Solving Nonlinear Algebraic Systems Using Artificial Neural Networks

Athanasios Margaris and Miltiadis Adamopoulos
University of Macedonia
Department of Applied Informatics
Egnatia 156, Thessaloniki, Greece
amarg@uom.gr, miltos@uom.gr

Abstract

The objective of this research is the proposal of neural network structures capable of solving nonlinear algebraic systems with polynomials equations; however these structures can be easily modified to solve systems of nonlinear equations of any type. The basic features of the proposed structures include among others the usage of product units trained by the gradient descent algorithm. The presented theory is applied for solving 2×2 *and* 3×3 *nonlinear algebraic systems 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 has the form $F(\vec{z}) = 0$ where the function F is defined as $F : R^n \rightarrow R^n$ $(n > 1)$ and it is an n-dimensional vector in the form $F = [f_1, f_2, \ldots, f_n]^T$ with $f_i : R^n \rightarrow R\,(i = 1, 2, \ldots, n)$. 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, the system solving requires the identification of points that are mutually common to N unrelated zero-contour hyper-surfaces each of dimension $N - 1$ [1].

2 Review of previous work

The solution of nonlinear algebraic systems is in general 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 class introduced in 1984 by Abaffy, Broyden and Spedicato [2] for initially solving linear systems, but to be extended later to solve non linear 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 $Az = b$ $(z \in R^n, b \in R^m, m \leq n)$ 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, goto 3. If $s_i = 0$ and $\tau = u_i^T r_i = 0$, then set $z_{i+1} = z_i$, $H_{i+1} = H_i$ and goto 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 $H_{i+1} = H_i - H_i A^T u_i W_i^T H_i / w_i^T H_i A^T u_i$ where $w_i \in R^n$ is arbitrary save for the 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 goto 2.

In the above description, the matrices H_i which are generalizations of the projection matrices, are known as Abaffians. The choice of these 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, 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. Old 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 setting $u_i = (A^{-1})^T p_i$. 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 $Span(p_1, p_2, \ldots, p_i)$, 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 k_{th} 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 $y_{i+1} = y_i - u_i^T F(y_i) p_i / u_i^T A(u_i) p_i$.
6. Set $H_{i+1} = H_i - H_i A^T u_i W_i^T H_i / w_i^T H_i A^T u_i$.
7. Set $x_{k+1} = y_{n+1}$.

A particular ABS method is defined by the arbitrary parameters $V = [u_1, u_2, \ldots, u_n]$, $W = [w_1, w_2, \ldots, w_n]$, $X = [x_1, x_2, \ldots, x_n]$ and τ_{ij}. These parameters are subjected to the conditions $u_i^T A(u_i) p_i \neq 0$ and $w_i^T H_i A(u_i) u_i \neq 0$ $(i = 1, 2, \ldots, n)$. 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 evaluations, Huang introduced quasi-Newton based AVS methods known as row update methods, to which, the Jacobian matrix of the non linear system, $A(z) = F'(z)$ is not fixed, but its rows are updated at each iteration, and therefore has the form $A_k = [\alpha_1^{(k)}, \alpha_2^{(k)}, \ldots, \alpha_n^{(k)}]^T$ $(\alpha_j^{(k)} \in R^n)$. Based on this formalism, the k_{th} 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 $k = 1$ goto step 5; else set $s_i = y_i^{(k)} - y_i^{(k-1)}$ and $g_i = f_(y_i^{(k)}) - f_i(y_i^{(k-1)})$.
4. If $s_i \neq 0$, set $\alpha_i^{(k)} = \alpha_i^{(k-1)} + [g_i - \alpha_i^{(k-1)T} s_i] s_i / sIi^T s_i$; else set $\alpha_i^{(k)} = \alpha_i^{(k-1)}$.
5. Set $p_i = H_i^T x_i$.
6. Set $y_{i+1}^{(k)} = y_i^{(k)} - f_i(y_i^{(k)}) p_i / p_i^T \alpha_i^{(k)}$.
7. Set $H_{i+1} = H_i - H_i \alpha_i^{(k)} w_i^T H_i / w_i^T H_i \alpha_i^{(k)}$.
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 $[y_1^{(k-1)}, y_2^{(k-1)}, \ldots, y_n^{(k-1)}]$.

