Neural Comput & Applic
DOI 10.1007/s00521-010-0488-z

ORIGINAL ARTICLE

Finding all roots of 2 × 2 nonlinear algebraic systems using back-propagation neural networks

Athanasios Margaris · Konstantinos Goulianas

Received: 2 June 2010 / Accepted: 9 November 2010

Abstract The objective of this research is the numerical estimation of the roots of a complete 2×2 nonlinear algebraic system of polynomial equations using a feed forward back-propagation neural network. The main advantage of this approach is the simple solution of the system, by building a structure—including product units—that simulates exactly the nonlinear system under consideration and find its roots via the classical back-propagation approach. Examples of systems with four or multiple roots were used, in order to test the speed of convergence and the accuracy of the training algorithm. Experimental results produced by the network were compared with their theoretical values.

Keywords Nonlinear algebraic systems · Neural networks · Generalized delta rule

1 Introduction

A typical nonlinear algebraic system is generally defined as $F(\mathbf{z}) = 0$ where F is a function $R^n \to R^n \, (n > 1)$ described by an n-dimensional vector in the form

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

with $f_i : R^n \to R \, (i = 1, 2, \ldots, n)$. It should be noted, that in general, there are no good methods for solving such a system: 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 $N - 1$ [1].

2 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; these algorithms introduced in 1984 by Abaffy et al. [2] to solve linear systems, but they also used with success to least squares methods and to nonlinear equations and system of equations [3]. The basic function of the initial ABS algorithms is to solve a determined or under-determined $n \times m$ linear system $Az = b \, (z \in R^n, b \in R^m, m \le n)$ where $rank(A)$ is arbitrary and $A^T = (\alpha_1, \alpha_2, \ldots, \alpha_m)$, with the system solution to be estimated as follows: [4]

1. Give $z_1 \in R^n$ arbitrary, $H_1 \in R^{n \times n}$ nonsingular arbitrary, and $u_i \in R^m$ arbitrary and nonzero. Set $i = 1$.
2. Compute the residual $r_i = Az_i - b$. If $r_i = 0$, stop (z_i solves the system), otherwise compute $s_i = H_i A^T u_i$. If $s_i \neq 0$, then, go to 3. If $s_i = 0$ and $\tau = u_i^T r_i = 0$,

A. Margaris (✉) · K. Goulianas
ATEI of Thessaloniki, Sindos, Thessaloniki, Greece
e-mail: amarg@uom.gr

K. Goulianas
e-mail: gouliana@it.teithe.gr

then set $z_{i+1} = z_i$, $H_{i+1} = H_i$ and go to 6. Otherwise, stop, since the system has no solution.

3. Compute the search vector $p_i = H_i^T x_i$ where $x_i \in R^n$ is arbitrary save for the condition $u_i^T A H_i x_i \neq 0$.
4. Update the estimate of the solution by $z_{i+1} = z_i - \alpha_i p_i$ and $\alpha_i = u_i^T r_i / u_i^T A p_i$.
5. Update the matrix H_i by using the equation

$$H_{i+1} = H_i - \frac{H_i A^T u_i W_i^T H_i}{w_i^T H_i A^T u_i}$$

where $w_i \in R^n$ is arbitrary save for the mathematical condition $w_i^T H_i A^T u_i \neq 0$.

6. If $i = m$, stop (z_{m+1} solves the system). Otherwise, give $u_{i+1} \in R^n$ arbitrary, linearly independent from $u_1, u_1, \ldots, u_i$. Increment i by one and go to 2.

In the above description, the matrices H_i, which are generalizations of the projection matrices, are known as Abaffians. The choice of those matrices as well as the quantities u_i, x_i and w_i, determine particular subclasses of the ABS algorithms. The most important of them are the following:

- The conjugate direction subclass; this subclass is obtained by setting $u_i = p_i$. It is well defined under the sufficient but not necessary condition that matrix A is symmetric and positive definite. Older versions of the Cholesky, Hestenes-Stiefel and Lanczos algorithms belong to this subclass.
- The orthogonality scaled subclass obtained by setting $u_i = Ap_i$. It is well defined if matrix A has full column rank and remains well defined even if $m > n$. It contains the ABS formulation of the QR algorithm, the GMRES and the conjugate residual algorithm.
- The optimally stable subclass obtained by making the assignment $u_i = (A^{-1})^T p_i$. The search vectors in this class are orthogonal. If $z_1 = 0$, then, the vector z_{i+1} is the vector of least Euclidean norm over the space defined as $Span(p_1, p_2, \ldots, p_i)$, and the solution is approached monotonically from bellow in the Euclidean norm. The methods of Gram-Schmidt and of Craig belong to this subclass.

The extension of the ABS methods for solving nonlinear algebraic systems is straightforward and there are many of them [5, 6]. The kth iteration of a typical nonlinear ABS algorithm includes the following steps:

1. Set $y_1 = z_k$ and $H_1 = E_n$ where E_n is the $n \times n$ unitary matrix.
2. Perform steps 3 to 6 for all $i = 1, 2, \ldots, n$.
3. Set $p_i = H_i^T x_i$.
4. Set $u_i = \sum_{j=1}^{i} \tau_{ji} y_j$ such that $\tau_{ji} > 0$ and $\sum_{j=1}^{i} \tau_{ji} = 1$.
5. Set $y_{i+1} = y_i - \frac{u_i^T F(y_i) p_i}{u_i^T A(u_i) p_i}$.
6. Set $H_{i+1} = H_i - \frac{H_i A^T u_i W_i^T H_i}{w_i^T H_i A^T u_i}$.
7. Set $x_{k+1} = y_{n+1}$.

A particular ABS method is defined by the arbitrary vectors of parameters

$$V = [u_1, u_2, \ldots, u_n]$$
$$W = [w_1, w_2, \ldots, w_n]$$
$$X = [x_1, x_2, \ldots, x_n]$$

as well as the single parameter τ_{ij}. These parameters are subjected to the conditions

$$u_i^T A(u_i) p_i \neq 0$$

and

$$w_i^T H_i A(u_i) u_i \neq 0 \quad (i = 1, 2, \ldots, n)$$

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 introduced quasi-Newton-based ABS methods known as row update methods. In these methods, the Jacobian matrix of the nonlinear system, $A(z) = F'(z)$ is not fixed, but its rows are updated at each iteration, having the form

$$A_k = \left[\alpha_1^{(k)}, \alpha_2^{(k)}, \ldots, \alpha_n^{(k)}\right]^T, \quad \left(\alpha_j^{(k)} \in R^n\right)$$

Based on this formalism, the kth iteration of the Huang row update ABS method is performed as follows:

1. Set $y_1^{(k)} = z_k$ and $H_1 = E_n$.
2. Perform steps 3 to 7 for all $i = 1, 2, \ldots, n$.
3. If $k = 1$ go to step 5; else set $s_i = y_i^{(k)} - y_i^{(k-1)}$ and $g_i = f\left(y_i^{(k)}\right) - f_i\left(y_i^{(k-1)}\right)$.
4. If $s_i \neq 0$, set $\alpha_i^{(k)} = \alpha_i^{(k-1)} + \frac{\left[g_i - \alpha_i^{(k-1)T} s_i\right] s_i}{sIi^T s_i}$; else set $\alpha_i^{(k)} = \alpha_i^{(k-1)}$.
5. Set $p_i = H_i^T x_i$.
6. Set $y_{i+1}^{(k)} = y_i^{(k)} - \frac{f_i\left(y_i^{(k)}\right) p_i}{p_i^T \alpha_i^{(k)}}$.
7. Set $H_{i+1} = H_i - \frac{H_i \alpha_i^{(k)} w_i^T H_i}{w_i^T H_i \alpha_i^{(k)}}$.
8. Set $x_{k+1} = y_{k+1}^{(k)}$.

Since the row update method does not require the a-priori computation of the Jacobian matrix, its computational cost is $O(n^3)$; however, an extra memory space is required for the $n \times n$ matrix $\left[y_1^{(k-1)}, y_2^{(k-1)}, \ldots, y_n^{(k-1)}\right]$.

Galantai and Jeney [7] have proposed alternative methods for solving nonlinear systems of equations that are combinations of the nonlinear ABS methods and quasi-Newton methods. Another interesting class of methods has been proposed by Kublanovskaya and Simonova [8] 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 equation

$$F(\lambda, \mu) = [f_1(\lambda, \mu), f_2(\lambda, \mu), \ldots, f_m(\lambda, \mu)]^T = 0$$

with the function $f_k(\lambda, \mu)\,(k = 1, 2, \ldots, m)$ to be a polynomial in the form

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

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

$$(\lambda, \mu) = [\varphi(\mu), \mu] \text{ and } (\lambda, \mu) = [\lambda, \tilde{\varphi}(\lambda)]$$

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. The second method is based on the factorization of $F(\lambda, \mu)$ into irreducible factors and on 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 et al. [9] developed a method for the counting and identification of the roots of a nonlinear algebraic system based on the concept of topological degree and by using bisection techniques (for the application of those techniques see also [10]. A method for solving such a system using linear programming techniques can be found in [11], while an interesting method that uses evolutionary algorithms is described in [12]. There are many other methods concerning this subject; some of them are described in [13–16].

3 Theory of nonlinear algebraic systems

According to the basic principles of the nonlinear algebra [17], a complete nonlinear algebraic system of n polynomial equations with n unknowns $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ is identified completely by the number of equations n, and their degrees $(s_1, s_2, \ldots, s_n)$, it is expressed mathematically as

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

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

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

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

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

When all degrees coincide, i.e. $s_1 = s_2 = \cdots = s_n = s$, the resultant $\Re_{n|s}$ is reduced to a simple polynomial of degree $d_{n|s} = deg_A \Re_{n|s} = ns^{n-1}$ 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 contravariant 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 [18].

