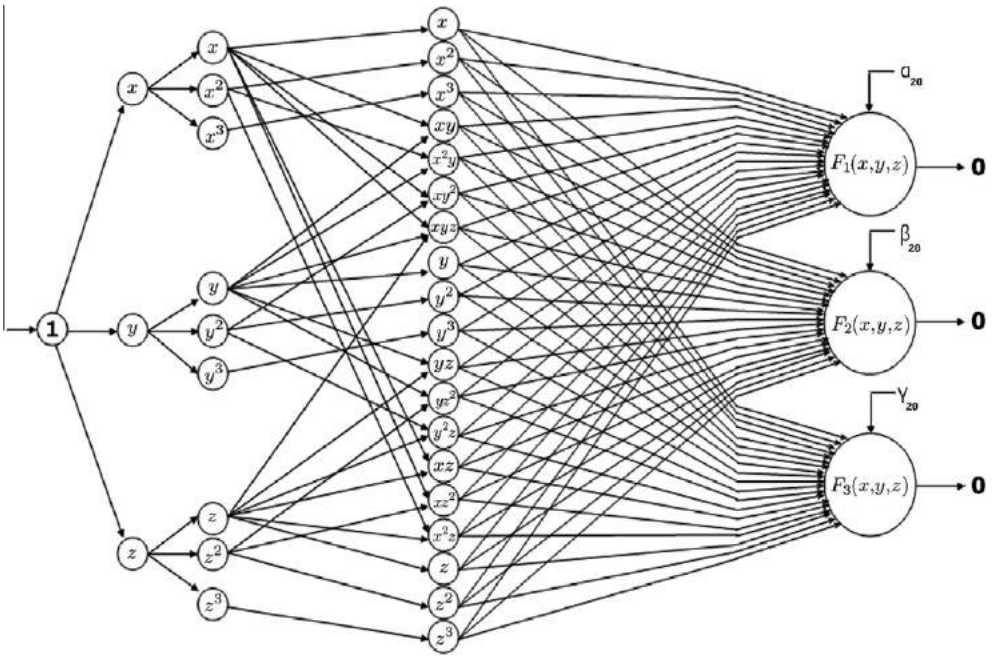

Fig. 2. The structure of the complete 3×3 nonlinear algebraic system neural solver. Since the back propagation algorithm requires full connections between consecutive layers, the missing synapses are modeled as synapses with zero weight values. For the sake of simplicity, the parameters x_1, x_2 and x_3 are denoted as x, y and z, respectively, while the parameters $\alpha_{i,20}$ $(i = 1, 2, 3)$ are denoted as α_{20}, β_{20} and γ_{20}.

$$\boldsymbol{W}_{45} = \begin{bmatrix} \alpha_{1,17} & \alpha_{2,17} & \alpha_{3,17} \\ \alpha_{1,14} & \alpha_{2,14} & \alpha_{3,14} \\ \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} \\ \alpha_{1,11} & \alpha_{2,11} & \alpha_{3,11} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} \\ \alpha_{1,5} & \alpha_{2,5} & \alpha_{3,5} \\ \alpha_{1,10} & \alpha_{2,10} & \alpha_{3,10} \\ \alpha_{1,18} & \alpha_{2,18} & \alpha_{3,18} \\ \alpha_{1,15} & \alpha_{2,15} & \alpha_{3,15} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} \\ \alpha_{1,13} & \alpha_{2,13} & \alpha_{3,13} \\ \alpha_{1,9} & \alpha_{2,9} & \alpha_{3,9} \\ \alpha_{1,8} & \alpha_{2,8} & \alpha_{3,8} \\ \alpha_{1,12} & \alpha_{2,12} & \alpha_{3,12} \\ \alpha_{1,7} & \alpha_{2,7} & \alpha_{3,7} \\ \alpha_{1,6} & \alpha_{2,6} & \alpha_{3,6} \\ \alpha_{1,19} & \alpha_{2,19} & \alpha_{3,19} \\ \alpha_{1,16} & \alpha_{2,16} & \alpha_{3,16} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} \end{bmatrix}.$$

The activation function for the neurons of the network structure are defined as follows:

- For the second layer: all neurons use the identity function; therefore we have

  $$\boldsymbol{f}_2^{(i)}(\upsilon) = \upsilon, \quad i = 1, 2, 3.$$

- For the third layer: the activation functions of the nine neurons are defined as follows:

  $$\boldsymbol{f}_3 = \left[\boldsymbol{f}_3^{(1)} \;\; \boldsymbol{f}_3^{(2)} \;\; \boldsymbol{f}_3^{(3)} \;\; \boldsymbol{f}_3^{(4)} \;\; \boldsymbol{f}_3^{(5)} \;\; \boldsymbol{f}_3^{(6)} \;\; \boldsymbol{f}_3^{(7)} \;\; \boldsymbol{f}_3^{(8)} \;\; \boldsymbol{f}_3^{(9)}\right] = \left[\upsilon \;\; \upsilon^2 \;\; \upsilon^3 \;\; \upsilon \;\; \upsilon^2 \;\; \upsilon^3 \;\; \upsilon \;\; \upsilon^2 \;\; \upsilon^3\right].$$

- For the fourth layer: all the neurons use the identity function, namely

$$\boldsymbol{f}_4^{(i)}(\upsilon) = \upsilon, \quad i = 1,2,3,\ldots,17,18,19.$$

- For the fifth (output) layer: all the neurons use the hyperbolic tangent function, namely

$$\boldsymbol{f}_5^{(i)}(\upsilon) = \tanh(\upsilon), \quad i = 1,2,3.$$

6. Building the back propagation equations

By following the traditional back propagation approach, the equation for training the neural network are constructed as follows:

6.1. Forward pass

The input to the three neurons of the second layer are estimated as

$$\boldsymbol{\upsilon}_2^{(i)} = \boldsymbol{W}_{12}^{(i)} = x_i, \quad i = 1,2,3$$

while the outputs produced by them have the values

$$\boldsymbol{v}_2^{(i)} = \boldsymbol{f}_2^{(i)}(\boldsymbol{\upsilon}_2^{(i)}) = \boldsymbol{f}_2^{(i)}(x_i) = x_i, \quad i = 1,2,3.$$

These outputs are sent as inputs to the neurons of the third layer, whose total input is thus given by the equation

$$\boldsymbol{\upsilon}_3^{(i)} = \sum_{j=1}^{3} \boldsymbol{W}_{23}^{(j,i)} \boldsymbol{v}_2^{(j)}, \quad j = 1,2,3,4,5,6,7,8,9$$

or in expanded form

$$\begin{aligned}
\boldsymbol{\upsilon}_3^{(1)} &= \boldsymbol{W}_{23}^{(1,1)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,1)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,1)}\boldsymbol{v}_2^{(3)} = x_1,\\
\boldsymbol{\upsilon}_3^{(2)} &= \boldsymbol{W}_{23}^{(1,2)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,2)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,2)}\boldsymbol{v}_2^{(3)} = x_1,\\
\boldsymbol{\upsilon}_3^{(3)} &= \boldsymbol{W}_{23}^{(1,3)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,3)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,3)}\boldsymbol{v}_2^{(3)} = x_1,\\
\boldsymbol{\upsilon}_3^{(4)} &= \boldsymbol{W}_{23}^{(1,4)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,4)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,4)}\boldsymbol{v}_2^{(3)} = x_2,\\
\boldsymbol{\upsilon}_3^{(5)} &= \boldsymbol{W}_{23}^{(1,5)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,5)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,5)}\boldsymbol{v}_2^{(3)} = x_2,\\
\boldsymbol{\upsilon}_3^{(6)} &= \boldsymbol{W}_{23}^{(1,6)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,6)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,6)}\boldsymbol{v}_2^{(3)} = x_2,\\
\boldsymbol{\upsilon}_3^{(7)} &= \boldsymbol{W}_{23}^{(1,7)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,7)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,7)}\boldsymbol{v}_2^{(3)} = x_3,\\
\boldsymbol{\upsilon}_3^{(8)} &= \boldsymbol{W}_{23}^{(1,8)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,8)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,8)}\boldsymbol{v}_2^{(3)} = x_3,\\
\boldsymbol{\upsilon}_3^{(9)} &= \boldsymbol{W}_{23}^{(1,9)}\boldsymbol{v}_2^{(1)} + \boldsymbol{W}_{23}^{(2,9)}\boldsymbol{v}_2^{(2)} + \boldsymbol{W}_{23}^{(3,9)}\boldsymbol{v}_2^{(3)} = x_3.
\end{aligned}$$

On the other hand, the outputs of the neurons of the third layer are estimated as

$$\begin{aligned}
\boldsymbol{v}_3^{(1)} &= \boldsymbol{f}_3^{(1)}(\boldsymbol{\upsilon}_3^{(1)}) = \boldsymbol{f}_3^{(1)}(x) = x_1,\\
\boldsymbol{v}_3^{(2)} &= \boldsymbol{f}_3^{(2)}(\boldsymbol{\upsilon}_3^{(2)}) = \boldsymbol{f}_3^{(2)}(x) = x_1^2,\\
\boldsymbol{v}_3^{(3)} &= \boldsymbol{f}_3^{(3)}(\boldsymbol{\upsilon}_3^{(3)}) = \boldsymbol{f}_3^{(3)}(x) = x_1^3,\\
\boldsymbol{v}_3^{(4)} &= \boldsymbol{f}_3^{(4)}(\boldsymbol{\upsilon}_3^{(4)}) = \boldsymbol{f}_3^{(4)}(y) = x_2,\\
\boldsymbol{v}_3^{(5)} &= \boldsymbol{f}_3^{(5)}(\boldsymbol{\upsilon}_3^{(5)}) = \boldsymbol{f}_3^{(5)}(y) = x_2^2,\\
\boldsymbol{v}_3^{(6)} &= \boldsymbol{f}_3^{(6)}(\boldsymbol{\upsilon}_3^{(6)}) = \boldsymbol{f}_3^{(6)}(y) = x_2^3,\\
\boldsymbol{v}_3^{(7)} &= \boldsymbol{f}_3^{(7)}(\boldsymbol{\upsilon}_3^{(7)}) = \boldsymbol{f}_3^{(7)}(z) = x_3,\\
\boldsymbol{v}_3^{(8)} &= \boldsymbol{f}_3^{(8)}(\boldsymbol{\upsilon}_3^{(8)}) = \boldsymbol{f}_3^{(8)}(z) = x_3^2,\\
\boldsymbol{v}_3^{(9)} &= \boldsymbol{f}_3^{(9)}(\boldsymbol{\upsilon}_3^{(9)}) = \boldsymbol{f}_3^{(9)}(z) = x_3^3.
\end{aligned}$$