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 have 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 vector equation

$$F(\lambda, \mu) = [f_1(\lambda, \mu), f_2(\lambda, \mu), \ldots, f_m(\lambda, \mu)]^T = 0 \quad (1)$$

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

$$f_k(\lambda, \mu) = [\alpha_{ts}^{(k)} \mu^t + \cdots + \alpha_{0s}^{(k)}] \lambda^s + \ldots + [\alpha_{t0}^{(k)} \mu^t + \cdots + \alpha_{00}^{(k)}] \quad (2)$$

In Equation (2), 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 able to find the zero-dimensional roots (λ^*, μ^*), i.e. the pairs of fixed numbers satisfying the nonlinear system, as well as the one-dimensional roots $(\lambda, \mu) = (\varphi(\mu), \mu)$ 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.

The last family of methods mentioned in this section is the one proposed by Emiris, Mourrain and Vrahatis [9]. These methods are able to count and identify the roots of a nonlinear algebraic system based to the concept of topological degree and by using bisection techniques.

3 ANNs as linear systems solvers

An effective tool for solving linear as well as nonlinear algebraic systems, is the artificial neural networks (ANNs). Before the description of their usage in the nonlinear case, let us briefly describe how they can be used to solve linear systems of m equations with n unknowns in the form $Az = B$ [11]. In this approach the design of the neural network allows the minimization of the Euclidean distance $r = Az - b$ or the equivalent residual $r = Cz - d$ where $C = A^T A$ and $d = A^T b$. The proposed neural network is a Hopfield network with a Lyapunov energy function in the form

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i=1}^{n} T_{ij}v_i v_j - \sum_{i=1}^{n} b_i v_i - \sum_{i=1}^{n}\frac{1}{r_i}\int_0^{v_i} g_i^{-1}(v)dv \quad (3)$$

where $b_i = d_i/C_{ii}$ is the externally applied stimulation, $v_i = z_i = g_i(u_i) = u_i$ is the neuron output, u_i is the neuron state, $g_i(u_i)$ is the neuron activation functions and

$$T_{ij} = T_{ji} = \begin{cases} -C_{ij}/C_{ii} & i \neq j \\ 0 & i = j \end{cases} \quad (4)$$

$(i, j = 1, 2, \ldots, n)$ are the synaptic weights. The neural network has been designed in such a way, that after training, the outputs of the network to be the roots of the linear system under consideration. This result has been theoretically established by proving that the Lyapunov function of the proposed neural network is minimized by the roots of the given linear system.

4 Theory of algebraic nonlinear systems

According to the basic principles of the nonlinear algebra [12], 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

$$A_i(\vec{z}) = \sum_{j_1, j_2, \ldots, j_{s_i}}^{n} A_i^{j_1 j_2 \cdots j_{s_i}} z_{j_1} z_{j_2} \ldots z_{j_{s_i}} = 0 \quad (5)$$

$(i = 1, 2, \ldots, n)$, and it has one non-vanishing solution (i.e. at least one $j_j \neq 0$) if and only if the equation $\Re_{s_1, s_2, \ldots s_n}\{A_i^{j_1 j_2 \cdots j_{s_i}}\} = 0$ 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

$$d_{s_1, s_2, \ldots, s_n} = deg_A \Re_{s_1, s_2, \ldots, s_n} = \sum_{i=1}^{n}\left(\prod_{j\neq i} s_j\right) \quad (6)$$

When all degrees coincide, i.e. $s_1 = s_2 = \cdots = s_n = s$, the resultant $\Re_{n|s}$ is reduced to a polynomial of degree $d_{n|s} = deg_A \Re_{n|s} = n s^{n-1}$ and it is described completely by the values of the parameters n and s.