Even though the notion of the resultant has been defined for homogenous nonlinear equations, it can also describe nonhomogenous algebraic equations as well. In the general case, the resultant $\Re$ satisfies the nonlinear Cramer rule

$$\Re_{s_1, s_2, \ldots, s_n}\left\{A^{(k)}(Z_k)\right\} = 0 \tag{6}$$

where Z_k is the kth component of the solution of the nonhomogenous system, and $A^{(k)}$ is the kth column of the coefficient matrix A.

Since in the next sections, we propose neural models for solving the complete 2×2 nonlinear algebraic system, let us examine the special case $s_1 = s_2 = s$. The complete 2×2 nonlinear system is defined as $F_1(x, y) = 0$ and $F_2(x, y) = 0$ where

$$F_1(x,y) = \sum_{k=0}^{s} \alpha_k x^k y^{s-k} = \alpha_s \prod_{j=1}^{s} (x - \lambda_j y) = y^s \tilde{A}(t) \tag{7}$$

$$F_2(x,y) = \sum_{k=0}^{s} \beta_k x^k y^{s-k} = \beta_s \prod_{j=1}^{s} (x - \mu_j y) = y^s \tilde{B}(t) \tag{8}$$

with $t = y/x$, $x = z_1$ and $y = z_2$. The resultant of this system has the form

$$\Re = (\alpha_s \beta_s)^s \prod_{i,j=1}^{s} (\lambda_i - \mu_j) = (\alpha_0 \beta_0)^s \prod_{i,j=1}^{s} \left(\frac{1}{\mu_j} - \frac{1}{\lambda_i} \right) \tag{9}$$

(where $\Re = \Re_{2|s}\{F_1, F_2\}$) and it can be expressed as the determinant of the $2s \times 2s$ matrix of coefficients. In the particular case of a linear map ($s = 1$) where the system is described by the linear equations

$$F_1(x,y) = \alpha_0 y + \alpha_1 x = 0 \tag{10}$$

$$F_2(x,y) = \beta_0 y + \beta_1 x = 0 \tag{11}$$

the resultant reduces to the determinant of the 2×2 matrix, and therefore, it has the form

$$\Re_{2|1}\{A\} = \begin{vmatrix} \alpha_1 & \alpha_0 \\ \beta_1 & \beta_0 \end{vmatrix} \tag{12}$$

On the other hand, for $s = 2$ (this is the case of interest in this project) the homogenous nonlinear algebraic system has the form

$$\alpha_{11}x^2 + \alpha_{13}xy + \alpha_{12}y^2 = 0$$
$$\alpha_{21}x^2 + \alpha_{23}xy + \alpha_{22}y^2 = 0$$

with a resultant in the form

$$\Re_2 = \Re_{2|2}\{A\} = \begin{vmatrix} \alpha_{11} & \alpha_{13} & \alpha_{12} & 0 \\ 0 & \alpha_{11} & \alpha_{13} & \alpha_{12} \\ \alpha_{21} & \alpha_{23} & \alpha_{22} & 0 \\ 0 & \alpha_{21} & \alpha_{23} & \alpha_{22} \end{vmatrix} \tag{13}$$

In complete accordance with the theory of the linear algebraic systems, the above system has a solution if the resultant $\Re$ satisfies the equation $\Re = 0$. The nonhomogenous complete 2×2 nonlinear system can be derived from the homogenous one by adding, and the linear terms therefore has the form

$$\alpha_{11}x^2 + \alpha_{13}xy + \alpha_{12}y^2 + \alpha_{14}x + \alpha_{15}y = \alpha_{16}$$
$$\alpha_{21}x^2 + \alpha_{23}xy + \alpha_{22}y^2 + \alpha_{24}x + \alpha_{25}y = \alpha_{26}$$

To solve this system, we note that if (X, Y) is the desired solution, then, the solution of the homogenous systems

$$\left(\alpha_{11}X^2 + \alpha_{14}X - \alpha_{16}\right)z^2 + (\alpha_{13}X + \alpha_{15})yz + \alpha_{12}y^2 = 0$$
$$\left(\alpha_{21}X^2 + \alpha_{24}X - \alpha_{26}\right)z^2 + (\alpha_{23}X + \alpha_{25})yz + \alpha_{22}y^2 = 0$$

and

$$\alpha_{11}x^2 + (\alpha_{13}Y + \alpha_{14})xz + \left(\alpha_{12}Y^2 + \alpha_{15}Y - \alpha_{16}\right)z^2 = 0$$
$$\alpha_{21}x^2 + (\alpha_{23}Y + \alpha_{24})xz + \left(\alpha_{22}Y^2 + \alpha_{25}Y - \alpha_{26}\right)z^2 = 0$$

has the form $(z, y) = (1, Y)$ and $(x, z) = (X, 1)$ for the first and the second system, respectively. But this implies that the corresponding resultants vanish, i.e. the X variable satisfies the equation

$$\begin{vmatrix} \alpha_{11}X^2 + \alpha_{14}X - \beta_1; \alpha_{13}X + \alpha_{15}; \alpha_{12}; 0 \\ 0; \alpha_{11}X^2 + \alpha_{14}X - \beta_1; \alpha_{13}X + \alpha_{15}; \alpha_{12} \\ \alpha_{21}X^2 + \alpha_{24}X - \beta_2; \alpha_{23}X + \alpha_{25}; \alpha_{22}; 0 \\ 0; \alpha_{21}X^2 + \alpha_{24}X - \beta_2; \alpha_{23}X + \alpha_{25}; \alpha_{22} \end{vmatrix} = 0 \tag{14}$$

while the Y variable satisfies the equation

$$\begin{vmatrix} \alpha_{11}; \alpha_{13}Y + \alpha_{14}; \alpha_{12}Y^2 + \alpha_{15}Y - \beta_1; 0 \\ 0; \alpha_{11}; \alpha_{13}Y + \alpha_{14}; \alpha_{12}Y^2 + \alpha_{15}Y - \beta_1 \\ \alpha_{21}; \alpha_{23}Y + \alpha_{24}; \alpha_{22}Y^2 + \alpha_{25}Y - \beta_2; 0 \\ 0; \alpha_{21}; \alpha_{23}Y + \alpha_{24}; \alpha_{22}Y^2 + \alpha_{25}Y - \beta_1 \end{vmatrix} = 0 \tag{15}$$

(in the above equations the symbol ";" is used as column separator in the tabular environment). Therefore, the variables X and Y got separated, and they can be estimated from separate algebraic equations. However, these solutions are actually correlated: the above equations are of the 4th power in X and Y, respectively, but making a choice of one of the four $X's$, one fixes associated choice of Y. Thus, the total number of solutions for the complete 2×2 nonlinear algebraic system is $s^2 = 4$.

The extension of this description for the complete 3×3 system and in general, for the complete $n \times n$ system is straightforward. Regarding the type of the system roots—namely, real, imaginary, or complex roots—it depends on the values of the coefficients of the matrix A.

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 [19, 20] 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 [22] 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 [23] 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 [24] used the gradient descent rule with a variable step size to solve nonlinear algebraic systems at very rapid convergence and very high accuracy.

The main idea behind the proposed approach for solving a nonlinear algebraic system of p equations with p unknowns (see Margaris and Adamopoulos [25]) is to construct a network with p output neurons, with the output of the lth neuron $(1 \leq \ell \leq p)$ to represent the value of the left hand part of the lth equation; in this way, the real output vector of the output layer is an estimate of the values of the left hand part of all the system equations[1]. An efficient way to construct such a network is shown in Fig. 1a. In this figure, the proposed model is composed of four layers, an input layer, with one linear (i.e. 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 nonlinear (i.e. product units) that generate the nonlinear terms x^2, xy and y^2 and finally, an output layer of summation units whose total input is the expression of the left hand side of the corresponding equation. To achieve this goal, 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 second and the third layer to the neurons of the 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. The only variable-weight synapses are the two synapses joining the input neuron with the two linear neurons of the first hidden layer. Since this network structure is capable of providing an estimate of the value of the left hand part of the system, it is clear that if it be trained using the traditional back-propagation algorithm and the fixed terms of the system equations α_{16} and α_{26} as the desired output vector, then, after a complete successful training, the weights of the two variable-weight synapses will contain the components x and y of one of the system roots.