These outputs are sent as inputs to the neurons of the fourth layer. For the summation neurons, their total input is estimated as

$$\boldsymbol{\upsilon}_4^{(i)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,i)} \boldsymbol{v}_3^{(j)}, \quad i = 1,2,3,8,9,10,17,18,19$$

or in expanded form

$$\boldsymbol{v}_4^{(1)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,1)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(1,1)} \boldsymbol{v}_3^{(1)}) = x_1,$$

$$\boldsymbol{v}_4^{(2)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,2)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(2,2)} \boldsymbol{v}_3^{(2)}) = x_1^2,$$

$$\boldsymbol{v}_4^{(3)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,3)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(3,3)} \boldsymbol{v}_3^{(3)}) = x_1^3,$$

$$\boldsymbol{v}_4^{(8)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,8)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(4,8)} \boldsymbol{v}_3^{(4)}) = x_2,$$

$$\boldsymbol{v}_4^{(9)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,9)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(5,9)} \boldsymbol{v}_3^{(5)}) = x_2^2,$$

$$\boldsymbol{v}_4^{(10)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,10)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(6,10)} \boldsymbol{v}_3^{(6)}) = x_2^3,$$

$$\boldsymbol{v}_4^{(17)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,17)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(7,17)} \boldsymbol{v}_3^{(7)}) = x_3,$$

$$\boldsymbol{v}_4^{(18)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,18)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(8,18)} \boldsymbol{v}_3^{(8)}) = x_3^2.$$

$$\boldsymbol{v}_4^{(19)} = \sum_{j=1}^{9} \boldsymbol{W}_{34}^{(j,19)} \boldsymbol{v}_3^{(j)} = (\boldsymbol{W}_{34}^{(9,19)} \boldsymbol{v}_3^{(9)}) = x_3^3$$

while the outputs of the product units of this fourth layer are estimated as

$$\boldsymbol{v}_4^{(i)} = \prod_{j \in S_i} \boldsymbol{W}_{34}^{(j,i)} \boldsymbol{v}_3^{(j)}, \quad i = 4, 5, 6, 7, 11, 12, 13, 14, 15, 16,$$

where S_i is the set of indices of the neurons of the third layer that are connected to the i_{th} neuron of the fourth layer ($i = 4, 5, 6, 7, 11, 12, 13, 14, 15, 16$). If we expand this equation we get a set of equations in the form

$$\boldsymbol{v}_4^{(4)} = (\boldsymbol{W}_{34}^{(1,4)} \boldsymbol{v}_3^{(1)})(\boldsymbol{W}_{34}^{(4,4)} \boldsymbol{v}_3^{(4)}) = x_1 x_2,$$
$$\boldsymbol{v}_4^{(5)} = (\boldsymbol{W}_{34}^{(2,5)} \boldsymbol{v}_3^{(2)})(\boldsymbol{W}_{34}^{(4,5)} \boldsymbol{v}_3^{(4)}) = x_1^2 x_2,$$
$$\boldsymbol{v}_4^{(6)} = (\boldsymbol{W}_{34}^{(1,6)} \boldsymbol{v}_3^{(1)})(\boldsymbol{W}_{34}^{(5,6)} \boldsymbol{v}_3^{(5)}) = x_1 x_2^2,$$
$$\boldsymbol{v}_4^{(7)} = (\boldsymbol{W}_{34}^{(1,7)} \boldsymbol{v}_3^{(1)})(\boldsymbol{W}_{34}^{(4,7)} \boldsymbol{v}_3^{(4)})(\boldsymbol{W}_{34}^{(7,7)} \boldsymbol{v}_3^{(7)}) = x_1 x_2 x_3,$$
$$\boldsymbol{v}_4^{(11)} = (\boldsymbol{W}_{34}^{(4,11)} \boldsymbol{v}_3^{(4)})(\boldsymbol{W}_{34}^{(7,11)} \boldsymbol{v}_3^{(7)}) = x_2 x_3,$$
$$\boldsymbol{v}_4^{(12)} = (\boldsymbol{W}_{34}^{(4,12)} \boldsymbol{v}_3^{(4)})(\boldsymbol{W}_{34}^{(8,12)} \boldsymbol{v}_3^{(8)}) = x_2 x_3^2,$$
$$\boldsymbol{v}_4^{(13)} = (\boldsymbol{W}_{34}^{(5,13)} \boldsymbol{v}_3^{(5)})(\boldsymbol{W}_{34}^{(7,13)} \boldsymbol{v}_3^{(7)}) = x_2^2 x_3,$$
$$\boldsymbol{v}_4^{(14)} = (\boldsymbol{W}_{34}^{(1,14)} \boldsymbol{v}_3^{(1)})(\boldsymbol{W}_{34}^{(7,14)} \boldsymbol{v}_3^{(7)}) = x_1 x_3,$$
$$\boldsymbol{v}_4^{(15)} = (\boldsymbol{W}_{34}^{(1,15)} \boldsymbol{v}_3^{(1)})(\boldsymbol{W}_{34}^{(8,15)} \boldsymbol{v}_3^{(8)}) = x_1 x_3^2,$$
$$\boldsymbol{v}_4^{(16)} = (\boldsymbol{W}_{34}^{(2,16)} \boldsymbol{v}_3^{(2)})(\boldsymbol{W}_{34}^{(7,16)} \boldsymbol{v}_3^{(7)}) = x_1^2 x_3$$

with the outputs of the fourth layer estimated as

$$\boldsymbol{v}_4^{(1)} = \boldsymbol{f}_4^{(1)}(\boldsymbol{v}_4^{(1)}) = \boldsymbol{f}_4^{(1)}(x) = x_1,$$
$$\boldsymbol{v}_4^{(2)} = \boldsymbol{f}_4^{(2)}(\boldsymbol{v}_4^{(2)}) = \boldsymbol{f}_4^{(2)}(x^2) = x_1^2,$$
$$\boldsymbol{v}_4^{(3)} = \boldsymbol{f}_4^{(3)}(\boldsymbol{v}_4^{(3)}) = \boldsymbol{f}_4^{(3)}(x^3) = x_1^3,$$
$$\boldsymbol{v}_4^{(4)} = \boldsymbol{f}_4^{(4)}(\boldsymbol{v}_4^{(4)}) = \boldsymbol{f}_4^{(4)}(xy) = x_1 x_2,$$
$$\boldsymbol{v}_4^{(5)} = \boldsymbol{f}_4^{(5)}(\boldsymbol{v}_4^{(5)}) = \boldsymbol{f}_4^{(5)}(x^2 y) = x_1^2 x_2,$$
$$\boldsymbol{v}_4^{(6)} = \boldsymbol{f}_4^{(6)}(\boldsymbol{v}_4^{(6)}) = \boldsymbol{f}_4^{(6)}(xy^2) = x_1 x_2^2,$$
$$\boldsymbol{v}_4^{(7)} = \boldsymbol{f}_4^{(7)}(\boldsymbol{v}_4^{(7)}) = \boldsymbol{f}_4^{(7)}(xyz) = x_1 x_2 x_3,$$
$$\boldsymbol{v}_4^{(8)} = \boldsymbol{f}_4^{(8)}(\boldsymbol{v}_4^{(8)}) = \boldsymbol{f}_4^{(8)}(y) = x_2,$$

$$\begin{aligned}
\boldsymbol{v}_4^{(9)} &= \boldsymbol{f}_4^{(9)}(\boldsymbol{v}_4^{(9)}) = \boldsymbol{f}_4^{(9)}(y^2) = x_2^2,\\
\boldsymbol{v}_4^{(10)} &= \boldsymbol{f}_4^{(10)}(\boldsymbol{v}_4^{(10)}) = \boldsymbol{f}_4^{(10)}(y^3) = x_2^3,\\
\boldsymbol{v}_4^{(11)} &= \boldsymbol{f}_4^{(11)}(\boldsymbol{v}_4^{(11)}) = \boldsymbol{f}_4^{(11)}(yz) = x_2x_3,\\
\boldsymbol{v}_4^{(12)} &= \boldsymbol{f}_4^{(12)}(\boldsymbol{v}_4^{(12)}) = \boldsymbol{f}_4^{(12)}(yz^2) = x_2x_3^2,\\
\boldsymbol{v}_4^{(13)} &= \boldsymbol{f}_4^{(13)}(\boldsymbol{v}_4^{(13)}) = \boldsymbol{f}_4^{(13)}(y^2z) = x_2^2x_3,\\
\boldsymbol{v}_4^{(14)} &= \boldsymbol{f}_4^{(14)}(\boldsymbol{v}_4^{(14)}) = \boldsymbol{f}_4^{(14)}(xz) = x_1x_3,\\
\boldsymbol{v}_4^{(15)} &= \boldsymbol{f}_4^{(15)}(\boldsymbol{v}_4^{(15)}) = \boldsymbol{f}_4^{(15)}(xz^2) = x_1x_3^2,\\
\boldsymbol{v}_4^{(16)} &= \boldsymbol{f}_4^{(16)}(\boldsymbol{v}_4^{(16)}) = \boldsymbol{f}_4^{(16)}(x^2z) = x_1^2x_3,\\
\boldsymbol{v}_4^{(17)} &= \boldsymbol{f}_4^{(17)}(\boldsymbol{v}_4^{(17)}) = \boldsymbol{f}_4^{(17)}(z) = x_3,\\
\boldsymbol{v}_4^{(18)} &= \boldsymbol{f}_4^{(18)}(\boldsymbol{v}_4^{(18)}) = \boldsymbol{f}_4^{(18)}(z^2) = x_3^2,\\
\boldsymbol{v}_4^{(19)} &= \boldsymbol{f}_4^{(19)}(\boldsymbol{v}_4^{(19)}) = \boldsymbol{f}_4^{(19)}(z^3) = x_3^3.
\end{aligned}$$

Finally, the total input to the three summation neurons of the output layer is estimated as

$$\begin{aligned}
\boldsymbol{v}_5^{(1)} &= \sum_{j=1}^{19} \boldsymbol{W}_{45}^{(j,1)} \boldsymbol{v}_4^{(j)} = \alpha_{1,17}x_1 + \alpha_{1,14}x_1^2 + \alpha_{1,1}x_1^3 + \alpha_{1,11}x_1x_2 + \alpha_{1,4}x_1^2x_2 + \alpha_{1,5}x_1x_2^2 + \alpha_{1,10}x_1x_2x_3 + \alpha_{1,18}x_2 + \alpha_{1,15}x_2^2\\
&\quad + \alpha_{1,2}x_2^3 + \alpha_{1,13}x_2x_3 + \alpha_{1,9}x_2x_3^2 + \alpha_{1,8}x_2^2x_3 + \alpha_{1,12}x_1x_3 + \alpha_{1,7}x_1x_3^2 + \alpha_{1,6}x_2^2x_3 + \alpha_{1,19}x_3 + \alpha_{1,16}x_3^2 + \alpha_{1,3}x_3^3 - \alpha_{1,20}\\
&= F_1(x_1, x_2, x_3),
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{v}_5^{(2)} &= \sum_{j=1}^{19} \boldsymbol{W}_{45}^{(j,2)} \boldsymbol{v}_4^{(j)} = \alpha_{2,17}x_1 + \alpha_{2,14}x_1^2 + \alpha_{2,1}x_1^3\alpha_{2,11}x_1x_2 + \alpha_{2,4}x_1^2x_2 + \alpha_{2,5}x_1x_2^2 + \alpha_{2,10}x_1x_2x_3 + \alpha_{2,18}x_2 + \alpha_{2,15}x_2^2 + \alpha_{2,2}x_2^3\\
&\quad + \alpha_{2,13}x_2x_3 + \alpha_{2,9}x_2x_3^2 + \alpha_{2,8}x_2^2x_3 + \alpha_{2,12}x_1x_3 + \alpha_{2,7}x_1x_3^2 + \alpha_{2,6}x_2^2x_3 + \alpha_{2,19}x_3 + \alpha_{2,16}x_3^2 + \alpha_{2,3}x_3^3 - \alpha_{2,20} = F_2(x_1, x_2, x_3)\\
\boldsymbol{v}_5^{(3)} &= \sum_{j=1}^{19} \boldsymbol{W}_{45}^{(j,3)} \boldsymbol{v}_4^{(j)} = \alpha_{3,17}x_1 + \alpha_{3,14}x_1^2 + \alpha_{3,1}x_1^3 + \alpha_{3,11}x_1x_2 + \alpha_{3,4}x_1^2x_2 + \alpha_{3,5}x_1x_2^2 + \alpha_{3,10}x_1x_2x_3 + \alpha_{3,18}x_2 + \alpha_{3,15}x_2^2\\
&\quad + \alpha_{3,2}x_2^3 + \alpha_{3,13}x_2x_3 + \alpha_{3,9}x_2x_3^2 + \alpha_{3,8}x_2^2x_3 + \alpha_{3,12}x_1x_3 + \alpha_{3,7}x_1x_3^2 + \alpha_{3,6}x_2^2x_3 + \alpha_{3,19}x_3\\
&\quad + \alpha_{3,16}x_3^2 + \alpha_{3,3}x_3^3 - \alpha_{3,20} = F_3(x_1, x_2, x_3)
\end{aligned}$$

while the output produced by them and it is the real output of the neural network is given by the equations

$$\begin{aligned}
\boldsymbol{v}_5^{(1)} &\equiv o_1 = \tanh[\boldsymbol{v}_5^{(1)}] = \tanh[F_1(x_1, x_2, x_3)] = f(F_1),\\
\boldsymbol{v}_5^{(2)} &\equiv o_2 = \tanh[\boldsymbol{v}_5^{(2)}] = \tanh[F_2(x_1, x_2, x_3)] = f(F_2),\\
\boldsymbol{v}_5^{(3)} &\equiv o_3 = \tanh[\boldsymbol{v}_5^{(3)}] = \tanh[F_3(x_1, x_2, x_3)] = f(F_3),
\end{aligned}$$

where for the sake of brevity we define $f(x) = \tanh(x)$ and use the convention $F_i = F_i(x, y, z)$ $(i = 1, 2, 3)$.