To understand the mathematical description of a complete nonlinear algebraic system let us consider the case of $n = s = 2$. By applying the defining equation, the i_{th} equation $(i = 1, 2, \ldots, n)$ of the nonlinear system is estimated as

$$\begin{aligned} A_i(\vec{z}) &= \sum_{j_1=1}^{2}\sum_{j_2=1}^{2} A_i^{j_1 j_2} z_{j_1} z_{j_2} = \\ &= A_i^{11} z_1^2 + (A_i^{12} + A_i^{21}) z_1 z_2 + A_i^{22} z_2^2 \end{aligned} \quad (7)$$

where the analytical expansion has been omitted for the sake of brevity. The same equation for the case $n = 3$ and $s = 3$ gets the form

$$\begin{aligned} A_i(\vec{z}) =& \sum_{j_1=1}^{3}\sum_{j_2=1}^{3}\sum_{j_3=1}^{3} A_i^{j_1 j_2 j_3} z_{j_1} z_{j_2} z_{j_3} = \\ &= A_i^{111} z_1^3 + A_i^{222} z_2^3 + A_i^{333} z_3^3 + (A_i^{121} + A_i^{211} + \\ &+ A_i^{112}) z_1^2 z_2 + (A_i^{131} + A_i^{113} + A_i^{311}) z_1^2 z_3 + \\ &+ (A_i^{221} + A_i^{212} + A_i^{122}) z_1 z_2^2 + (A_i^{331} + A_i^{313} + \\ &+ A_i^{133}) z_1 z_3^2 + (A_i^{332} + A_i^{233} + A_i^{323}) z_2 z_3^2 + \\ &+ (A_i^{232} + A_i^{223} + A_i^{322}) z_2^2 z_3 + (A_i^{321} + A_i^{231} + \\ &+ A_i^{312} + A_i^{132} + A_i^{231} + A_i^{123}) z_1 z_2 z_3 \end{aligned} \quad (8)$$

From the above equations, it is clear 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. It can be proven that such a tensor has only $nM_{n|s}$ independent coefficients, with $M_{n|s} = (n + s - 1)|/(n - 1)!s!$.

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 $\Re$, satisfies the nonlinear Craemer rule

$$\Re_{s_1,s_2,\ldots,s_n}\{A^{(k)}(Z_k)\}=0 \quad (9)$$

where Z_k is the k_{th} component of the solution of the non-homogenous system, and $A^{(k)}$ the k_{th} column of the coefficient matrix, A.

Since in the next sections neural models for solving the complete 2×2 and 3×3 nonlinear algebraic systems are proposed, let us describe their basic features for the simple case $s_1=s_2=s$. The complete 2×2 nonlinear system is defined as $A(x,y)=0$ and $B(x,y)=0$ where

$$A(x,y)=\sum_{k=0}^{s}\alpha_k x^k y^{s-k}=\alpha_s\prod_{j=1}^{s}(x-\lambda_j y)=y^s\tilde{A}(t) \quad (10)$$

$$B(x,y)=\sum_{k=0}^{s}\beta_k x^k y^{s-k}=\beta_s\prod_{j=1}^{s}(x-\mu_j y)=y^s\tilde{B}(t) \quad (11)$$

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

$$\Re=(\alpha_s\beta_s)^s\prod_{i,j=1}^{s}(\lambda_i-\mu_j)=(\alpha_0\beta_0)^s\prod_{i,j=1}^{s}\left(\frac{1}{\mu_j}-\frac{1}{\lambda_i}\right) \quad (12)$$

(for sake of simplicity we used the notation $\Re=\Re_{2|s}\{A,B\}$) and it can be expressed as the determinant of the $2s\times 2s$ matrix of coefficients. In the particular case of a linear map ($s=1$), this resultant reduces to the determinant of the 2×2 matrix, and therefore, it has the form $\Re_{2|1}\{A\}=\|\alpha_1;\alpha_0;\beta_1;\beta_0\|$. On the other hand, for $s=2$ (this is a case of interest in this project) the homogenous nonlinear algebraic system has the form

$$\begin{aligned}\alpha_{11}x^2+\alpha_{13}xy+\alpha_{12}y^2 &= 0 \quad (13)\\ \alpha_{21}x^2+\alpha_{23}xy+\alpha_{22}y^2 &= 0 \quad (14)\end{aligned}$$

with a resultant in the form

$$\Re_2=\Re_{2|2}\{A\}=\left\|\begin{array}{cccc}\alpha_{11}&\alpha_{13}&\alpha_{12}&0\\0&\alpha_{11}&\alpha_{13}&\alpha_{12}\\\alpha_{21}&\alpha_{23}&\alpha_{22}&0\\0&\alpha_{21}&\alpha_{23}&\alpha_{22}\end{array}\right\| \quad (15)$$