The neural structure described earlier uses the identity function $y = f(x) = x$ as the activation function for all the network neurons and it has been tested in practice for solving typical complete nonlinear algebraic systems of two equations with two unknowns. The neural solvers were built using the Neuralware simulation tool and trained by the back-propagation algorithm with a learning rate equal to 0.7 and a momentum equal to 0.0. No bias units were available. The average number of iterations for the network to converge was a few thousands iterations (5,000–10,000), and the global training error was equal to $10^{-7} - 10^{-8}$. An example of complete 2×2 nonlinear algebraic system solved by the network was the one defined by the equations

$$0.1x^2 + 0.1y^2 + 0.3xy - 0.1x + 0.1y = +0.8$$
$$0.1x^2 - 0.3y^2 + 0.2xy - 0.1x - 0.3y = +0.4$$

This system has four real roots, whose estimation with the Mathematica software tool gave the values

$$(x_1, y_1) = (+12.23410, -03.82028)$$
$$(x_2, y_2) = (-04.11479, +01.29870)$$
$$(x_3, y_3) = (-01.40464, -01.07718)$$
$$(x_4, y_4) = (+02.28537, +00.59876)$$

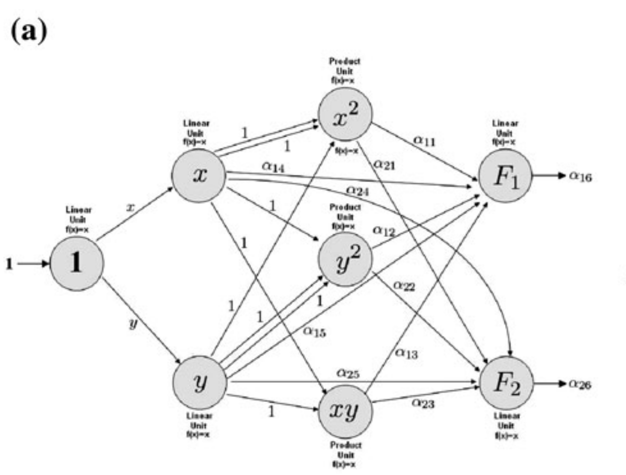
The root of this system estimated by the neural solver was the $(x, y) = (-1.40464, -1.07718)$ and therefore the neural network reached the third root (x_3, y_3). It is not difficult to note that the estimation accuracy is very good, a fact that was actually expected, since the global training error of the neural network has a value of $10^{-7} - 10^{-8}$.

Even though the neural solver described earlier had an excellent performance regarding the estimation accuracy, it suffers from a strong drawback: it always reaches the same root and therefore it has very limiting capabilities, since in general, the nonlinear systems to be solved have many roots. A remarkable thing is that the network converged always to the same root, even though the initial conditions were very close to other roots, different than the estimated one. An attempt to use this structure to identify the period-2 fixed points of the Henon chaotic map (see Margaris and Adamopoulos [26]) showed that the root of the system identified by the network was always an unstable one. This restrictive behavior of the neural solver was attributed to the identity activation function used for the neurons of the output layer. This identity function is a linear one and therefore it does not perform well in the general case of a nonlinear algebraic system.

To overcome this limitation, a nonlinear activation function was used for the neurons of the output layer; furthermore, the network structure was redesigned from the beginning to become more flexible and easy to be implemented. More specifically, the following modifications performed:

- In the initial design, the neuron produced the x^2 term was a product unit with two input synapses each one of them had an input value of x and a weight value of 1; therefore, the total input to the neuron was estimated as $\upsilon = (1 \cdot x) \cdot (1 \cdot x) = x^2$ and because the activation function was the identity function $f(\upsilon) = \upsilon$, the output of the neuron was estimated as $v = f(\upsilon)\,|_{\upsilon = x^2} = x^2$. In

[1] The originality of this idea has to be attributed to Dr. Miltiades Adamopoulos, an Emeritus Professor of the Technological Educational Institute of Thessaloniki.



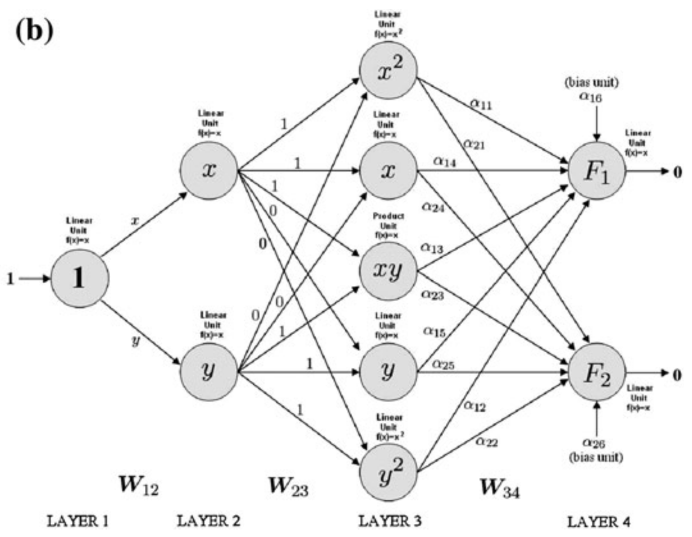


Fig. 1 The structure of the initial (**a**) and the updated (**b**) **2** × **2** nonlinear system solver

the new model, there is a simple linear summation unit that gets a simple input of $\upsilon = x$ and uses the activation function $f(\upsilon) = \upsilon^2$ to produce the output $v = x^2$. The same is true for the unit that produces the y^2 term.

- In the initial design, the neurons produced the linear and the nonlinear terms were grouped into distinct hidden layers; a layer of two linear units producing the terms x and y and a second hidden layer of product units producing the nonlinear terms x^2, xy, and y^2. Even though the network worked fine, in the new design the two linear units of the second layer were duplicated to the third layer in order to draw all the synapses between neurons of consecutive layers as it is required by the back-propagation algorithm. This was not the case in the initial design where there were synapses joining directly the linear units of the second layer with the output units of the fourth layer. This duplication increases a bit the degree of complexity of the network structure, but allows the easy extraction of the weight update equations by applying the well-known back-propagation approach. It is clear that if a nonlinear term is not present to the algebraic system, the associated synaptic weights are set to the zero value.
- Finally, in order to overcome the limitation of getting always the same root, the configuration of the output layer has been altered accordingly. In the new model, the fixed term of the system α_{16} and α_{26} is now associated with the two output neurons as bias units. The desired output vector now has only zero values, and furthermore, the activation function of the output neuron has been set to the hyperbolic tangent function. The reasoning behind this new setup is that the hyperbolic tangent of zero is equal to zero; therefore, if the weights of the two variable-weight synapses are equal to the components of one of the system roots, the total input of the output units must have a zero value and due to the use of the hyperbolic tangent function, their outputs should be zero, too.

The adaptation of these modifications solved all the problems caused by the original design, and the new network model is capable of estimating all the system roots as it is described in the next section. This new updated structure is presented in Fig. 1b.

5 Building the back-propagation equations

Since the new version of the neural simulator is compatible with the requirements of the back-propagation approach and the gradient descent algorithm, the update equations of the synaptic weights can be constructed immediately, by applying the well-known traditional approach. In the following description, the parameter $\boldsymbol{W}_{ij}$ denotes the synaptic weights matrix between the neurons of the layers i and j. Based on the network structure, these matrices are defined as

$$\boldsymbol{W}_{12} = \begin{bmatrix} \boldsymbol{W}_{12}^{(1)} & \boldsymbol{W}_{12}^{(2)} \end{bmatrix} = [x \quad y]$$

$$\boldsymbol{W}_{23} = \begin{bmatrix} \boldsymbol{W}_{23}^{(11)} & \boldsymbol{W}_{23}^{(12)} & \boldsymbol{W}_{23}^{(13)} & \boldsymbol{W}_{23}^{(14)} & \boldsymbol{W}_{23}^{(15)} \\ \boldsymbol{W}_{23}^{(21)} & \boldsymbol{W}_{23}^{(22)} & \boldsymbol{W}_{23}^{(23)} & \boldsymbol{W}_{23}^{(24)} & \boldsymbol{W}_{23}^{(25)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$\boldsymbol{W}_{34} = \begin{bmatrix} \boldsymbol{W}_{34}^{(11)} & \boldsymbol{W}_{34}^{(12)} \\ \boldsymbol{W}_{34}^{(21)} & \boldsymbol{W}_{34}^{(22)} \\ \boldsymbol{W}_{34}^{(31)} & \boldsymbol{W}_{34}^{(32)} \\ \boldsymbol{W}_{34}^{(41)} & \boldsymbol{W}_{34}^{(42)} \\ \boldsymbol{W}_{34}^{(51)} & \boldsymbol{W}_{34}^{(52)} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{14} & \alpha_{24} \\ \alpha_{13} & \alpha_{23} \\ \alpha_{15} & \alpha_{25} \\ \alpha_{12} & \alpha_{22} \end{bmatrix}$$

In the above notation, the element $\boldsymbol{W}_{k\ell}^{(ij)}$ denotes the weight of the synapse that joins the ith neuron of the kth layer with the jth neuron of the lth layer.