6.2. Backward pass – estimation of the δ parameters

In this stage we use the vectors

$$\boldsymbol{\delta}_5 = \left[\boldsymbol{\delta}_5^{(1)} \quad \boldsymbol{\delta}_5^{(2)} \quad \boldsymbol{\delta}_5^{(3)}\right],$$

$$\boldsymbol{\delta}_4 = \left[\boldsymbol{\delta}_4^{(1)} \quad \boldsymbol{\delta}_4^{(2)} \quad \boldsymbol{\delta}_4^{(3)} \quad \cdots \quad \cdots \quad \cdots \quad \boldsymbol{\delta}_4^{(17)} \quad \boldsymbol{\delta}_4^{(18)} \quad \boldsymbol{\delta}_4^{(19)}\right],$$

$$\boldsymbol{\delta}_3 = \left[\boldsymbol{\delta}_3^{(1)} \quad \boldsymbol{\delta}_3^{(2)} \quad \boldsymbol{\delta}_3^{(3)} \quad \boldsymbol{\delta}_3^{(4)} \quad \boldsymbol{\delta}_3^{(5)} \quad \boldsymbol{\delta}_3^{(6)} \quad \boldsymbol{\delta}_3^{(7)} \quad \boldsymbol{\delta}_3^{(8)} \quad \boldsymbol{\delta}_3^{(9)}\right],$$

$$\boldsymbol{\delta}_2 = \left[\boldsymbol{\delta}_2^{(1)} \quad \boldsymbol{\delta}_2^{(2)} \quad \boldsymbol{\delta}_2^{(3)}\right]$$

to denote the δ values of the fifth, the fourth, the third and the second layer, respectively.

Since the desired output of the neural network are the values $d_1 = d_2 = d_3 = 0$ it can be easily seen that

$$\delta_5^{(1)} = (d_1 - o_1)\mathbf{f}_5^{(1)'}(\mathbf{v}_5^{(1)}) = -o_1 f'(F_1) = -f(F_1)\{1 - f^2(F_1)\},$$
$$\delta_5^{(2)} = (d_2 - o_2)\mathbf{f}_5^{(2)'}(\mathbf{v}_5^{(2)}) = -o_2 f'(F_2) = -f(F_2)\{1 - f^2(F_2)\},$$
$$\delta_5^{(3)} = (d_3 - o_3)\mathbf{f}_5^{(1)'}(\mathbf{v}_5^{(3)}) = -o_3 f'(F_1) = -f(F_3)\{1 - f^2(F_3)\}.$$

Having estimated the components of the $\boldsymbol{\delta}_5$ vector, we can now estimate the components of the $\boldsymbol{\delta}_4$ vector (namely the delta values of the fourth layer) by using the equation

$$\boldsymbol{\delta}_4^{(k)x_i} = \left(\sum_{j=1}^{3} \boldsymbol{W}_{45}^{(k,j)} \boldsymbol{\delta}_5^{(j)}\right) \xi_4^{(k)x_i}, \quad k = 1, \ldots, 19, \quad \text{where } \xi_4^{(k)x_i} = \frac{\partial \mathbf{f}_4^{(k)}}{\partial \mathbf{v}_4^{(k)}} \frac{\partial \mathbf{v}_4^{(k)}}{\partial x_i}.$$

Since these neurons produce nonlinear terms we have generally to estimate more than one delta values for each one them, and for each independent variable participating to the nonlinear term they produce.

In the above equation, there are many function derivatives with respect to x, y and z; these derivatives are estimated as

- $\xi_4^{(k)x_i} = 1$ for $(k,i) = \{(1,1),(8,2),(17,3)\}$ with $\xi_4^{(k)x_j} = 0$ for $j \neq i$.
- $\xi_4^{(k)x_i} = 2x_i$ for $(k,i) = \{(2,1),(9,2),(18,3)\}$ with $\xi_4^{(k)x_j} = 0$ for $j \neq i$.
- $\xi_4^{(k)x_i} = 3x_i^2$ for $(k,i) = \{(3,1),(10,2),(19,3)\}$ with $\xi_4^{(k)x_j} = 0$ for $j \neq i$.
- $\xi_4^{(7)x_i} = x_j x_\ell$ for $(i,j,\ell) = \{(1,2,3),(2,1,3),(3,1,2)\}$.
- $\xi_4^{(k)x_i} = x_j$ for $(k,i,j) = \{(4,1,2),(4,2,1),(11,2,3),(11,3,2),(14,1,3),(14,3,1)\}$.
- $\xi_4^{(k)x_i} = 0$ for $(k,i) = \{(4,3),(11,1),(14,2)\}$.
- $\xi_4^{(k)x_i} = x_j^2$ for $(k,i,j) = \{(5,2,1),(6,1,2),(12,2,3),(13,3,2),(15,1,3),(16,3,1)\}$.
- $\xi_4^{(k)x_i} = 2x_i x_j$ for $(k,i,j) = \{(5,1,2),(6,2,1),(12,3,2),(13,2,3),(15,3,1),(16,1,3)\}$.
- $\xi_4^{(k)x_i} = 0$ for $(k,i) = \{(5,3),(6,3),(12,1),(13,1),(15,2),(16,2)\}$.

Based on the above expressions,we get the results

$$\boldsymbol{\delta}_4^{(1)x_1} = \sum_{j=1}^{3} \alpha_{j,17} \boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(1)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(1)x_3} = 0, \quad \boldsymbol{\delta}_4^{(2)x_1} = 2x_1 \sum_{j=1}^{3} \alpha_{j,14} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(2)x_2} = 0, \quad \boldsymbol{\delta}_4^{(2)x_3} = 0,$$

$$\boldsymbol{\delta}_4^{(3)x_1} = 3x_1^2 \sum_{j=1}^{3} \alpha_{j,1} \boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(3)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(3)x_3} = 0, \quad \boldsymbol{\delta}_4^{(4)x_1} = x_2 \sum_{j=1}^{3} \alpha_{j,11} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(4)x_2} = x_1 \sum_{j=1}^{3} \alpha_{j,11} \boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(4)x_3} = 0,$$

$$\boldsymbol{\delta}_4^{(5)x_1} = 2x_1 x_2 \sum_{j=1}^{3} \alpha_{j,4} \boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(5)x_2} = x_1^2 \sum_{j=1}^{3} \alpha_{j,4} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(5)x_3} = 0, \quad \boldsymbol{\delta}_4^{(6)x_1} = x_2^2 \sum_{j=1}^{3} \alpha_{j,5} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(6)x_2} = 2x_1 x_2 \sum_{j=1}^{3} \alpha_{j,5} \boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(6)x_3} = 0,$$

$$\boldsymbol{\delta}_4^{(7)x_1} = x_2 x_3 \sum_{j=1}^{3} \alpha_{j,10} \boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(7)x_2} = x_1 x_3 \sum_{j=1}^{3} \alpha_{j,10} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(7)x_3} = x_1x_2\sum_{j=1}^{3}\alpha_{j,10}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(8)x_1} = 0,$$

$$\boldsymbol{\delta}_4^{(8)x_2} = \sum_{j=1}^{3}\alpha_{j,18}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(8)x_3} = 0,$$

$$\boldsymbol{\delta}_4^{(9)x_1} = 0, \quad \boldsymbol{\delta}_4^{(9)x_2} = 2x_2\sum_{j=1}^{3}\alpha_{j,15}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(9)x_3} = 0, \quad \boldsymbol{\delta}_4^{(10)x_1} = 0,$$

$$\boldsymbol{\delta}_4^{(10)x_2} = 3x_2^2\sum_{j=1}^{3}\alpha_{j,2}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(10)x_3} = 0,$$

$$\boldsymbol{\delta}_4^{(11)x_1} = 0, \quad \boldsymbol{\delta}_4^{(11)x_2} = x_3\sum_{j=1}^{3}\alpha_{j,13}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(11)x_3} = x_2\sum_{j=1}^{3}\alpha_{j,13}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(12)x_1} = 0,$$

$$\boldsymbol{\delta}_4^{(12)x_2} = x_3^2\sum_{j=1}^{3}\alpha_{j,9}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(12)x_3} = 2x_2x_3\sum_{j=1}^{3}\alpha_{j,9}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(13)x_1} = 0, \quad \boldsymbol{\delta}_4^{(13)x_2} = 2x_2x_3\sum_{j=1}^{3}\alpha_{j,8}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(14)x_1} = x_3\sum_{j=1}^{3}\alpha_{j,12}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(14)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(15)x_1} = x_3^2\sum_{j=1}^{3}\alpha_{j,7}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(15)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(16)x_1} = 2x_1x_3\sum_{j=1}^{3}\alpha_{j,6}\boldsymbol{\delta}_5^{(j)}, \quad \boldsymbol{\delta}_4^{(16)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(17)x_1} = 0, \quad \boldsymbol{\delta}_4^{(17)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(18)x_1} = 0, \quad \boldsymbol{\delta}_4^{(18)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(19)x_1} = 0, \quad \boldsymbol{\delta}_4^{(197)x_2} = 0,$$

$$\boldsymbol{\delta}_4^{(13)x_3} = x_2^2\sum_{j=1}^{3}\alpha_{j,8}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(14)x_3} = x_1\sum_{j=1}^{3}\alpha_{j,12}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(15)x_3} = 2x_1x_3\sum_{j=1}^{3}\alpha_{j,7}\boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(16)x_3} = x_1^2 \sum_{j=1}^{3} \alpha_{j,6} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(17)x_3} = \sum_{j=1}^{3} \alpha_{j,19} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(18)x_3} = 2x_3 \sum_{j=1}^{3} \alpha_{j,16} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_4^{(19)x_3} = 3x_3^2 \sum_{j=1}^{3} \alpha_{j,3} \boldsymbol{\delta}_5^{(j)}.$$

In the next step we use these values to estimate the parameters $\boldsymbol{\delta}_3^{(i)}$ $(i = 1, 2, 3, \ldots, 9)$ via the expression

$$\boldsymbol{\delta}_3^{(i)} = \begin{cases} \sum_{j=1}^{19} \boldsymbol{W}_{34}^{(i,j)} \boldsymbol{\delta}_4^{(i)x_1}, & i = 1, 2, 3, \\ \sum_{j=1}^{19} \boldsymbol{W}_{34}^{(i,j)} \boldsymbol{\delta}_4^{(i)x_2}, & i = 4, 5, 6, \\ \sum_{j=1}^{19} \boldsymbol{W}_{34}^{(i,j)} \boldsymbol{\delta}_4^{(i)x_3}, & i = 7, 8, 9. \end{cases}$$