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

$$\alpha_{11}x^2+\alpha_{13}xy+\alpha_{12}y^2=-\alpha_{14}x-\alpha_{15}y+\beta_1 \quad (16)$$

$$\alpha_{21}x^2+\alpha_{23}xy+\alpha_{22}y^2=-\alpha_{24}x-\alpha_{25}y+\beta_2 \quad (17)$$

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

$$(\alpha_{11}X^2+\alpha_{14}X-\beta_1)z^2+(\alpha_{13}X+\alpha_{15})yz+\alpha_{12}y^2=0 \quad (18)$$

$$(\alpha_{21}X^2+\alpha_{24}X-\beta_2)z^2+(\alpha_{23}X+\alpha_{25})yz+\alpha_{22}y^2=0 \quad (19)$$

and

$$\alpha_{11}x^2+(\alpha_{13}Y+\alpha_{14})xz+(\alpha_{12}Y^2+\alpha_{15}Y-\beta_1)z^2=0 \quad (20)$$

$$\alpha_{21}x^2+(\alpha_{23}Y+\alpha_{24})xz+(\alpha_{22}Y^2+\alpha_{25}Y-\beta_1)z^2=0 \quad (21)$$

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

$$\left\|\begin{array}{l}\alpha_{11}X^2+\alpha_{14}X-\beta_1;\alpha_{13}X+\alpha_{15};\alpha_{12};0\\0;\alpha_{11}X^2+\alpha_{14}X-\beta_1;\alpha_{13}X+\alpha_{15};\alpha_{12}\\\alpha_{21}X^2+\alpha_{24}X-\beta_2;\alpha_{23}X+\alpha_{25};\alpha_{22};0\\0;\alpha_{21}X^2+\alpha_{24}X-\beta_2;\alpha_{23}X+\alpha_{25};\alpha_{22}\end{array}\right\|=0 \quad (22)$$

while, the Y variable satisfies the equation

$$\left\|\begin{array}{l}\alpha_{11};\alpha_{13}Y+\alpha_{14};\alpha_{12}Y^2+\alpha_{15}Y-\beta_1;0\\0;\alpha_{11};\alpha_{13}Y+\alpha_{14};\alpha_{12}Y^2+\alpha_{15}Y-\beta_1\\\alpha_{21};\alpha_{23}Y+\alpha_{24};\alpha_{22}Y^2+\alpha_{25}Y-\beta_2;0\\0;\alpha_{21};\alpha_{23}Y+\alpha_{24};\alpha_{22}Y^2+\alpha_{25}Y-\beta_1\end{array}\right\|=0 \quad (23)$$

(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 4^{th} 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 of the values of the coefficients of the matrix A.

5 ANNs as nonlinear system solvers

The extension of the structure of the previously described neural network to work with the nonlinear case is straightforward; however a few modifications are necessary. The most important of them is the usage of multiplicative neuron types known as product units [10] in order to produce the nonlinear terms of the algebraic system (such as x^2, xy, x^2z) during training. The learning algorithm of the network is the back propagation algorithm: since there is no Lyapunov energy function to be minimized, the roots of

the nonlinear system are not estimated as the outputs of the neural network, but as the weights of the synapses that join the input neuron with the neurons of the first hidden layer. These weights are updated during training in order to give the desired roots. On the other hand, the weights of the remaining synapses are kept to fixed values. Some of them have a value equal to the unity and contribute to the generation of the linear and the nonlinear terms, while some others are set to the coefficients of the nonlinear system to be solved. The network neurons are joined in such as way, that the total input of the output neurons - whose number coincides with the number of the unknowns of the system - to be equal to the left hand part of the corresponding system equation. Regarding the input layer, it has only one neuron whose input has a always a fixed value equal to the unity. The structure of the neural network solver for the complete 2×2 and 3×3 nonlinear systems are presented in the next sections.