The representation of the input/output characteristics of the network neurons is based on the notation $v = f(\upsilon)$ where υ is the input to the neuron, f its activation function and v the output produced by it. Using vector notation and the indices 2, 3 and 4 to denote the second, the third and the fourth layer, the input vectors of the three layers are defined as

$$\boldsymbol{\upsilon}_2 = \left[\boldsymbol{\upsilon}_2^{(1)} \quad \boldsymbol{\upsilon}_2^{(2)}\right] = [x \quad y]$$

$$\begin{aligned}\boldsymbol{\upsilon}_3 &= \left[\boldsymbol{\upsilon}_3^{(1)} \quad \boldsymbol{\upsilon}_3^{(2)} \quad \boldsymbol{\upsilon}_3^{(3)} \quad \boldsymbol{\upsilon}_3^{(4)} \quad \boldsymbol{\upsilon}_3^{(5)}\right] \\ &= [x \quad x \quad xy \quad y \quad y]\end{aligned}$$

$$\boldsymbol{\upsilon}_4 = \left[\boldsymbol{\upsilon}_4^{(1)} \quad \boldsymbol{\upsilon}_4^{(2)}\right] = [F_1(x,y) \quad F_2(x,y)]$$

the neuron activation functions have the form

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

$$\begin{aligned}\boldsymbol{f}_3 &= \left[\boldsymbol{f}_3^{(1)}(\upsilon) \quad \boldsymbol{f}_3^{(2)}(\upsilon) \quad \boldsymbol{f}_3^{(3)}(\upsilon) \quad \boldsymbol{f}_3^{(4)}(\upsilon) \quad \boldsymbol{f}_3^{(5)}(\upsilon)\right] \\ &= [\upsilon^2 \quad \upsilon \quad \upsilon \quad \upsilon \quad \upsilon^2]\end{aligned}$$

$$\boldsymbol{f}_4 = \left[\boldsymbol{f}_4^{(1)} \quad \boldsymbol{f}_4^{(2)}\right] = [\tanh(\upsilon) \quad \tanh(\upsilon)]$$

and the corresponding output vector are described by the expression

$$v_2 = \left[v_2^{(1)} \quad v_2^{(2)}\right] = [x \quad y]$$

$$\begin{aligned}v_3 &= \left[v_3^{(1)} \quad v_3^{(2)} \quad v_3^{(3)} \quad v_3^{(4)} \quad v_3^{(5)}\right] \\ &= [x^2 \quad x \quad xy \quad y \quad y^2]\end{aligned}$$

$$v_4 = \left[v_4^{(1)} \quad v_4^{(2)}\right] = [\tanh[F_1(x,y)] \quad \tanh[F_2(x,y)]]$$

Regarding the type of the neural processing elements, all the neurons expect the third multiplicative neuron of the third layer that generates the term xy are typical linear (i.e. summation) units.

5.1 Forward pass

According to the back-propagation algorithm [27], the network input to the jth hidden neuron of some hidden layer is given by the expression

$$\text{net}_{pj}^h = \sum_{i=1}^{N} w_{ji}^h x_{pi} + \vartheta_j^h \tag{16}$$

where w_{ji}^h is the weight of the connection from the ith input unit, x_{pi} is the ith component of the pth training input vector and ϑ_{ij} is the bias of the jth hidden unit (the superscript h denotes simply a hidden unit). The output of that unit is estimated as

$$i_{pj} = f_j^h\left(\text{net}_{pj}^h\right) \tag{17}$$

where f_j^h is the activation function of the jth hidden unit. This theory, based on the network structure and on the notation introduced in the previous section, is formulated as follows:

The input to the neurons of the second hidden layer has the form

$$\boldsymbol{\upsilon}_2^{(i)} = \boldsymbol{W}_{12}^{(i)} \quad (i = 1, 2) \tag{18}$$

or equivalently

$$\boldsymbol{\upsilon}_2^{(1)} = \boldsymbol{W}_{12}^{(1)} = x$$

$$\boldsymbol{\upsilon}_2^{(2)} = \boldsymbol{W}_{12}^{(2)} = y$$

while the outputs produced by them are estimated as

$$v_2^{(i)} = \boldsymbol{f}_2^{(i)}\left(\boldsymbol{\upsilon}_2^{(i)}\right) \quad (i = 1, 2) \tag{19}$$

or equivalently

$$v_2^{(1)} = \boldsymbol{f}_2^{(1)}(x) = x$$

$$v_2^{(2)} = \boldsymbol{f}_2^{(2)}(y) = y$$

The third layer (i.e. the second hidden layer) is composed of four summation units corresponding to the values $i = 1, 2, 4, 5$ and one product unit associated with the value $i = 3$. The inputs to the four summation units are estimated as

$$\begin{aligned}\boldsymbol{\upsilon}_3^{(i)} &= \sum_{j=1}^{2} \boldsymbol{W}_{23}^{(ji)} v_2^{(j)} = \boldsymbol{W}_{23}^{(1i)} v_2^{(1)} + \boldsymbol{W}_{23}^{(2i)} v_2^{(2)} \\ &= \boldsymbol{W}_{23}^{(1i)} x + \boldsymbol{W}_{23}^{(2i)} y\end{aligned} \tag{20}$$

(for the values $i = 1, 2, 4, 5$) or equivalently

$$\boldsymbol{\upsilon}_3^{(1)} = \boldsymbol{W}_{23}^{(11)} x + \boldsymbol{W}_{23}^{(21)} y = x$$

$$\boldsymbol{\upsilon}_3^{(2)} = \boldsymbol{W}_{23}^{(12)} x + \boldsymbol{W}_{23}^{(22)} y = x$$

$$\boldsymbol{\upsilon}_3^{(4)} = \boldsymbol{W}_{23}^{(14)} x + \boldsymbol{W}_{23}^{(24)} y = y$$

$$\boldsymbol{\upsilon}_3^{(5)} = \boldsymbol{W}_{23}^{(15)} x + \boldsymbol{W}_{23}^{(25)} y = y$$

while the input value to the product unit has the form

$$\boldsymbol{\upsilon}_3^{(3)} = \prod_{j=1}^{2} \boldsymbol{W}_{23}^{(j3)} v_2^{(j)} = \left(\boldsymbol{W}_{23}^{(13)} v_2^{(1)}\right)\left(\boldsymbol{W}_{23}^{(23)} v_2^{(2)}\right) = xy \tag{21}$$

The outputs of the neurons of the third layer are given by the equations

$$v_3^{(1)} = \boldsymbol{f}_3^{(1)}\left(\boldsymbol{\upsilon}_3^{(1)}\right) = x^2 \tag{22}$$

$$v_3^{(2)} = \boldsymbol{f}_3^{(2)}\left(\boldsymbol{\upsilon}_3^{(2)}\right) = x \tag{23}$$

$$v_3^{(3)} = \boldsymbol{f}_3^{(3)}\left(\boldsymbol{\upsilon}_3^{(3)}\right) = xy \tag{24}$$

$$v_3^{(4)} = \boldsymbol{f}_3^{(4)}\left(\boldsymbol{\upsilon}_3^{(4)}\right) = y \tag{25}$$

$$v_3^{(5)} = \boldsymbol{f}_3^{(5)}\left(\boldsymbol{\upsilon}_3^{(5)}\right) = y^2 \tag{26}$$

Finally, the inputs to the two linear neurons of the output layer have the form

$$\begin{aligned}\boldsymbol{\upsilon}_4^{(i)} &= \sum_{j=1}^{5} \boldsymbol{W}_{34}^{(ji)} v_3^{(j)} - \alpha_{i6} = \boldsymbol{W}_{34}^{(1i)} v_3^{(1)} + \boldsymbol{W}_{34}^{(2i)} v_3^{(2)} \\ &\quad + \boldsymbol{W}_{34}^{(3i)} v_3^{(3)} + \boldsymbol{W}_{34}^{(4i)} v_3^{(4)} + \boldsymbol{W}_{34}^{(5i)} v_3^{(5)} - \alpha_{i6} \\ &= \boldsymbol{W}_{34}^{(1i)} x^2 + \boldsymbol{W}_{34}^{(2i)} x + \boldsymbol{W}_{34}^{(3i)} xy + \boldsymbol{W}_{34}^{(4i)} y \\ &\quad + \boldsymbol{W}_{34}^{(5i)} y^2 - \alpha_{i6}\end{aligned} \tag{27}$$

(for the values $i = 1, 2$) or equivalently

$$\begin{aligned}\boldsymbol{\upsilon}_4^{(1)} &= \boldsymbol{W}_{34}^{(11)} x^2 + \boldsymbol{W}_{34}^{(21)} x + \boldsymbol{W}_{34}^{(31)} xy + \boldsymbol{W}_{34}^{(41)} y \\ &\quad + \boldsymbol{W}_{34}^{(51)} y^2 - \alpha_{16}\alpha_{11}x^2 + \alpha_{14}x + \alpha_{13}xy + \alpha_{15}y \\ &\quad + \alpha_{12}y^2 - \alpha_{16} = F_1(x,y)\end{aligned} \tag{28}$$

$$\begin{aligned}\boldsymbol{\upsilon}_4^{(2)} &= \boldsymbol{W}_{34}^{(12)} x^2 + \boldsymbol{W}_{34}^{(22)} x + \boldsymbol{W}_{34}^{(32)} xy + \boldsymbol{W}_{34}^{(42)} y + \boldsymbol{W}_{34}^{(52)} y^2 - \alpha_{26} \\ &= \alpha_{21}x^2 + \alpha_{24}x + \alpha_{23}xy + \alpha_{25}y + \alpha_{22}y^2 - \alpha_{26} \\ &= F_2(x,y)\end{aligned} \tag{29}$$

while the corresponding outputs are defined as

$$v_4^{(1)} \equiv o_1 = \tanh(\boldsymbol{\upsilon}_4^{(1)}) = \tanh[F_1(x,y)] \tag{30}$$

$$v_4^{(2)} \equiv o_2 = \tanh(\boldsymbol{\upsilon}_4^{(2)}) = \tanh[F_2(x,y)] \tag{31}$$

5.2 Backward pass: estimation of the δ parameters

After the forward pass, the real output of the kth output neuron is subtracted by the corresponding desired output to define the quantity

$$\delta_{pk} = y_{pk} - o_{pk} \tag{32}$$

and then, the δ value of that neuron is estimated as

$$\delta_{pk}^{o} = (y_{pk} - o_{pk}) f_k^{o'}\left(\text{net}_{pk}^{o}\right) = \delta_{pk} f_k^{o'}\left(\text{net}_{pk}^{o}\right) \tag{33}$$

where net_{pk}^{o} is the net input to the kth output neuron and f_k^o the activation function of that neuron. This error is then back-propagated, and the corresponding δ value of the jth hidden neuron is estimated as

$$\delta_{pj}^{h} = f_j^{h'}(net_{pj}^{h}) \sum_{k=1}^{M} \delta_{pk}^{o} w_{kj}^{o} \tag{34}$$

where M is the number of output neurons and w_{kj}^{o} is the weight of the connection from the jth hidden neuron to the kth output neuron. This equation holds for the hidden neurons of the last hidden layer but the δ values of the intermediate hidden neurons are estimated in a similar way.