The results are the following

$$\begin{aligned} \boldsymbol{\delta}_3^{(1)} &= \boldsymbol{\delta}_4^{(1)x_1} + \boldsymbol{\delta}_4^{(4)x_1} + \boldsymbol{\delta}_4^{(6)x_1} + \boldsymbol{\delta}_4^{(7)x_1} + \boldsymbol{\delta}_4^{(14)x_1} + \boldsymbol{\delta}_4^{(15)x_1} \\ &= \sum_{j=1}^{3} \alpha_{j,17} \boldsymbol{\delta}_5^{(j)} + x_2 \sum_{j=1}^{3} \alpha_{j,11} \boldsymbol{\delta}_5^{(j)} + x_2 \sum_{j=1}^{3} \alpha_{j,11} \boldsymbol{\delta}_5^{(j)} + x_2^2 \sum_{j=1}^{3} \alpha_{j,5} \boldsymbol{\delta}_5^{(j)} + x_2 x_3 \sum_{j=1}^{3} \alpha_{j,10} \boldsymbol{\delta}_5^{(j)} + x_3 \sum_{j=1}^{3} \alpha_{j,12} \boldsymbol{\delta}_5^{(j)} + x_3^2 \sum_{j=1}^{3} \alpha_{j,7} \boldsymbol{\delta}_5^{(j)}, \end{aligned}$$

$$\boldsymbol{\delta}_3^{(2)} = \boldsymbol{\delta}_4^{(2)x_1} + \boldsymbol{\delta}_4^{(5)x_1} + \boldsymbol{\delta}_4^{(16)x_1} = 2x_1 \sum_{j=1}^{3} \alpha_{j,14} \boldsymbol{\delta}_5^{(j)} + 2x_1 x_2 \sum_{j=1}^{3} \alpha_{j,4} \boldsymbol{\delta}_5^{(j)} + 2x_1 x_3 \sum_{j=1}^{3} \alpha_{j,6} \boldsymbol{\delta}_5^{(j)}$$

$$\boldsymbol{\delta}_3^{(3)} = \boldsymbol{\delta}_4^{(3)x_1} = 3x_1^2 \sum_{j=1}^{3} \alpha_{j,1} \boldsymbol{\delta}_5^{(j)}$$

$$\begin{aligned} \boldsymbol{\delta}_3^{(4)} &= \boldsymbol{\delta}_4^{(4)x_2} + \boldsymbol{\delta}_4^{(5)x_2} + \boldsymbol{\delta}_4^{(7)x_2} + \boldsymbol{\delta}_4^{(8)x_2} + \boldsymbol{\delta}_4^{(11)x_2} + \boldsymbol{\delta}_4^{(12)x_2} \\ &= x_1 \sum_{j=1}^{3} \alpha_{j,11} \boldsymbol{\delta}_5^{(j)} + x_1^2 \sum_{j=1}^{3} \alpha_{j,4} \boldsymbol{\delta}_5^{(j)} + x_1 x_3 \sum_{j=1}^{3} \alpha_{j,10} \boldsymbol{\delta}_5^{(j)} + \sum_{j=1}^{3} \alpha_{j,18} \boldsymbol{\delta}_5^{(j)} + x_3 \sum_{j=1}^{3} \alpha_{j,13} \boldsymbol{\delta}_5^{(j)} + x_3^2 \sum_{j=1}^{3} \alpha_{j,9} \boldsymbol{\delta}_5^{(j)}, \end{aligned}$$

$$\boldsymbol{\delta}_3^{(5)} = \boldsymbol{\delta}_4^{(6)x_2} + \boldsymbol{\delta}_4^{(9)x_2} + \boldsymbol{\delta}_4^{(13)x_2} = 2x_1 x_2 \sum_{j=1}^{3} \alpha_{j,5} \boldsymbol{\delta}_5^{(j)} + 2x_2 \sum_{j=1}^{3} \alpha_{j,15} \boldsymbol{\delta}_5^{(j)} + 2x_2 x_3 \sum_{j=1}^{3} \alpha_{j,8} \boldsymbol{\delta}_5^{(j)},$$

$$\boldsymbol{\delta}_3^{(6)} = \boldsymbol{\delta}_4^{(10)x_2} = 3x_2^2 \sum_{j=1}^{3} \alpha_{j,2} \boldsymbol{\delta}_5^{(j)}$$

$$\begin{aligned} \boldsymbol{\delta}_3^{(7)} &= \boldsymbol{\delta}_4^{(7)x_3} + \boldsymbol{\delta}_4^{(11)x_3} + \boldsymbol{\delta}_4^{(13)x_3} + \boldsymbol{\delta}_4^{(14)x_3} + \boldsymbol{\delta}_4^{(16)x_3} + \boldsymbol{\delta}_4^{(17)x_3} \\ &= x_1 x_2 \sum_{j=1}^{3} \alpha_{j,10} \boldsymbol{\delta}_5^{(j)} + x_2 \sum_{j=1}^{3} \alpha_{j,13} \boldsymbol{\delta}_5^{(j)} + x_2^2 \sum_{j=1}^{3} \alpha_{j,8} \boldsymbol{\delta}_5^{(j)} + x_1 \sum_{j=1}^{3} \alpha_{j,12} \boldsymbol{\delta}_5^{(j)} + x_1^2 \sum_{j=1}^{3} \alpha_{j,6} \boldsymbol{\delta}_5^{(j)} + \sum_{j=1}^{3} \alpha_{j,19} \boldsymbol{\delta}_5^{(j)}, \end{aligned}$$

$$\boldsymbol{\delta}_3^{(8)} = \boldsymbol{\delta}_4^{(12)x_3} + \boldsymbol{\delta}_4^{(15)x_3} + \boldsymbol{\delta}_4^{(18)x_3} = 2x_2 x_3 \sum_{j=1}^{3} \alpha_{j,9} \boldsymbol{\delta}_5^{(j)} + 2x_1 x_3 \sum_{j=1}^{3} \alpha_{j,7} \boldsymbol{\delta}_5^{(j)} + 2x_3 \sum_{j=1}^{3} \alpha_{j,16} \boldsymbol{\delta}_5^{(j)}$$

$$\boldsymbol{\delta}_3^{(9)} = \boldsymbol{\delta}_4^{(19)x_3} = 3x_3^2 \sum_{j=1}^{3} \alpha_{j,3} \boldsymbol{\delta}_5^{(j)}.$$

Finally, the values of $\delta_2^{(i)}$ $(i = 1, 2, 3)$ are estimated via the equation

$$\delta_2^{(i)} = \sum_{j=1}^{9} W_{23}^{(i,j)} \delta_3^{(i)} \quad (i = 1, 2, 3)$$

and the results are

$$\delta_2^{(1)} = \delta_3^{(1)} + \delta_3^{(2)} + \delta_3^{(3)} = \left(3x_1^2 \sum_{j=1}^{3} \alpha_{j,1} + 2x_1 x_2 \sum_{j=1}^{3} \alpha_{j,4} + x_2^2 \sum_{j=1}^{3} \alpha_{j,5} + 2x_1 x_3 \sum_{j=1}^{3} \alpha_{j,6} + x_3^2 \sum_{j=1}^{3} \alpha_{j,7} + x_2 x_3 \sum_{j=1}^{3} \alpha_{j,10} + x_2 \sum_{j=1}^{3} \alpha_{j,11} + x_3 \sum_{j=1}^{3} \alpha_{j,12} + 2x_1 \sum_{j=1}^{3} \alpha_{j,14} + \sum_{j=1}^{3} \alpha_{j,17} \right) \delta_5^{(j)} = \sum_{j=1}^{3} \left(\frac{\partial F_j(x_1, x_2, x_3)}{\partial x_1} \right) \delta_5^{(j)},$$

$$\delta_2^{(2)} = \delta_3^{(4)} + \delta_3^{(5)} + \delta_3^{(6)} = \left(3x_2^2 \sum_{j=1}^{3} \alpha_{j,2} + x_1^2 \sum_{j=1}^{3} \alpha_{j,4} + 2x_1 x_2 \sum_{j=1}^{3} \alpha_{j,5} + 2x_2 x_3 \sum_{j=1}^{3} \alpha_{j,8} + x_3^2 \sum_{j=1}^{3} \alpha_{j,9} + x_1 \sum_{j=1}^{3} \alpha_{j,11} + x_3 \sum_{j=1}^{3} \alpha_{j,13} + 2x_2 \sum_{j=1}^{3} \alpha_{j,15} + \sum_{j=1}^{3} \alpha_{j,18} + x_1 x_3 \sum_{j=1}^{3} \alpha_{j,10} \right) \delta_5^{(j)} = \sum_{j=1}^{3} \left(\frac{\partial F_j(x_1, x_2, x_3)}{\partial x_2} \right) \delta_5^{(j)},$$

$$\delta_2^{(3)} = \delta_3^{(7)} + \delta_3^{(8)} + \delta_3^{(9)} = \left(3x_3^2 \sum_{j=1}^{3} \alpha_{j,3} + x_1^2 \sum_{j=1}^{3} \alpha_{j,6} + 2x_1 x_3 \sum_{j=1}^{3} \alpha_{j,7} + x_2^2 \sum_{j=1}^{3} \alpha_{j,8} + 2x_2 x_3 \sum_{j=1}^{3} \alpha_{j,9} + x_1 x_2 \sum_{j=1}^{3} \alpha_{j,10} + x_1 \sum_{j=1}^{3} \alpha_{j,12} + x_2 \sum_{j=1}^{3} \alpha_{j,13} + 2x_3 \sum_{j=1}^{3} \alpha_{j,16} + \sum_{j=1}^{3} \alpha_{j,19} \right) \delta_5^{(j)} = \sum_{j=1}^{3} \left(\frac{\partial F_j(x_1, x_2, x_3)}{\partial x_3} \right) \delta_5^{(j)}.$$

6.3. Update of the synaptic weights

Proof. According to the back propagation algorithm, the update of the two variable weights $W_{12}^{(i)} = x_i$ $(i = 1, 2, 3)$ is performed as

$$\{W_{12}^{(i)}\}^{(k+1)} = \{W_{12}^{(i)}\}^{(k)} + \beta \delta_2^{(i)} = \{W_{12}^{(i)}\}^{(k)} + \beta \sum_{j=1}^{3} \frac{\partial F_j}{\partial x_i} \delta_5^{(j)}$$

or equivalently, as

$$\{W_{12}^{(i)}\}^{(k+1)} = \{W_{12}^{(i)}\}^{(k)} - \beta \sum_{j=1}^{3} f(F_j)(1 - f^2(F_j)) \frac{\partial F_j}{\partial W_{12}^{(i)}},$$

where we used the expressions for the $\delta_2^{(i)}$ and $\delta_5^{(j)}$ parameters.

From the definition of the mean square error of the back propagation algorithm

$$E = \frac{1}{2} \sum_{j=1}^{3} (d_j - o_j)^2 = \frac{1}{2} \sum_{j=1}^{3} o_j^2 = \frac{1}{2} \sum_{j=1}^{3} f^2(F_j)$$

it can be easily seen that

$$\frac{\partial E}{\partial W_{12}^{(i)}} = \sum_{j=1}^{3} f(F_j) f'(F_j) \frac{\partial F_j}{\partial W_{12}^{(i)}} = \sum_{j=1}^{3} f(F_j)(1 - f^2(F_j)) \frac{\partial F_j}{\partial W_{12}^{(i)}}$$

and therefore the weight update equation gets the form

$$\{W_{12}^{(i)}\}^{(k+1)} = \{W_{12}^{(i)}\}^{(k)} - \beta \frac{\partial E}{\partial W_{12}^{(i)}}$$

that completes the proof. □

7. Experimental results and comparison with other methods

To test the validity of the neural model and evaluate its precision and performance, typical 3×3 nonlinear systems were constructed and solved using the network described above. The results and the analysis of the associated simulations, are presented below.