5.1 The complete 2×2 nonlinear system

The complete nonlinear system with two equations and two unknowns has the general form

$$\alpha_{11}x^2 + \alpha_{12}y^2 + \alpha_{13}xy + \alpha_{14}x + \alpha_{15}y = \beta_1 \quad (24)$$
$$\alpha_{21}x^2 + \alpha_{22}y^2 + \alpha_{23}xy + \alpha_{24}x + \alpha_{25}y = \beta_2 \quad (25)$$

and the structure of the neural network that solves it, is shown in Figure 1. From Figure 1 it is clear that the neural network is composed of eight neurons grouped in four layers as follows: Neuron N_1 belongs to the input layer, neurons N_2 and N_3 belong to the first hidden layer, neurons N_4, N_5 and N_6 belong to the second hidden layer, while, neurons N_7 and N_8 belong to the output layer. From these neurons, the input neuron N_1 gets a fixed input with a value equal to the unity. The activation function of all the network neurons is the identity function, meaning that each neuron copies its input to its output without modifying it - this property in mathematical form is expressed as $I_k = O_k$ $(k = 1, 2, \ldots, 8)$ where I_k and O_k is the total input and the total output of the k_{th} neuron, respectively. Regarding the synaptic weights they are denoted as $W_{ij} \equiv W(N_i \rightarrow N_j)$ - in other words, W_{ij} is the weight of synapse that joins the source neuron N_i with the target neuron N_j - and they are assigned as follows:

$$W_{12} = x, \quad W_{13} = y$$

These weights are updated with the back propagation algorithm and after the termination of the training operation they keep the values (x, y) of one of the roots of the non-linear algebraic system.

$$W_{25}^{\alpha} = W_{25}^{\beta} = W_{24} = W_{34} = W_{36}^{\alpha} = W_{36}^{\beta} = 1$$

These weights have a fixed value equal to the unity; the role of the corresponding synapses is simply to supply to the multiplicative neurons N_5, N_4 and N_6 the current values of x and y, to form the quantities x^2, xy, and y^2, respectively. In the above notation, the superscripts α and β are used to distinguish between the two synapses that join the neuron N_2 with the neuron N_5 as well as the neuron N_3 with the neuron N_6.

$$W_{57} = \alpha_{11}, W_{67} = \alpha_{12}, W_{47} = \alpha_{13}, W_{27} = \alpha_{14}, W_{37} = \alpha_{15}$$
$$W_{58} = \alpha_{21}, W_{68} = \alpha_{22}, W_{48} = \alpha_{23}, W_{28} = \alpha_{24}, W_{38} = \alpha_{25}$$

The values of the above weights are fixed, and equal to the constant coefficients of the nonlinear system.

One of the remarkable features of the above network structure is the use of the product units N_5, N_4 and N_6, whose total input is not estimated as the sum of the individual inputs received from the input neurons, but as the product of those inputs. If we denote with I_s and I_p the total net input of a summation and a product unit respectively, each one connected to N input neurons, then, if this neuron is the i_{th} neuron of some layer, the above total input is estimated as $I_a = \sum_{j=1}^{N} W_{ij}x_j$ and $I_m = \prod_{j=1}^{N} x_j^{W_{ij}}$ where W_{ij} is the weight of the synapse joining the j_{th} input neuron with the current neuron, and x_j is the input coming from that neuron. It can be proven [10] that if the neural network is trained by using the gradient descend algorithm, then, the weight update equation for the input synapse connecting the j_{th} input unit with the ℓ_{th} hidden product unit has the form

$$\begin{aligned} w_{\ell j}^{h}(t+1) &= w_{\ell h}^{h}(t) + \eta f_l^{h'}(net_{p\ell}^{h}) e^{\rho_{p\ell}} \times \\ &\times \left(\ln|x_{jp}| \cos(\pi\xi_{p\ell}) - \pi I_{\ell}^{p} \sin(\pi\xi_{p\ell}) \right) \times \\ &\times \sum_{k=1}^{M} (d_{pk} - o_{pk}) f_k^{o'}(net_{pk}^{o}) w_{k\ell}^{o} \end{aligned} \quad (26)$$