To apply this theory to the current neural structure, we adopt a notation similar to the one used so far; therefore, the vectors of δ parameters of the second, third and fourth layer are denoted as

$$\delta_2 = \begin{bmatrix} \delta_2^{(1)} & \delta_2^{(2)} \end{bmatrix}$$

$$\delta_3 = \begin{bmatrix} \delta_3^{(1)} & \delta_3^{(2)} & \delta_3^{(3)} & \delta_3^{(4)} & \delta_3^{(5)} \end{bmatrix}$$

$$\delta_4 = \begin{bmatrix} \delta_4^{(1)} & \delta_4^{(2)} \end{bmatrix}$$

Starting with the output layer and keeping in mind that (a) the real network outputs have the form

$$o_1 = \tanh[F_1(x,y)] \; o_2 = \tanh[F_2(x,y)]$$

and (b) the corresponding desired outputs have the values $d_1 = d_2 = 0$ and the output neurons activation function is the hyperbolic tangent function whose derivative is defined as

$$\tanh'(x) = \frac{d\{\tanh(x)\}}{dx} = 1 - \tanh^2(x)$$

it can be easily proven that the elements of the δ_4 vector are given by the equations

$$\begin{aligned}\delta_4^{(1)} &= (d_1 - o_1)\tanh'\left(\upsilon_4^{(1)}\right) = -o_1 \tanh'[F_1(x,y)] \\ &= -\tanh[F_1(x,y)]\{1 - \tanh^2[F_1(x,y)]\}\end{aligned} \tag{35}$$

$$\begin{aligned}\delta_4^{(2)} &= (d_2 - o_2)\tanh'\left(\upsilon_4^{(2)}\right) = -o_2 \tanh'[F_2(x,y)] \\ &= -\tanh[F_2(x,y)]\{1 - \tanh^2[F_2(x,y)]\}\end{aligned} \tag{36}$$

In the next step, we can use the δ values of the output neurons to find the δ values of the third layer neurons through the equation

$$\delta_3^{(i)} = \left(\sum_{j=1}^{2} \boldsymbol{W}_{34}^{(ij)} \delta_4^{(j)}\right) \cdot \left[\boldsymbol{f}_3^{(i)'}\left(\boldsymbol{\upsilon}_3^{(i)}\right)\right] (i = 1,2,3,4,5) \tag{37}$$

with the derivatives that appear in the above equation to be estimated easily as $\boldsymbol{f}_3^{(1)'}\left(\boldsymbol{\upsilon}_3^{(1)}\right) = 2x$, $\boldsymbol{f}_3^{(2)'}\left(\boldsymbol{\upsilon}_3^{(2)}\right) = 1$, $\boldsymbol{f}_3^{(4)'}\left(\boldsymbol{\upsilon}_3^{(4)}\right) = 1$, $\boldsymbol{f}_3^{(5)'}\left(\boldsymbol{\upsilon}_3^{(5)}\right) = 2y$, $\boldsymbol{f}_3^{(3)'x}\left(\boldsymbol{\upsilon}_3^{(3)}\right) = y$, and $\boldsymbol{f}_3^{(3)'y}\left(\boldsymbol{\upsilon}_3^{(3)}\right) = x$, respectively. It has to be mentioned that for the product unit of the third layer, two different derivatives were estimated, one with respect to x and one with respect to y. This step is necessary, since the nonlinear term produced by this neuron is the xy term, a function of both x and y. Using the above equations, it can be easily proven that

$$\begin{aligned}\delta_3^{(1)} &= \left(\boldsymbol{W}_{34}^{(11)}\delta_4^{(1)} + \boldsymbol{W}_{34}^{(12)}\delta_4^{(2)}\right)(2x) \\ &= 2x\{\alpha_{11}[-f(F_1)\{1 - f^2(F_1)\}] \\ &\quad + \alpha_{21}[-f(F_2)\{1 - f^2(F_2)\}]\}\end{aligned} \tag{38}$$

$$\begin{aligned}\delta_3^{(2)} &= \boldsymbol{W}_{34}^{(21)}\delta_4^{(1)} + \boldsymbol{W}_{34}^{(22)}\delta_4^{(2)} \\ &= \alpha_{14}\left[-f(F_1)\left\{1-f^2(F_1)\right\}\right] \\ &\quad + \alpha_{24}\left[-f(F_2)\left\{1-f^2(F_2)\right\}\right]\end{aligned} \tag{39}$$

$$\begin{aligned}\delta_3^{(3)x} &= \left(\boldsymbol{W}_{34}^{(31)}\delta_4^{(1)} + \boldsymbol{W}_{34}^{(32)}\delta_4^{(2)}\right)(y) \\ &= y\left\{\alpha_{13}\left[-f(F_1)\left\{1-f^2(F_1)\right\}\right]\right. \\ &\quad \left.+\alpha_{23}\left[-f(F_2)\left\{1-f^2(F_2)\right\}\right]\right\}\end{aligned} \tag{40}$$

$$\begin{aligned}\delta_3^{(3)y} &= \left(\boldsymbol{W}_{34}^{(31)}\delta_4^{(1)} + \boldsymbol{W}_{34}^{(32)}\delta_4^{(2)}\right)(x) \\ &\quad + x\left\{\alpha_{13}\left[-f(F_1)\left\{1-f^2(F_1)\right\}\right]\right. \\ &\quad \left.+\alpha_{23}\left[-f(F_2)\left\{1-f^2(F_2)\right\}\right]\right\}\end{aligned} \tag{41}$$

$$\begin{aligned}\delta_3^{(4)} &= \boldsymbol{W}_{34}^{(41)}\delta_4^{(1)} + \boldsymbol{W}_{34}^{(42)}\delta_4^{(2)} \\ &= \alpha_{15}\left[-f(F_1)\left\{1-f^2(F_1)\right\}\right] \\ &\quad + \alpha_{25}\left[-f(F_2)\left\{1-f^2(F_2)\right\}\right]\end{aligned} \tag{42}$$

$$\begin{aligned}\delta_3^{(5)} &= \left(\boldsymbol{W}_{34}^{(51)}\delta_4^{(1)} + \boldsymbol{W}_{34}^{(52)}\delta_4^{(2)}\right)(2y) \\ &= 2y\left\{\alpha_{21}\left[-f(F_1)\left\{1-f^2(F_1)\right\}\right]\right. \\ &\quad \left.+\alpha_{22}\left[-f(F_2)\left\{1-f^2(F_2)\right\}\right]\right\}\end{aligned} \tag{43}$$

where for the sake of brevity the symbols F_1 and F_2 were used, instead the functions $F_1(x, y)$ and $F_2(x, y)$, as well as the notation $f(x) = \tanh(x)$.

Finally, based on δ_3 vector, the delta values of the second layer neurons can be estimated by the equation

$$\delta_2^{(i)} = \left(\sum_{j=1}^{5}\boldsymbol{W}_{23}^{(ij)}\delta_3^{(j)}\right)\cdot\left[\boldsymbol{f}_2^{(i)\prime}\left(\boldsymbol{v}_2^{(i)}\right)\right], \quad (i=1,2) \tag{44}$$

with the values of all the derivatives to be equal to unity since the activation function of those two neurons is the identity function. Using the nonzero elements of the $\boldsymbol{W}_{23}$ matrix as well as the partial derivatives with respect to x and y for the corresponding neurons, their δ values are estimated as

$$\begin{aligned}\delta_2^{(1)} &= \sum_{j=1}^{5}\boldsymbol{W}_{23}^{(1j)}\delta_3^{(j)} = \delta_3^{(1)} + \delta_3^{(2)} + \delta_3^{(3)x} \\ &= -\left[f(F_1)\left\{1-f^2(F_1)\right\}(2\alpha_{11}x+\alpha_{13}y+\alpha_{14})\right. \\ &\quad \left.+f(F_2)\left\{1-f^2(F_2)\right\}(2\alpha_{21}x+\alpha_{23}y+\alpha_{24})\right]\end{aligned} \tag{45}$$

$$\begin{aligned}\delta_2^{(2)} &= \sum_{j=1}^{5}\boldsymbol{W}_{23}^{(2j)}\delta_3^{(j)} = \delta_3^{(3)y} + \delta_3^{(4)} + \delta_3^{(5)} \\ &= -\left[f(F_1)\left\{1-f^2(F_1)\right\}(2\alpha_{21}y+\alpha_{13}x+\alpha_{15})\right. \\ &\quad \left.+f(F_2)\left\{1-f^2(F_2)\right\}(2\alpha_{22}y+\alpha_{23}x+\alpha_{25})\right]\end{aligned} \tag{46}$$