Example 1. The first system we present has eight real roots and it consists of the following equations:

$$F_1(x_1,x_2,x_3) = x_1^3 + x_2^3 + x_3^3 - x_1^2x_2 + x_1x_2^2 - x_2^2x_3 + x_1x_2x_3 - x_1 - x_2 - x_3 = 0,$$
$$F_2(x_1,x_2,x_3) = 0.5x_1^3 + 0.5x_2^3 + 0.5x_3^3 - x_1^2x_2 + x_1x_2^2 - x_2^2x_3 + x_1x_2x_3 - 0.5x_1^2 - 0.5x_2^2 - 0.5x_3^2 - 0.5x_1 - 0.5x_2 - 0.5x_3 + 1.5 = 0,$$
$$F_3(x_1,x_2,x_3) = x_1^3 + x_2^3 + x_3^3 + x_1^2x_2 - x_1x_2^2 + x_2^2x_3 - x_1x_2x_3 + x_1^2 + x_2^2 + x_3^2 - x_1 - x_2 - x_3 - 3 = 0$$

The solution of this system by means of symbolic packages such as Mathematica, reveals the existence of 8 discrete roots with values

1. $(x_1,x_2,x_3) = (-1,-1,-1)$.
2. $(x_1,x_2,x_3) = (-1,-1,+1)$.
3. $(x_1,x_2,x_3) = (-1,+1,-1)$.
4. $(x_1,x_2,x_3) = (+1,-1,+1)$.
5. $(x_1,x_2,x_3) = (+1,+1,-1)$.
6. $(x_1,x_2,x_3) = (+1,+1,+1)$.
7. $(x_1,x_2,x_3) = (-1.224744,-2E-8,1.224744)$.
8. $(x_1,x_2,x_3) = (1.224744,2E-8,-1.224744)$.

To solve the above system, the neural model run many times with different initial conditions in the intervals $-3 \leqslant x_1 \leqslant 2.5$, $-3 \leqslant x_2 \leqslant 1.4$, and $-3 \leqslant x_3 \leqslant 0.5$ with a variation step $\Delta x_1 = \Delta x_2 = \Delta x_3 = 0.1$. Therefore, the model run 207375 times to identify the basin of attraction of each root and to calculate the percentage of the initial conditions associated with each root.

The operation of the neural network was simulated in MATLAB, and the components x_1, x_2 and x_3 of each root were estimated by using recurrently the last equation of the previous section. The learning rate for the back propagation algorithm was $\beta = 0.005$. Since the speed of convergence was too fast, the maximum number of iterations was set to $N = 10000$; therefore after 10000 iterations, the system was characterized as nonconvergent if it was unable to reach a root, with the process to move to the next point of initial conditions. For each such point, the network either reached one of the eight system roots (it is important to say that the network was able to identify all roots) or it was unable to estimate one of those roots. The basin of attraction for the eight roots as well as the number of points of that basin are depicted graphically in Figs. 3–6. Since these figures are three dimensional, in the above plots the projections $x_1 - x_2, x_1 - x_3$ and $x_2 - x_3$ are shown. Even though MATLAB supports the construction of 3D plots, these 2D graphs are more suitable for preview and they can be very easily extended in three dimensions.

The basin of attraction of a root is defined as the fraction of the parameter space points (x_1,x_2,x_3) – namely, the values of the initial conditions – for which the neural network converged to that root. The ratio of the number of these basin points to the total number of the tested initial condition points multiplied by 100, gives the percentage defined above. For each one of the estimated roots, the best simulation run with respect to the minimal iteration number was identified.

The experimental results associated with this system are shown in Tables 2 and 3. Table 2 contains for each one of the identified roots, the number of systems converged to that root, the average number of iterations and the mean square error as well as the standard deviation of those values, and the average value of the x_1, x_2 and x_3 components for each root. On the other hand, the data of Table 3 are associated with the best run for each root, with the criterion for the best system to be the minimum number of iterations. The values shown for each best run, are the initial conditions (x_1^0,x_2^0,x_3^0), the number of iterations, as well as the values of the functions $F_1(x_1,x_2,x_3)$, $F_2(x_1,x_2,x_3)$ and $F_3(x_1,x_2,x_3)$. For all best runs the value of MSE were equal to $E-16$.

ROOT	No of Systems	PERCENTAGE
1	15	0,007%
2	9348	4,508%
3	11487	5,539%
4	11946	5,761%
5	8475	4,087%
6	18	0,009%
7	7486	3,610%
8	8057	3,885%
NO ROOT	150543	72,595%
TOTAL	**207375**	

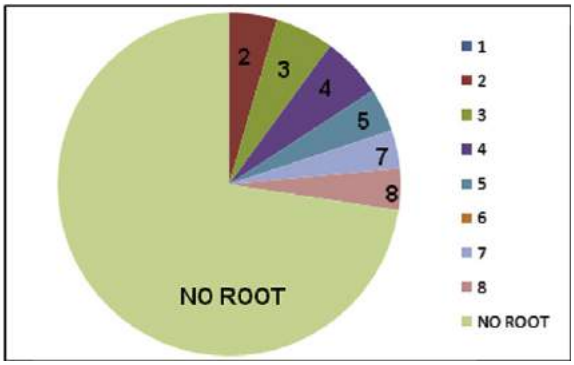

Fig. 3. Number of basin points for each one of the roots of System 1 and the nonconverging behavior, and the associated pie chart.

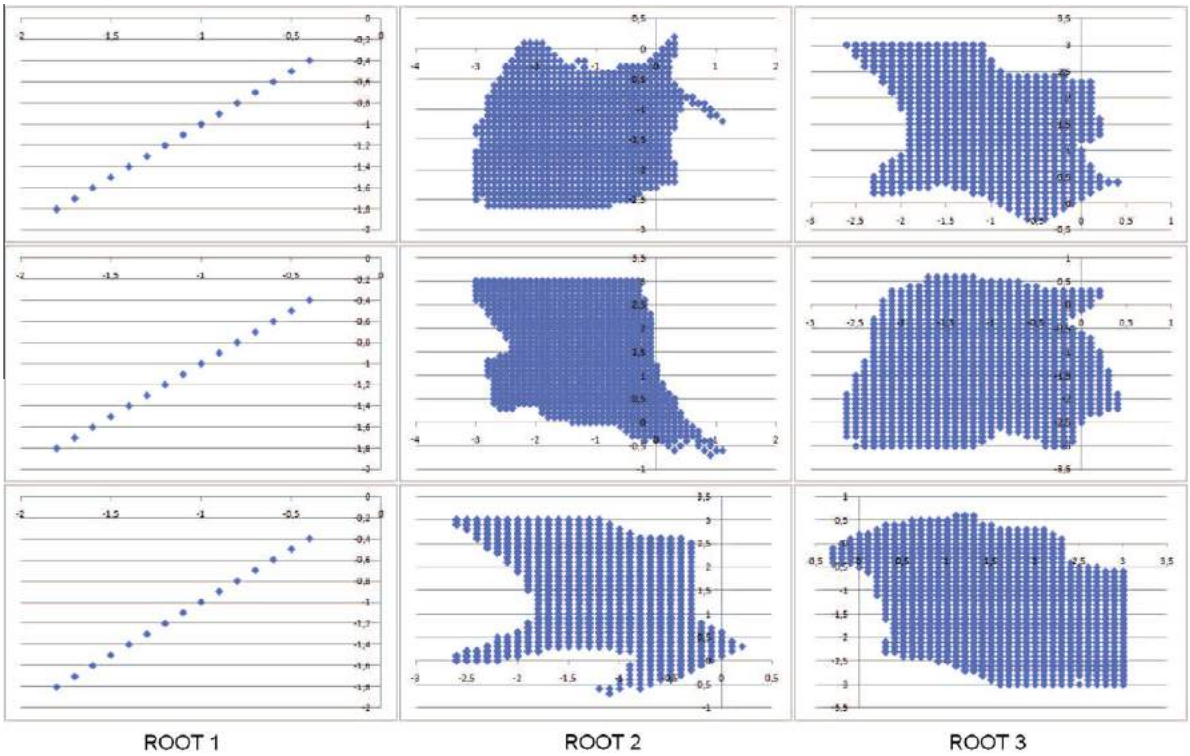

Fig. 4. The basin of attraction for the roots 1,2 and 3 of the System 1 in the intervals defined above. For each root, the projections $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$ are shown in the top, middle, and bottom figure, respectively.

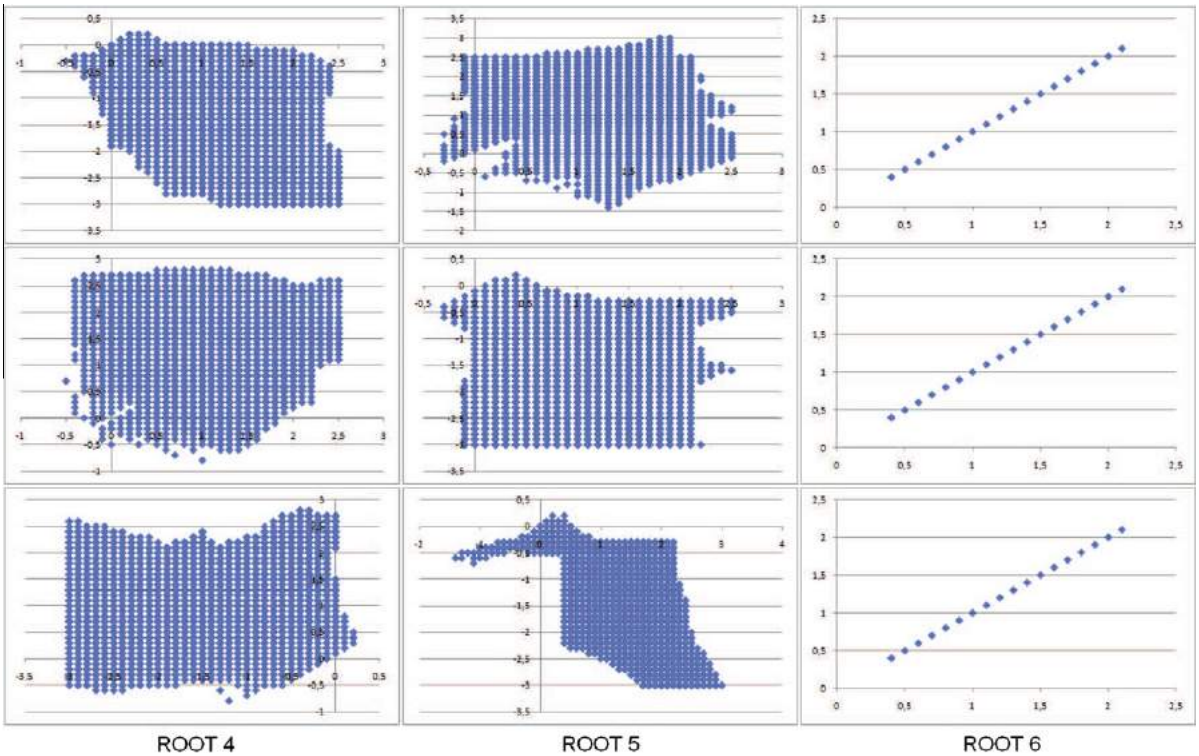

Fig. 5. The basin of attraction for the roots 4,5 and 6 of the System 1 in the intervals defined above. For each root, the projections $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$ are shown in the top, middle, and bottom figure, respectively.