where $net_{p\ell}^{h}$ is the total net input of the product unit and for the p_{th} training pattern, x_{jp} is the j_{th} component of that pattern, net_{pk}^{o} is the total net input of the k_{th} output neuron, d_{pk} and o_{pk} are the k_{th} component of the desired and the real output vector respectively, M is the number of output neurons, I_{ℓ}^{p} has a value of 0 or 1 according to the sign of x_{jp}, n is the learning rate value, while, the quantities $\rho_{p\ell}$ and $\xi_{p\ell}$ are defined as

$$\rho_{pm} = \sum_{j=1}^{N} w_{mj}^{h} \ln|x_{jp}| \quad \text{and} \quad \xi_{pm} = \sum_{j=1}^{N} w_{mj}^{h} I_m^p$$

respectively. In the adopted non linear model, all the threshold values are set to zero; furthermore, since the activation function of all neurons is the identity function, it is clear

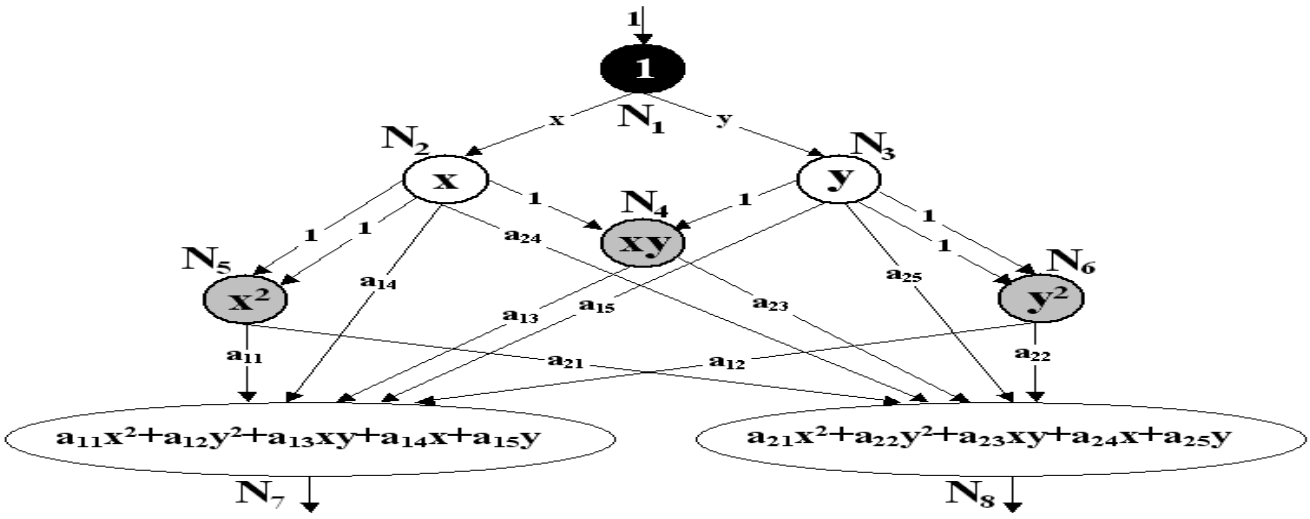

Figure 1. The structure of the 2×2 nonlinear system solver

that the total input calculated above is also the output that this neuron sends to its output processing units.