By using the equations

$$\begin{aligned}\frac{\partial F_1(x,y)}{\partial x} &= 2\alpha_{11}x+\alpha_{13}y+\alpha_{14} \\ \frac{\partial F_1(x,y)}{\partial y} &= 2\alpha_{12}y+\alpha_{13}x+\alpha_{15} \\ \frac{\partial F_2(x,y)}{\partial x} &= 2\alpha_{21}x+\alpha_{23}y+\alpha_{24} \\ \frac{\partial F_2(x,y)}{\partial y} &= 2\alpha_{22}y+\alpha_{23}x+\alpha_{25}\end{aligned}$$

the above relations get the form

$$\begin{aligned}\delta_2^{(1)} &= -\left[f(F_1)\left\{1-f^2(F_1)\right\}\frac{\partial F_1(x,y)}{\partial x}\right. \\ &\quad \left.+f(F_2)\left\{1-f^2(F_2)\right\}\frac{\partial F_2(x,y)}{\partial x}\right] \\ &= -\sum_{i=1}^{2}f(F_i)\left\{1-f^2(F_i)\right\}\frac{\partial F_i(x,y)}{\partial x} \\ &= \sum_{i=1}^{2}\delta_4^{(i)}\frac{\partial F_i(x,y)}{\partial x}\end{aligned} \tag{47}$$

$$\begin{aligned}\delta_2^{(2)} &= -\left[f(F_1)\left\{1-f^2(F_1)\right\}\frac{\partial F_1(x,y)}{\partial y}\right. \\ &\quad \left.+f(F_2)\left\{1-f^2(F_2)\right\}\frac{\partial F_2(x,y)}{\partial y}\right] \\ &= -\sum_{i=1}^{2}f(F_i)\left\{1-f^2(F_i)\right\}\frac{\partial F_i(x,y)}{\partial y} \\ &= \sum_{i=1}^{2}\delta_4^{(i)}\frac{\partial F_i(x,y)}{\partial y}\end{aligned} \tag{48}$$

5.3 Correcting the synaptic weights

In the last stage of the back-propagation algorithm, the correction of the weight of the variable-weight synapses must be performed; these weights are equal to the components x and y of one of the system roots. According to the back-propagation theory, the correction of the synaptic weights is performed by an equation of the form

$$\boldsymbol{W}_{12}^{(i)} = \boldsymbol{W}_{12}^{(i)} + \beta\delta_2^{(i)} = \boldsymbol{W}_{12}^{(i)} + \beta\sum_{j=1}^{2}\delta_4^{(j)}\frac{\partial F_j(x,y)}{\partial \boldsymbol{W}_{12}^{(j)}} \tag{49}$$

for the values $i = (1,2)$. By noting that the mean square training error has the form

$$E = \frac{1}{2}\sum_{j=1}^{2}(d_j - o_j) = \frac{1}{2}\sum_{j=1}^{2}o_j^2 = \frac{1}{2}\sum_{j=1}^{2}f^2[F_j(x,y)] \tag{50}$$

it can be easily seen that

$$
\begin{aligned}
-\frac{\partial E}{\partial \mathbf{W}_{12}^{(j)}} &= \sum_{j=1}^{2} f[F_j(x,y)] \frac{\partial f[F_j(x,y)]}{\partial \mathbf{W}_{12}^{(j)}} \\
&= \sum_{j=1}^{2} \underbrace{f[F_j(x,y)]\{1-f^2[F_j(x,y)]\}}_{\delta_4^{(j)}} \frac{\partial F_j(x,y)}{\partial \mathbf{W}_{12}^{(j)}} \\
&= \sum_{j=1}^{2} \delta_4^{(j)} \frac{\partial F_j(x,y)}{\partial \mathbf{W}_{12}^{(j)}} \qquad (51)
\end{aligned}
$$

and the weight update equation will get its final form

$$
\mathbf{W}_{12}^{(i)} = \mathbf{W}_{12}^{(i)} - \beta \frac{\partial E}{\partial \mathbf{W}_{12}^{(j)}}, \quad (i = 1, 2) \qquad (52)
$$

in full accordance with the associated equation

$$
w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \beta \left\{ f_j^{h'}\left(\mathrm{net}_{pj}^{h}\right) \sum_{k=1}^{M} \delta_{pk}^{o} w_{kj} \right\} x_j \qquad (53)
$$

(in the above notation the β parameter is the learning rate; see the book of Freeman & Skapura [27]). It is clear that in the neural solver there is only one input with a constant value equal to unity; furthermore, all the synaptic weights except the ones keeping the system roots are fixed and therefore this third stage of the back-propagation algorithm is much simpler than the general case.

6 Experimental results

The proposed neural structure described in the previous sections was tested by solving typical complete 2 × 2 nonlinear algebraic systems, whose roots were known in advance, in order to evaluate the simulation accuracy. For each system, the basin of attraction of each root was identified together with the percentage of the initial conditions associated with each root. The basin of attraction of a root is defined as the set of points formed by the values of the initial conditions for which the neural network converged to that root; the ratio of the number of the 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, and special curves that show the variation in the number of iterations used in each case were plotted.

The operation of the neural network was simulated in MATLAB, and the components x and y of each root were estimated by using recurrently the last equation and for a learning rate value $\beta = 0.3$. The first one of the nonlinear systems used for testing is defined by the equations

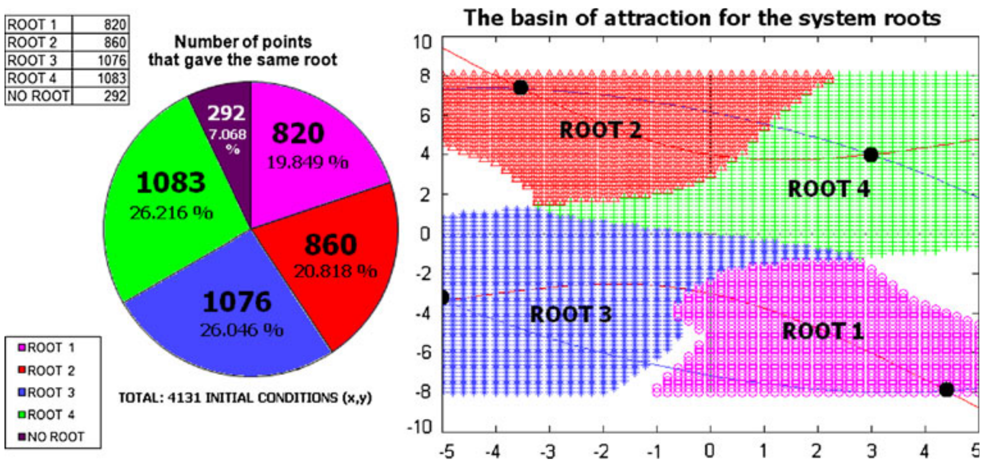
$$
x^2 + y^2 + xy + x + y = 44
$$
$$
x^2 - y^2 - xy + x + y = -12
$$

with four distinct real roots with values

ROOT 1:	$(x_1, y_1) = (+4.41870, -7.94356)$
ROOT 2:	$(x_2, y_2) = (-3.48896, +7.31612)$
ROOT 3:	$(x_3, y_3) = (-4.92974, -3.37256)$
ROOT 4:	$(x_4, y_4) = (+3.00000, +4.00000)$

as they were evaluated by solving the system in Mathematica. The neural network ran 4130 times with the initial conditions (x, y) of the two synaptic weights to vary in the intervals $-5 \le x \le 5$ and $-8 \le y \le 8$ with a variation step value $= \Delta y = 0.2$. For each pair (x, y), the network either reached one of the four system roots or it was unable to estimate one of those roots. The basin of attraction for the four roots (namely the region of the initial conditions that converged to each root) as well as the number of points of that basin is depicted graphically in Fig. 2.

The experimental results associated with this system are shown in Tables 1 and 2. Table 1 contains for each one of the identified roots, the number of systems converged to that root, the average number of iterations and the mean



ROOT 1	820
ROOT 2	860
ROOT 3	1076
ROOT 4	1083
NO ROOT	292

Fig. 2 The basin of attraction of the four system roots and the number of points associated with each one of them for the regions $-5 \le x \le 5$, $-8 \le y \le 8$ and a variation step $= \Delta y = 0.2$

Table 1 The experimental results for the four roots of the first algebraic system

Root no.	Average iterations	STDEV of iterations	Average MSE	STDEV of MSE	Average Xroot	Average Yroot
1	1,104	1755.946592	1.46E-12	1.8675E-14	+4.418691424	−7.94355695
2	0561	0563.723188	9.72E-13	1.6079E-14	−3.489556350	+7.31618430
3	0694	1265.226933	9.38E-13	3.6563E-14	−4.929735044	−3.37256040
4	0569	0861.438739	9.56E-13	3.7685E-14	+2.999999929	+3.99999943

Table 2 The experimental results for the best run for each root of the first algebraic system

Root	Xinit	Yinit	Iterations	MSE	Xroot	Yroot	$F_1(X,Y)$	$F_2(X,Y)$
1	+4.2	−8.0	236	9.451E-13	+4.41869883	−7.94356724	+0.00000137	+0.00000014
2	−3.2	+7.4	248	9.524E-13	−3.48896352	+7.31612832	+0.00000136	+0.00000021
3	−4.6	−3.6	099	9.752E-13	−4.92973223	−3.37255936	−0.00000137	−0.00000025
4	+2.8	+4.4	127	9.833E-13	+3.00000441	+4.00000061	+0.00000127	+0.00000059

square error as well as the standard deviation of those values, and also the average value of the x and y components for each root. On the other hand, the data of Table 2 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, y), the number of iterations, the mean square error, the x and y components of the associated root, as well as the values of the functions $F_1(x, y)$ and $F_2(x, y)$. The variation of the iteration number for each system root is plotted in Fig. 3.