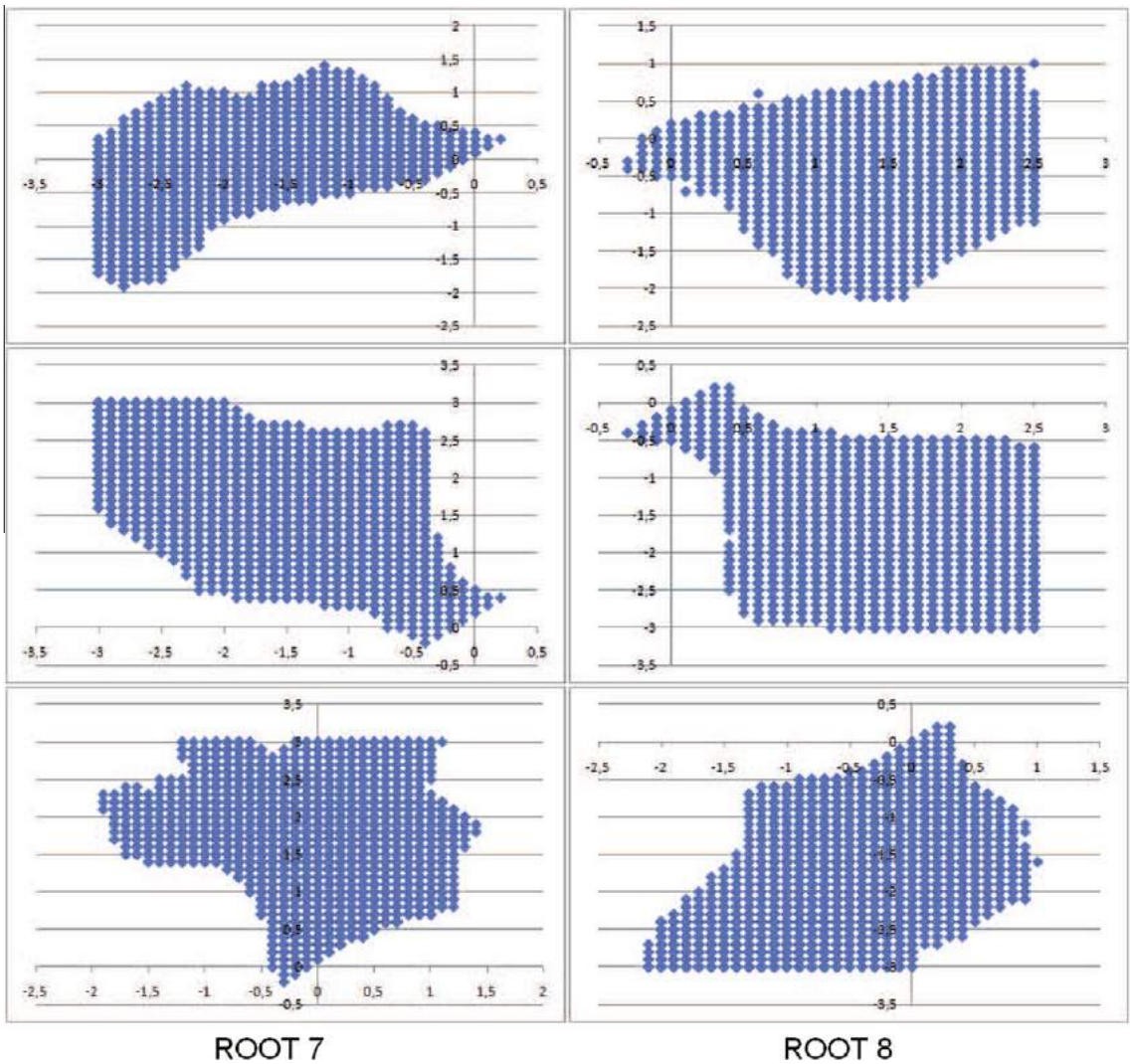

Fig. 6. The basin of attraction for the roots 7 and 8 of the System 1 in the intervals defined above. For each root, the projections $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$ are shown in the top, middle, and bottom figure, respectively.

The variation of the iteration number with the learning rate for the first example system is shown in Fig. 7. Due to the large variation of the iteration number for the eight roots of the system, the vertical axis is not in a linear scale as the horizontal axis but in a logarithmic one. From this figure it is clear that in most cases the number of iterations decreases, as a learning rate increases, a fact, that it is of course expected. However, it has to be mentioned that for greater values of learning rate, the neural network could not be able to converge to some root, and therefore, the allowable learning rate values are much smaller than the typical value of 0.5–0.7 associated with the classical back-propagation value. This relatively small values for the learning rate, leads to relatively large number of iterations, compared with the iteration number associated with other methods as it is pointed out in the concluding section.

Example 2. The second system we present has four real roots and it consists of the following equations:

$$F_1(x_1,x_2,x_3) = x_1^3 + x_2^3 + x_3^3 - 3x_1^2 - 2x_2^2 - 2x_3^2 + 2x_1 + x_2 + x_3 = 0,$$
$$F_2(x_1,x_2,x_3) = x_1^3 + x_2^3 + x_3^3 - x_1^2 - 5x_2^2 - x_3^2 + 8x_2 - 4 = 0,$$
$$F_3(x_1,x_2,x_3) = x_1^3 + x_2^3 + x_3^3 - 4x_1^2 - 4x_2^2 - 5x_3^2 + 4x_1 + 5x_2 + 8x_3 - 6 = 0.$$

The solution of this system with traditional methods reveals the existence of four roots with values

1. $(x_1,x_2,x_3) = (1.428042, 0.384132, 1.383866)$.
2. $(x_1,x_2,x_3) = (1.107007, 1.008547, 0.568966)$.

Table 2
The experimental results for the eight roots of the first algebraic system.

Root no.	AVG iteration	STDEV iteration	AVG MSE	STDEV MSE
1	01451	04383.449	$9.33 \cdot 10^{-17}$	$2.58 \cdot 10^{-17}$
2	19804	17287.220	$2.66 \cdot 10^{-15}$	$1.24 \cdot 10^{-13}$
3	19688	17374.550	$3.67 \cdot 10^{-15}$	$1.53 \cdot 10^{-13}$
4	24297	16527.800	$3.81 \cdot 10^{-15}$	$1.29 \cdot 10^{-13}$
5	23075	15753.260	$3.71 \cdot 10^{-15}$	$1.20 \cdot 10^{-13}$
6	3193	10744.260	$8.89 \cdot 10^{-17}$	$3.23 \cdot 10^{-17}$
7	19162	16127.700	$3.97 \cdot 10^{-15}$	$1.52 \cdot 10^{-13}$
8	19121	16205.900	$3.50 \cdot 10^{-15}$	$1.56 \cdot 10^{-13}$

Root no.	AVG x_1	AVG x_2	AVG x_3
1	−1	−1	−1
2	−1	−1	+1
3	−1	+1	−1
4	+1	−1	+1
5	+1	+1	−1
6	+1	+1	+1
7	−1.224744	$4 \cdot 10^{-9}$	1.224744
8	1.224744	$8 \cdot 10^{-9}$	−1.224744

Table 3
The results for the best run for each one of the eight roots of the first algebraic system.

Root	x_1^0	x_2^0	x_3^0	Iterations	F_1	F_2	F_3
1	−1.1	−1.1	−1.1	0138	-10^{-8}	-10^{-8}	0
2	−1.0	−0.5	+1.1	6224	-10^{-8}	-10^{-8}	0
3	−1.1	+0.7	−1.3	6188	-10^{-8}	-10^{-8}	0
4	+0.5	−1.4	+0.7	7357	$+10^{-8}$	-10^{-8}	0
5	+1.0	+0.7	−1.7	7045	-10^{-8}	$+10^{-8}$	0
6	+1.1	+1.1	+1.1	0054	0	$+10^{-8}$	0
7	−1.2	+0.1	+1.7	5434	$+10^{-8}$	-10^{-8}	0
8	+0.6	−0.3	−1.2	5615	$+10^{-8}$	-10^{-8}	0

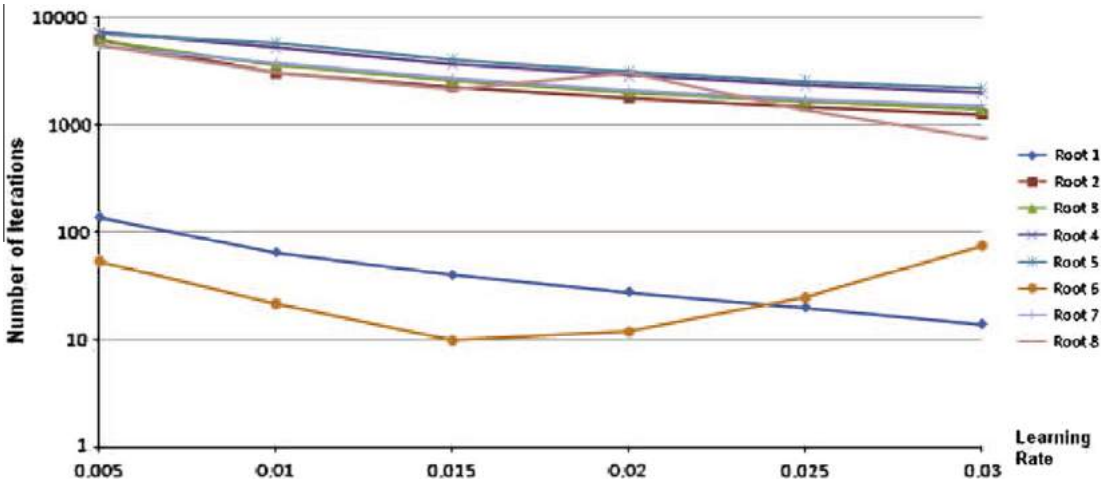

Fig. 7. The variation of the iteration number as a function of the learning rate for the first example system.

3. $(x_1, x_2, x_3) = (-0.020060, 0.856390, 1.143860)$.
4. $(x_1, x_2, x_3) = (0, 1, 1)$.

ROOT	No of Systems	PERCENTAGE
1	16	0,054%
2	3847	12,913%
3	1204	4,041%
4	3294	11,057%
NO ROOT	21430	71,934%
TOTAL	**29791**	

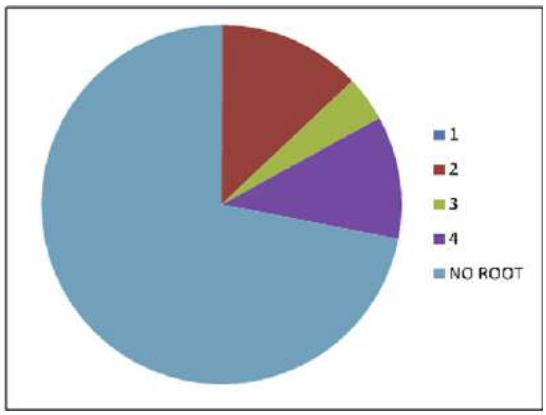

Fig. 8. Number of basin points for each one of the roots of System 2 and the nonconverging behavior, and the associated pie chart.