The second complete 2 × 2 nonlinear algebraic system used for testing the network is defined by the equations

$$0.25x^2 + y^2 = 1$$
$$(x-1)^2 + y^2 = 1$$

and it has two distinct roots with values

$$(x_1, y_1) = \left(\frac{2}{3}, +\frac{2}{3}\sqrt{2}\right) = (0.6666666, +0.942809)$$
$$(x_2, y_2) = \left(\frac{2}{3}, -\frac{2}{3}\sqrt{2}\right) = (0.6666666, -0.942809)$$

as well as a double root with value

$$(x_3, y_3) = (2, 0)$$

labeled as ROOT1, ROOT2 and ROOT3, respectively (see Ko et al. [28]). In this case, the network ran 1678 times with the synaptic weights to vary in the interval $-2 \le x, y \le 2$ and a variation step $\Delta x = \Delta y = 0.1$. The experimental results for this nonlinear system are shown in Tables 3 and 4 and Fig. 4.

Even though the system behaved well in finding the two distinct roots (x_1, y_1) and (x_2, y_2), major issues emerged regarding the identification of the double root (x_3, y_3). For example, the basin of attraction for this root is composed of points of the horizontal axis only. On the other hand, the 860 initial conditions (x, y) that did not lead to a root have shown a strange behavior. More specifically, there were 819 pairs (x, y) (a percentage of 95.232%) that converged to the point (− 2, 0), i.e. to the mirror point of the double root with respect to the vertical axis. These pairs correspond to the values $-2 \le x \le 0$ and $-2 \le y \le 2$ (with a variation step of $\Delta x = \Delta y = 0.1$) that define the basin of attraction of this point. This is not difficult to see that the point (− 2, 0) is a root for the first of the two equations, i.e. the equation $0.25x^2 + y^2 = 1$. The MSE value for all these systems was the same and equal to $0.499329 \approx 0.5$, while the average values of the identified root components and the functions F_1 and F_2 were $X_{\text{average}} = -1.999917746$, $Y_{\text{average}} = 0.000010783$, $F_1(x, y)_{\text{average}} = -2.20562E-07$ and $F_2(x, y)_{\text{average}} = 3.999794257 \approx 4$, respectively (Fig. 5).

Regarding the remaining 39 (out of 860) pairs (x, y) (a percentage of 4.534%) they correspond to the values $x = 0$ and $-2 \le y \le 2$ (with $\Delta y = 0.1$) and belong to the vertical axis of the system. All these points lead to a system point with coordinates $(x, y) = (0. - 0.90969095)$ with MSE value equal to 0.091284 and function values $F_1(x, y) = -0.172460$ and $F_2(x, y) = 0.413769$. It has to be noticed that all the systems lead to exactly the same values and there is no need to estimate the averages used above. It is clear that this behavior needs further investigation.

7 Conclusions and future work

The objective of this research was the numerical estimation of the roots of complete 2 × 2 nonlinear algebraic systems

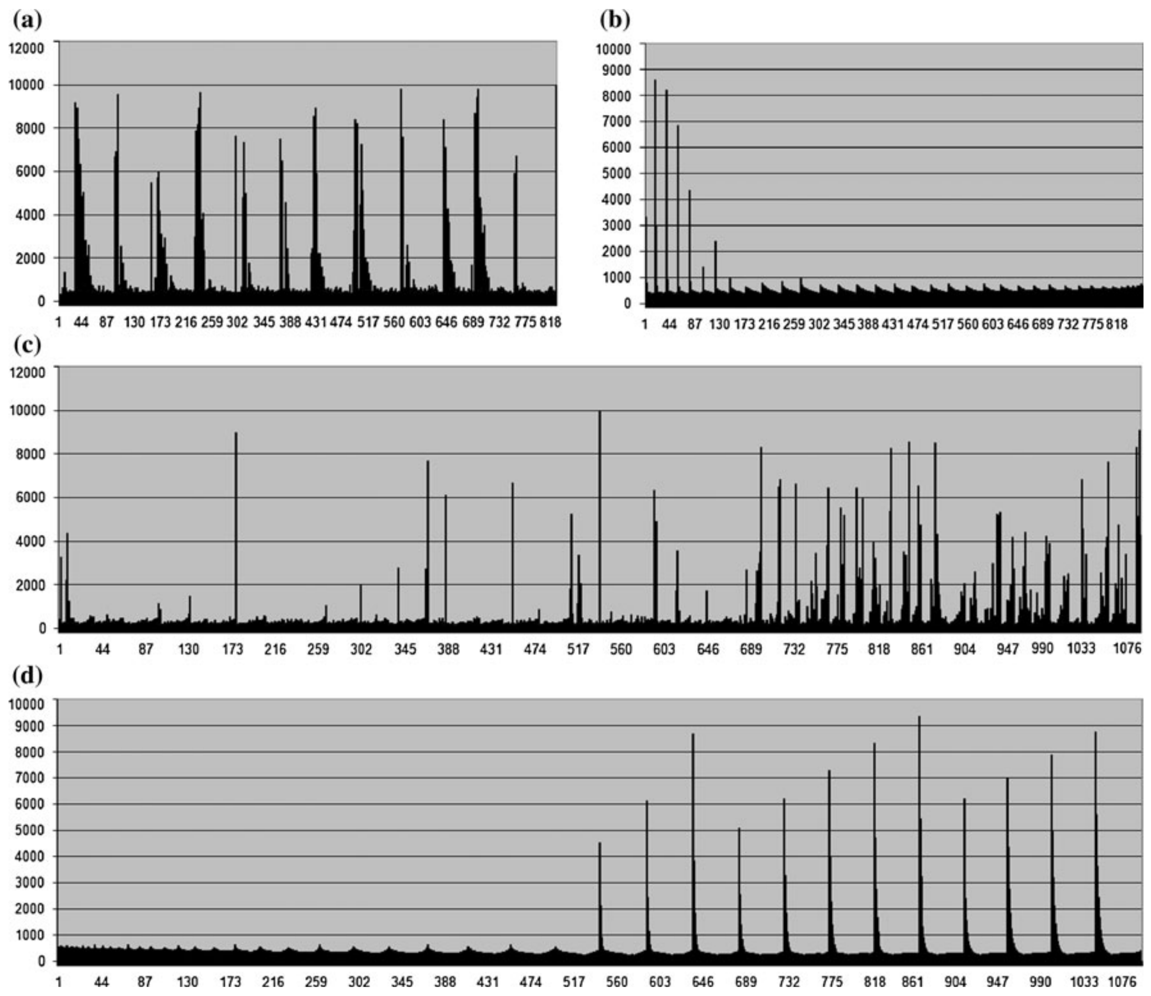


Fig. 3 The variation of the iteration number of the four roots of the first algebraic system

Table 3 The experimental results for the three roots (two distinct and one double) of the second algebraic system

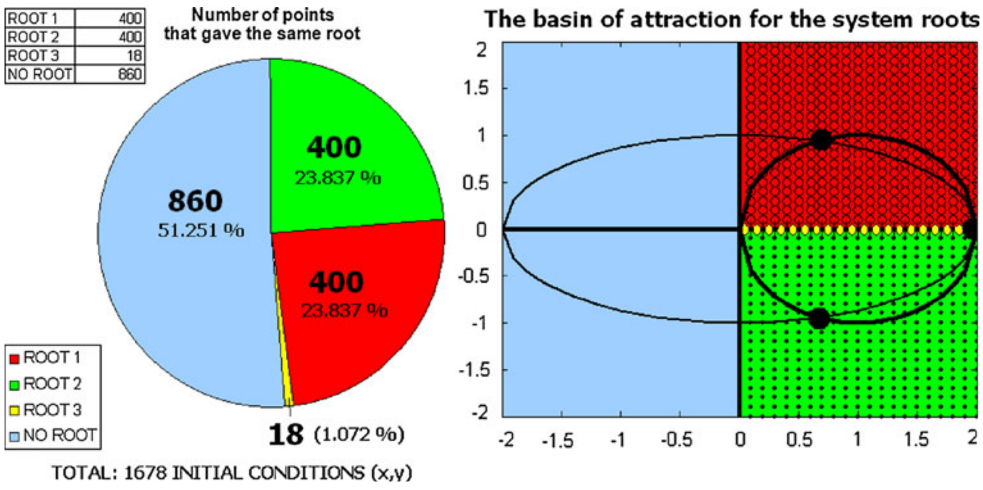
Root no.	Average iterations	STDEV of iterations	Average MSE	STDEV of MSE	Average Xroot	Average Yroot
1	599	67.489931	9.79E-13	1.16849E-14	+0.666667917	+0.942808954
2	599	67.489931	9.79E-13	1.16849E-14	+0.666667917	−0.942808954
3	049	12.989815	2.65E-13	1.50344E-14	+1.999999152	+0.000000000

Table 4 The experimental results for the best run for each root of the second algebraic system

Root	Xinit	Yinit	Iterations	MSE	Xroot	Yroot	$F_1(X,Y)$	$F_2(X,Y)$
1	+0.7	+1.3	345	9.845E-13	+0.6666698	+0.94280882	+6.2E-07	−1.3E-07
2	+0.7	−1.3	345	9.845E-13	+0.6666698	−0.94280882	+6.2E-07	−1.3E-07
3	+1.9	+0.0	033	7.160E-13	+1.9000000	+0.00000000	−8.7E-07	−8.7E-07

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 of the initial condition used in each case, each run converged to one of the four possible solutions or diverged

Fig. 4 The basin of attraction of the three system roots and the number of points associated with each one of them for the regions $-2 \leq x, y \leq 5$ and a variation step $\Delta x = \Delta y = 0.1$