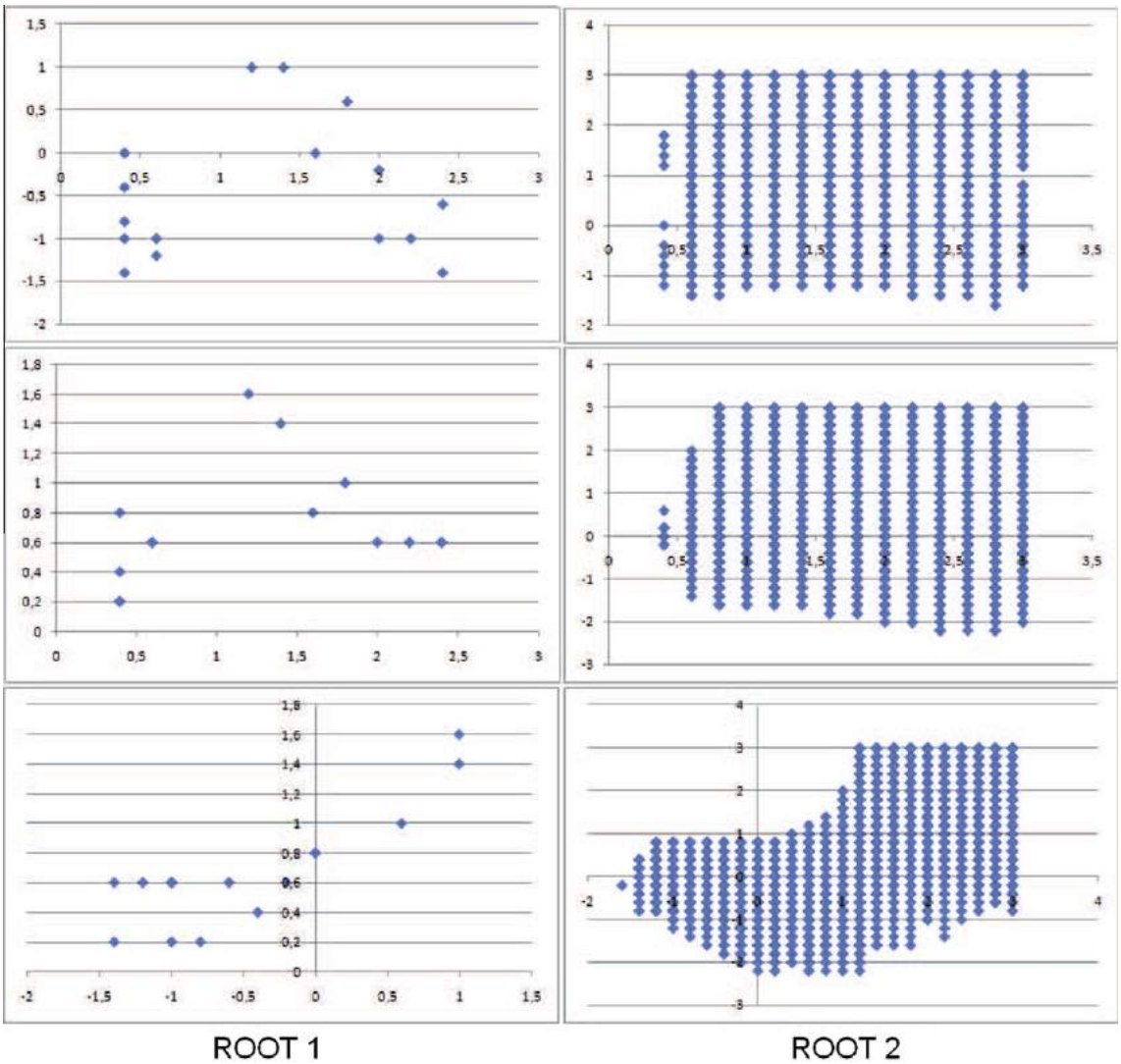

Fig. 9. The basin of attraction for the roots 1 and 2 of the System 2 in the intervals defined above. For each root, the projections $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$ are shown in the top, middle, and bottom figure, respectively.

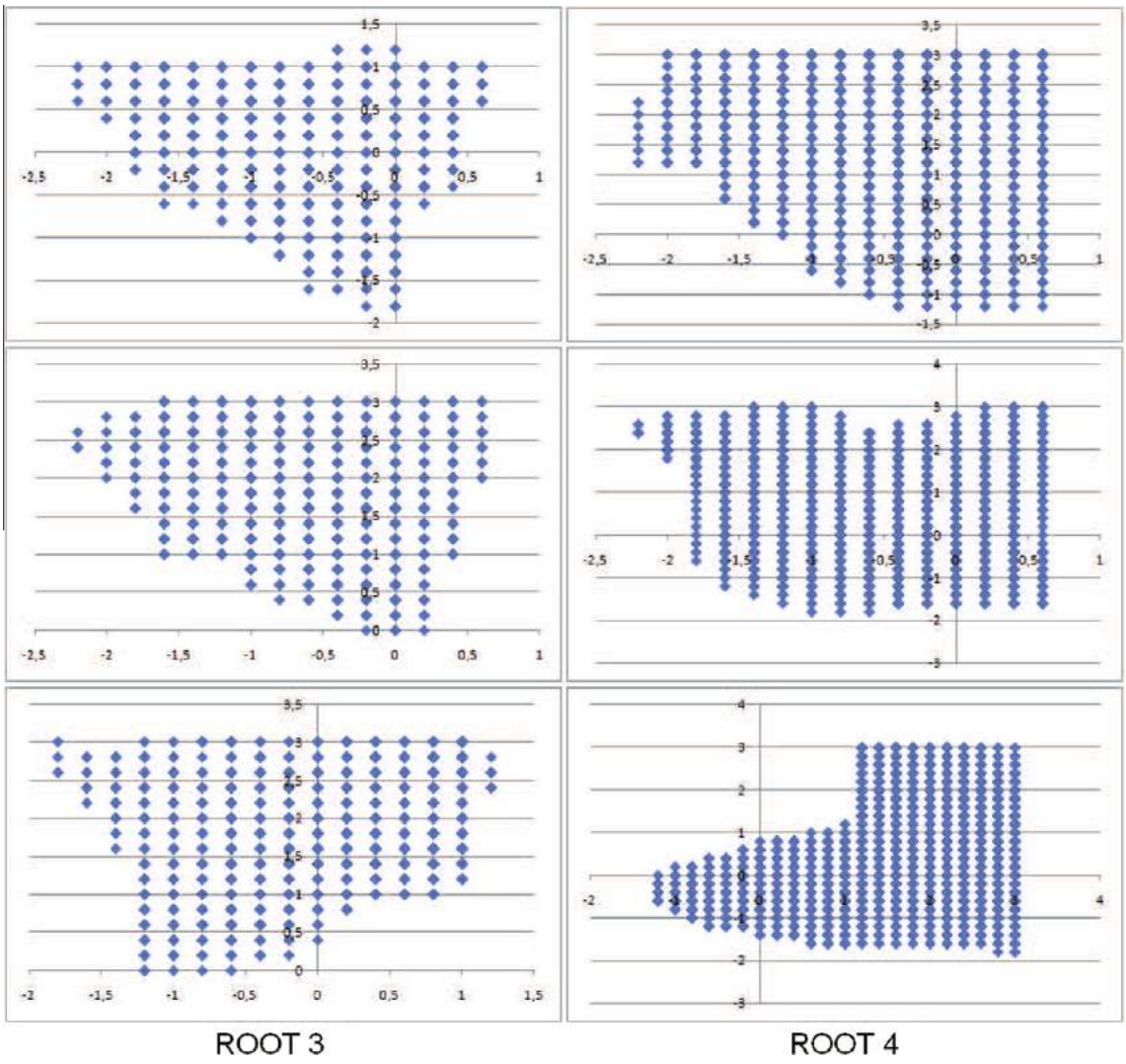

Fig. 10. The basin of attraction for the roots 3 and 4 of the System 2 in the intervals defined above. For each root, the projections $x_1 - x_2$, $x_1 - x_3$, and $x_2 - x_3$ are shown in the top, middle, and bottom figure, respectively.

Table 4
The experimental results for the four roots of the second algebraic system.

Root no.	AVG iteration	STDEV iteration	AVG MSE	STDEV MSE	AVG x_1	AVG x_2	AVG x_3
1	05755	16561.010	10^{-16}	$3.82 \cdot 10^{-32}$	1.428042	0.384133	1.383867
2	06778	15292.079	10^{-16}	$5.71 \cdot 10^{-30}$	1.070070	1.008547	0.568966
3	09366	15683.740	$1.03 \cdot 10^{-16}$	$9.22 \cdot 10^{-17}$	−0.020060	0.856387	1.143864
4	15433	20414.910	$1.01 \cdot 10^{-16}$	$4.45 \cdot 10^{-17}$	$9.52 \cdot 10^{-9}$	1	1

As in the previous case, the neural model solved the system with a lot of different initial conditions in order to identify all the four roots and estimate the basin of attraction for each one of them. More specifically, the parameters x_1, x_2 and x_3 were varied in the interval $-3 \leqslant x_i \leqslant 3$ with a variation step $\Delta x_i = 0.2$ $(i = 1, 2, 3)$. The learning rate value was $\beta = 0.05$ and the maximum number of iterations was set to $N = 100,000$. The percentage of the basin points for the four roots and the nonconverging behavior are shown in Fig. 8, while the projections $x_1 - x_2$, $x_1 - x_3$ and $x_2 - x_3$ are plotted in Figs. 9 and 10. Tables 4 and 5 are similar to Tables 2 and 3 respectively and contain the data of the statistical analysis applied to the second system.

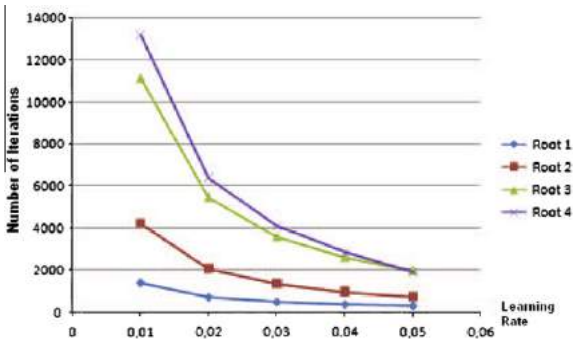

Fig. 11. The variation of the iteration number as a function of the learning rate for the second example system.

Table 5
The results for the best run for each one of the four roots of the second algebraic system.

Root	x_1^0	x_2^0	x_3^0	Iterations	F_1	F_2	F_3
1	+1.2	1	+1.6	0304	$-2\cdot10^{-8}$	0	0
2	+1.4	1	−0.2	0734	$-2\cdot10^{-8}$	-10^{-8}	0
3	+0.2	0	+0.8	1992	$+2\cdot10^{-8}$	0	-10^{-8}
4	−0.8	1	+1.0	1930	$+2\cdot10^{-8}$	0	-10^{-8}

Table 6
Simulation results for the proposed neural method, the trust-region-dogleg as well as the trust-region-reflective algorithms. In the results associated with the neural method the last column represents the learning rate associated with the run gave the presented results.

Root	x_1	x_2	x_3	Iterations	F_1	F_2	F_3	LRate
Neural based nonlinear system solver								
1	−1.000000	−1.000000	−1.000000	0006	-10^{-8}	-10^{-8}	10^{-8}	0.040
2	−1.000000	−1.000000	+1.000000	0797	10^{-8}	-10^{-8}	-10^{-8}	0.050
3	−1.000000	+0.999999	−0.999999	0886	-10^{-8}	-10^{-8}	0	0.050
4	+0.999999	−0.999999	+1.000000	1397	-10^{-8}	$+10^{-8}$	0	0.045
5	+1.000000	+0.999999	−0.999999	1602	-10^{-8}	$+10^{-8}$	0	0.045
6	+1.000000	+1.000000	+1.000000	0010	$+10^{-8}$	0	$+10^{-8}$	0.015
7	−1.224744	−0.000000	+1.224744	1524	-10^{-8}	$+10^{-8}$	0	0.030
8	+1.224744	−0.000000	−1.224744	0754	-10^{-8}	-10^{-8}	-10^{-8}	0.030
					$F_1\cdot10^{-\alpha}$	$F_2\cdot10^{-\alpha}$	$F_3\cdot10^{-\alpha}$	α
Trust-region-dogleg algorithm								
1	−0.999982	−0.999999	−1.000017	15	−0.123602	−0.062667	−0.180998	08
2	−0.999999	−1.000000	+0.999999	07	−0.105471	−0.071054	−0.004440	13
3	−1.000000	+1.000000	−0.999999	05	−0.000000	+0.666133	+0.444089	15
4	+1.000000	−0.999999	+1.000000	05	+0.174749	+0.013256	+0.373923	11
5	+0.999999	+0.999999	−1.000000	05	−0.654587	−0.462963	−0.385114	11
6	+0.999982	+0.999999	+1.000023	15	+0.298266	+0.122312	+0.317619	08
7	−1.224744	−0.000000	+1.224744	05	+0.888178	+0.444089	−0.888178	15
8	+1.224744	−0.000000	−1.224744	05	−0.103483	−0.068491	−0.069544	10
Trust-region-reflective algorithm								
1	−0.998635	−1.001183	−1.000180	12	−0.998635	−1.001183	−1.000180	04
2	−1.000000	−1.000000	+1.000000	06	+0.222044	+0.444089	−0.000000	15
3	−1.000000	+1.000000	−1.000000	05	+0.444089	+0.444089	+0.000000	15
4	+1.000000	−0.999999	+1.000000	05	+0.179523	+0.005684	+0.357758	11
5	+0.999999	+0.999999	−1.000000	05	−0.648014	−0.450794	−0.383248	11
6	+0.999999	+1.000000	+1.000000	07	+0.088817	+0.000000	+0.222044	14
7	−1.224744	−0.000000	+1.224744	05	+0.000000	+0.222044	+0.440089	15
8	+1.224744	−0.000000	−1.224744	07	+0.932587	+0.999200	−0.799360	14

The variation of the iteration number with the learning rate for the second example system is shown in Fig. 11 and it is characterized by the same features as in the case of the first example system discussed above. It can be easily seen that the variation of the learning rate is shown more clearly in Fig. 11 than Fig. 7 since the carves are shown in lin-lin axes and not in lin-log axes (as in Fig. 7) due to the large spreading of the iteration number for all the available roots.

Table 7
Simulation results for the Levenberg–Marquardt algorithm as well as the multivariate Newton–Raphson method with partial derivatives. The λ parameter of the Levenberg–Marquardt algorithm has the default value $\lambda = 0.01$.

Root	x_1	x_2	x_3	Iterations	$F_1 \cdot 10^{-\alpha}$	$F_2 \cdot 10^{-\alpha}$	$F_3 \cdot 10^{-\alpha}$	α
Levenberg–Marquardt algorithm								
1	−1.000000	−0.999999	−0.999999	04	−0.024558	+0.007482	−0.112176	11
2	−1.000000	−1.000000	+1.000000	07	+0.341948	−0.421884	−0.133226	13
3	−1.000000	+1.000000	−1.000000	05	+0.177635	−0.444089	−0.266453	14
4	+0.999999	−0.999999	+1.000000	05	+0.236477	+0.190514	+0.607514	12
5	+0.999999	+1.000000	−1.000000	05	−0.515010	−0.675504	−0.469135	11
6	+0.999982	+1.000000	+1.000000	04	+0.113242	−0.008881	+0.275335	13
7	−1.224744	−0.000000	+1.224744	05	+0.888178	+0.444089	−0.000000	15
8	+1.224744	−0.000000	−1.224744	10	−0.187116	+0.235652	+0.073100	09
Multivariate Newton–Raphson method with partial derivatives								
1	−1.000000	−1.000000	−1.000000	2865	−0.444089	−0.222044	−0.444089	15
2	+1.000000	−1.000000	+1.000000	0009	−0.444089	+0.000000	+0.888178	15
3	−1.000000	+1.000000	−1.000000	0005	+0.000000	+0.000000	+0.000000	00
4	+1.000000	−1.000000	+1.000000	0006	−0.044408	−0.044408	+0.177635	14
5	+1.000000	+1.000000	−1.000000	0006	−0.111022	+0.000000	+0.000000	15
6	+1.000000	−1.000000	+1.000000	0661	+0.000000	+0.000000	0.0000000	00
7	−1.224744	−0.000000	+1.224744	0005	−0.155431	−0.128785	−0.479616	13
8	−0.999999	−0.999999	−1.000000	0045	−0.888178	+0.666133	−0.888178	15

After the presentation of the experimental results that demonstrate the application of the proposed neural solver, let us proceed now to a comparison between the proposed method and other well known methods with respect to their performance. Since the proposed algorithm is an iterative one, we chose for this comparison well known iterative numerical algorithms such as the trust-region-dogleg [3], the trust-region-reflective [3] and the Levenberg–Marquardt algorithm [15,20], as well as the multivariate Newton–Raphson method with partial derivatives. The first three algorithms are supported by the *fsolve* function of the Optimization Toolbox of the MATLAB programming environment, while the Newton–Raphson MATLAB implementation can be found in the literature – for a theoretical description of this algorithm, see for example [28]. These algorithms were used to solve the problem using the initial conditions (x_0, y_0, z_0) of Table 3 associated with the best run (to save pages, this comparison is restricted only to the first example system with eight roots, since its extension to the second example system is straightforward).

The simulation results of these comparisons can be found in Tables 6 and 7. Table 6 contains the results of the proposed neural solver as well as the trust-region-dogleg and trust-region-reflective algorithms. On the other hand, Table 7 contains the results of the Levenberg–Marquardt and Newton–Raphson methods. Regarding the results associated with the proposed algorithm, the last column contains the learning rate value that gave the associated iteration number for the specified initial conditions, while for the other methods this column contains the order of magnitude of the functions evaluation in the position of the estimated root.

From Tables 6 and 7 it can be easily seen that even though the neural based approach gave the correct results and with the same accuracy, it requires much more iterations than the other methods, due to the small learning rate values that allow the network to converge (see the discussion above). Of course, this situation can be improved somehow by adjusting appropriately the learning rate value, as it can be easily seen by comparing the number of iterations in Tables 3 (a fixed learning rate value $\beta = 0.05$) and 6 (the optimum learning rate identified via experimentation). In some cases (namely for some methods and some roots) the neural method gave more accurate results than the other methods. From the last row in Table 7 it can be seen that the Newton–Raphson method converged to the Root 1 instead of Root 8, possibly because the associated initial condition is located near the boundary of the basin of attraction of these two roots. Besides this disadvantage of the neural based approach (which, actually is not a major problem since modern computing systems are characterized by very high speeds, and in most cases the duration of simulation time is not an issue), the proposed method is easy in its implementation (since it uses the classical back propagation approach and therefore it computes the function values in the first pass and the values of the partial derivatives via the estimation of δ parameters), in contrast with the other methods that are more difficult to implement since they require a lot of mathematics as well as complicated operations such as Jacobian evaluations at specified points.

8. Conclusions and future work

The objective of this research was the numerical estimation of the roots of complete 3×3 nonlinear algebraic systems of polynomial equations using back-propagation neural networks. The main advantage of this approach is the simple and straightforward solution of the system, by building a structure that simulates exactly the nonlinear system under consideration and find its roots via the classical back propagation approach. Depending on the position on the parameter space of the initial condition used in each case, each run converged to one of the eight possible solutions or diverged to infinity; therefore,

the proposed neural structure is capable of finding all the roots of such a system, although this cannot be done in only one step. One of the tasks associated with the future work on this subject is to improve or redesign the neural solver in order to find all roots in only one run.

This research is going to be extended to cover systems with more dimensions and different types of equations of non-polynomial nature. Special cases have to be investigated – as the case of double roots (and in general of roots with arbitrary multiplicity) and the associated theory has to be formulated. Finally, this structure can be extended to cover the case of complex roots and complex coefficients in nonlinear algebraic systems.

References

[1] A. Galantai, A. Jeney, Quasi-newton abs methods for solving nonlinear algebraic systems of equations, J. Optim. Theory Appl. 89 (3) (1996) 561–573.
[2] Chistine Jager, K.N. Dietmar Ralz, P. Pau, A combined method for enclosing all solutions of nonlinear systems of polynomial equations, Rel. Comput. 1 (1) (1995) 41–64.
[3] N.R. Conn, N.G., P. Toint, n.d. Trust-region methods, MPS/SIAM Series on Optimization.
[4] E. Spedicato, E. Bodon, A.P.N.M.-A. Abs methods and abspack for linear systems and optimization, a review, in: Proceedings of the third Seminar of Numerical Analysis, Zahedan, November 15/17 2000.
[5] E. Spedicato, Z. Huang, Numerical experience with newton-like methods for nonlinear algebraic systems, Computing 58 (1997) 69–89.
[6] C. Grosan, A. Abraham, Solving nonlinear equation systems using evolutionary algorithms, IEEE Trans. Syst. Man Cybernet. A 38 (3) (2008).
[7] I. Emiris, B.M. Vrahatis, Sign methods for counting and computing real roots of algebraic systems, Technical Report RR-3669, INRIA, Sophia Antipolis, 1999.
[8] J. Abaffy, A.E. Spedicato, The local convergence of abs methods for nonlinear algebraic equations, Numer. Math. 51 (1987) 429–439.
[9] J. Abaffy, A. Galantai, Conjugate direction methods for linear and nonlinear systems of algebraic equations, in: P. Rezsa, D. Greenspan (Eds.), Numerical Methods, Colloquia Mathematica Soc, Amsterdam, 1987, pp. 481–502.
[10] J. Abaffy, C.E. Spedicato, A class of direct methods for linear systems, Numerische Mathematik 45 (1984) 361–376.
[11] J. Abaffy, E. Spedicato, ABS Projection Algorithms: Mathematical Techniques for Linear and Nonlinear Equations, Ellis Horwood 1989.
[12] Jiwei Xue, Yaohui Zi, Y.F.-L.Y., Z. Lin, An intelligent hybrid algorithm for solving nonlinear polynomial systems, in: Proceedings of International Conference on Computational Science, LCNS 3037, 2004, pp. 26–33.
[13] K.G. Margaritis, M. Adamopoulos, K.G.-D.E. Artificial neural networks and iterative linear algebra methods, Parallel Algor. Appl. 3 (1) (1994) 31–44.
[14] K.G. Margaritis, M. Adamopoulos, K. Solving linear systems by artificial neural network energy minimisation, Universily of Macedonia Annals (vol. XII), 1993, pp. 502–525.
[15] K. Levenberg, A method for the solution of certain problems in least squares, Quart. Appl. Math. 2 (1944) 164–168.
[16] G. Li, Z. Zeng, A neural network algorithm for solving nonlinear equation systems, Proceedings of 20^{th} International Conference on Computational Intelligence and Security, Los Alamito, USA, 2008. pp. 20–23.
[17] D. Luo, Z. Han, Solving nonlinear equation systems by neural networks, Proc. Int. Conf. Syst. Man Cybernet. 1 (1995) 858–862.
[18] A. Margaris, M. Adamopoulos, Solving nonlinear algebraic systems using artificial neural networks, Proceedings of the 10th International Conference on Engineering Applications of Artificial Neural Networks, Thessaloniki, Greece, 2007. pp. 107–120.
[19] A. Margaris, K. Goulianas, Finding all roots of (2×2) nonlinear algebraic systems using backpropagation neural networks, Neural Comput. Appl. 21(2012) 891–904.
[20] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, IAM, J. Appl. Math 11 (1963) 431–441.
[21] K. Mathia, R. Saeks, Solving nonlinear equations using recurrent neural networks, World Congress on Neural Networks, Washington DC, USA, 1995.
[22] P. Mejzlik, A bisection method to find all solutions of a system of nonlinear equations, Proceedings of the Seventh International Conference on Domain Decomposition, October 27–30, 1993, The Pennsylvania State University, pp.277–282.
[23] D. Mishra, P.K. Kalva, Modified hopfield neural network approach for solving nonlinear algebraic equations, Eng. Lett. 14 (2007) (1).
[24] M.W. Smiley, C. Chun, An algorithm for finding all solutions of a nonlinear system, J. Comput. Appl. Math. 137 (2) (2001) 293–315.
[25] T-F. Tsai, M.-H.L. Finding all solutions of systems of nonlinear equations with free variables, Eng. Optim. 39 (6) (2007) 649–659.
[26] V. Dolotin, A. Morozov, Introduction to Nonlinear Algebra, World Scientific Publishing Company, 2007.
[27] V.N. Kublanovskaya, V.N. Simonova, An approach to solving nonlinear algebraic systems 2, J. Math. Sci. (3) (1996) 1077–1092.
[28] W. Press, S. Teukolsky, W.B. Numerical Recipes in C – The Art of Scientific Programming, 2nd ed., Cambridge University Press, 1992.
[29] S.O. Yusuke Nakaya, Find all solutions of nonlinear systems of equations using linear programming with guaranteed accuracies, J. Univ. Comput. Sci. 4 (2) (1998) 171–177.
[30] G. Zhang, L. Bi, Existence of solutions for a nonlinear algebraic system, Discrete Dynamics in, Nature and Society, 2009.