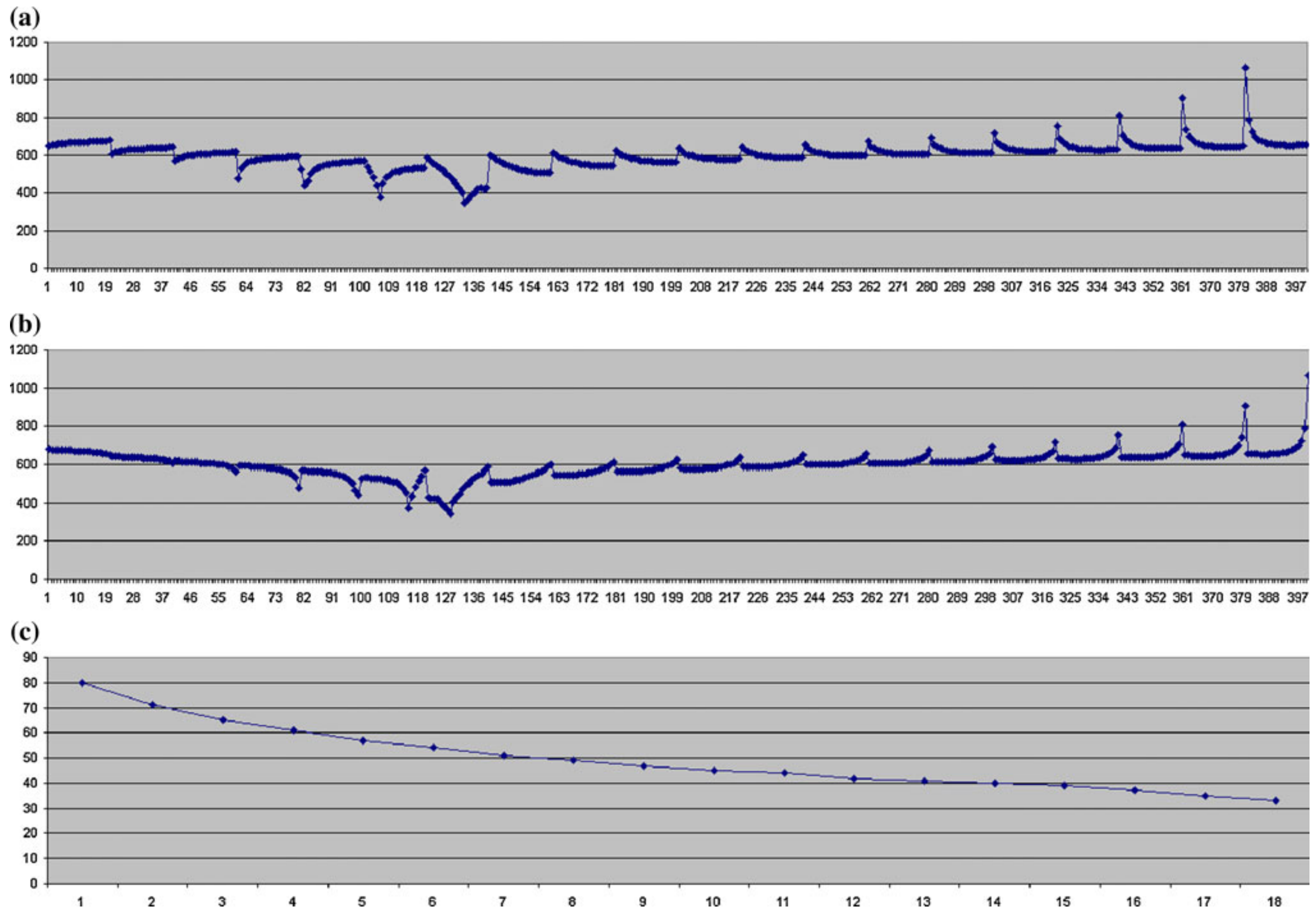


Fig. 5 The variation of the iteration number of the three roots of the second algebraic system

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 (as for example the 3 × 3 complete nonlinear algebraic systems, and the general $n \times n$ system) and different types of equations of nonpolynomial 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. Press W, Teukolsky S, Vetterling W, Flannery B (1992) Numerical recipes in C—the art of scientific programming, 2nd edn. Cambridge University Press, Cambridge
2. Abaffy J, Broyden CG, Spedicato E (1984) A class of direct methods for linear systems. Numerische Mathematik 45:361–376
3. Abaffy J, Spedicato E (1989) ABS projection algorithms: mathematical techniques for linear and nonlinear equations. Ellis Horwood, Chichester
4. Spedicato E, Bodon E, Del Popolo A, Mahdavi-Amiri N (2000) ABS methods and ABSPACK for linear systems and optimization, a review. In: Proceedings of the 3rd seminar of numerical analysis, Zahedan, November 15/17, University of Zahedan
5. Abaffy J, Galantai A (1987) Conjugate direction methods for linear and nonlinear systems of algebraic equations. Colloquia Mathematica Soc. Jfinos Bolyai. Numerical Methods, Miskolc, Hungary, 1986; Edited by R6zsa P and Greenspan D, North Holland, Amsterdam, Netherlands, vol 50, pp 481–502
6. Abaffy J, Galantai A, Spedicato E (1987) The local convergence of ABS methods for nonlinear algebraic equations. Numerische Mathematik 51:429–439
7. Galantai A, Jeney A (1996) Quasi-Newton ABS methods for solving nonlinear algebraic systems of equations. J Opt Theory Appl 89(3):561–573
8. Kublanovskaya VN, Simonova VN (1996) An approach to solving nonlinear algebraic systems. 2. J Math Sci 79(3):1077–1092
9. Emiris I, Mourrain B, Vrahatis M (1999) Sign methods for counting and computing real roots of algebraic systems. Technical Report RR-3669. INRIA, Sophia Antipolis
10. Mejzlik P (1994) A bisection method to find all solutions of a system of nonlinear equations. Domain decomposition methods in scientific and engineering computing. Book Chapter, AMS
11. Nakaya Y, Oishi S (1998) Find all solutions of nonlinear systems of equations using linear programming with guaranteed accuracies. J Univ Comput Sci 4(2):171–177
12. Grosan C, Abraham A (1996) Solving nonlinear equation systems using evolutionary algorithms. In: Proceedings of genetic & evolutionary computation conference, Seattle, USA, Proceedings on CD
13. Chistine J, Ralz D, Nerep K, PauΠ (March 1995) A combined method for enclosing all solutions of nonlinear systems of polynomial equations. Reliable computing. Springer, Berlin 1(1): 41–64
14. Tsai T-F, Lin M-H (2007) Finding all solutions of systems of nonlinear equations with free variables. Eng Opt 39(6):649–659
15. Michael WS, Chun C (2001) An algorithm for finding all solutions of a nonlinear system. J Comput Appl Math 137(2):293–315
16. Xue J, Zi Y, Feng Y, Yang L, Lin Z (2004) An intelligent hybrid algorithm for solving nonlinear polynomial systems. In: Bubak M et al (eds) Proceedings of international conference on computational science, LCNS 3037, pp 26–33
17. Dolotin V, Morozov A (September 2006) Introduction to nonlinear algebra v.2. online document found in the Arxiv repository (http://www.arxiv.org), ITEP, Moscow, Russia
18. Zhang G, Bi L, Existence of solutions for a nonlinear algebraic system. Discrete dynamics in nature and society, Volume 2009, Article ID 785068, available online from the URL http://www.hindawi.com/journals/ddns/2009/785068.html
19. Margaritis KG, Adamopoulos M, Goulianas K (1993) Solving linear systems by artificial neural network energy minimisation. University of Macedonia Annals, vol XII, pp 502–525, (in Greek)
20. Margaritis KG, Adamopoulos M, Goulianas K, Evans DJ (1994) Artificial neural networks and iterative linear algebra methods. Parallel Algorithms Appl 3(1–2):31–44
21. Mathia K, Saeks R (1995) Solving nonlinear equations using recurrent neural networks. World congress on neural networks, July 17–21, Washington DC, USA
22. Mishra D, Kalva PK, Modified Hopfield neural network approach for solving nonlinear algebraic equations. Engineering Letters, 14:1, EL_14_01_23 (advance online publication: 12 February 2007)
23. Luo D, Han Z (1995) Solving nonlinear equation systems by networks. In: Proceedings of international conference on systems, man and cybernetics, vol 1, pp 858–862
24. Li Guimei, Zhezhao Z (2008) A neural network algorithm for solving nonlinear equation systems. In: Proceedings of 20th international conference on computational intelligence and security, pp 20–23, Los Alamito, USA
25. Margaris A, Adamopoulos M (2007) Solving nonlinear algebraic systems using artificial neural networks. In: Proceedings of the 10th international conference on engineering applications of artificial neural networks, August, Thessaloniki, Greece
26. Margaris A, Adamopoulos M (2009) Identifying fixed points of Henon map using artificial neural networks. In: Proceedings of the 2nd chaotic modeling and simulation international conference, CHAOS2009, June 1–5, Chania, Crete, Greece, pp 107–120
27. Freeman JA, Skapura DM (1999) Neural networks: algorithms, applications and programming techniques. Addison Wesley: ISBN 0-201-51376-5
28. Ko KH, Sakkalis T, Patrikalakis NM (2004) Nonlinear polynomial systems: multiple roots and their multiplicities. In: Giannini F, Pasko A (eds) Proceedings of shape modelling international conference, SMI 2004, Genova, Italy