The neural network model presented so far, has been designed in such a way that the total input of the output neuron N_7 to be the expression $\alpha_{11}x^2 + \alpha_{12}y^2 + \alpha_{13}xy + \alpha_{14}x + \alpha_{15}y$ and the total input of the output neuron N_8 to be the expression $\alpha_{21}x^2 + \alpha_{22}y^2 + \alpha_{23}xy + \alpha_{24}x + \alpha_{25}y$. To understand this, let us identify all the paths that start from the input neuron N_1 and terminate to the output neuron N_7, as well as the input value associated with them. These paths are the following:

- Path P_1: it is defined as $N_1 \rightarrow N_2 \rightarrow N_5 \rightarrow N_7$. In this path, the neuron N_2 gets a total input $I_2 = W_{12} \cdot O_1 = 1 \cdot x = x$ and forwards it to the neuron N_5. There are two synapses between N_2 and N_5 with a fixed weight equal to the unity; since N_5 is a multiplicative neuron, its total input is estimated as $I_5 = (W^{\alpha}_{25}O_2) \cdot (W^{\beta}_{25}O_2) = (1 \cdot x) \cdot (1 \cdot x) = x^2$. Furthermore, since neuron N_5 works with the identity function, its output $O_5 = I_5 = x^2$ is sent to the output neuron N_7 multiplied by the weight $W_{57} = \alpha_{11}$. Therefore, the input to N_7 emerged from the path P_1 is $\xi^7_1 = \alpha_{11}x^2$.

- Path P_2: it is defined as $N_1 \rightarrow N_3 \rightarrow N_6 \rightarrow N_7$. Working in a similar way, the output of N_3 is $O_3 = I_3 = W_{13}O_1 = y$, the output of N_6 is $O_6 = I_6 = (W^{\alpha}_{36}O_3) \cdot (W^{\beta}_{36}O_3) = y^2$, and the total input to N_7 from the path P_2 is $\xi^7_2 = \alpha_{12}y^2$.

- Path P_3: it is defined as $N_1 \rightarrow N_2 \rightarrow N_4 \rightarrow N_7$. In this case the total input of the multiplicative neuron N_4 is equal to $I_4 = O_4 = W_{24}O_2 + W_{34}O_3 = xy$ and therefore, the contribution of the path P_3 to the total input of the neuron N_7 is equal to $\xi^7_3 = \alpha_{13}xy$.

- Path P_4: it is defined as $N_1 \rightarrow N_2 \rightarrow N_7$ and contributes to N_7 an input of $\xi^7_4 = \alpha_{14}x$.

- Path P_5: it is defined as $N_1 \rightarrow N_3 \rightarrow N_7$ and contributes to N_7 an input of $\xi^7_5 = \alpha_{15}y$.

Since N_7 is a simple additive neuron, its total input received from the five paths defined above is equal to

$$I_7 = \sum_{i=1}^{7} \xi^7_i = \alpha_{11}x^2 + \alpha_{12}y^2 + \alpha_{13}xy + \alpha_{14}x + \alpha_{15}y \quad (27)$$

Working in the same way, we can prove that the total input send to the output neuron N_8 is equal to

$$I_8 = \sum_{i=1}^{8} \xi^8_i = \alpha_{21}x^2 + \alpha_{22}y^2 + \alpha_{23}xy + \alpha_{24}x + \alpha_{25}y \quad (28)$$

Therefore, if the back propagation algorithm will be used with the values of β_1 and β_2 as desired outputs, the weights W_{12} and W_{13} will be configured during training in such a way, that when the trained network gets as input the unity, the network output will be the coefficients of the second part of the non-linear system. But this property means that the values of the weights W_{12} and W_{13} will be one of the roots (x, y) of the nonlinear system - which of the roots is actually the estimated root is something that requires further investigation.

5.2 The complete 3×3 nonlinear system

The complete 3×3 nonlinear system is given by the equations

$$\begin{aligned}\beta_i = &\alpha_{i01}x^3+\alpha_{i02}y^3+\alpha_{i03}z^3+\alpha_{i04}x^2y+\alpha_{i05}xy^2+\\ &+\alpha_{i06}x^2z+\alpha_{i07}xz^2+\alpha_{i08}y^2z+\alpha_{i09}yz^2+\alpha_{i10}xyz+\\ &+\alpha_{i11}xy+\alpha_{i12}xz+\alpha_{i13}yz+\alpha_{i14}x^2+\alpha_{i15}y^2+\\ &+\alpha_{i16}z^2+\alpha_{i17}x+\alpha_{i18}y+\alpha_{i19}z \end{aligned} \quad (29)$$

$(i = 1, 2, 3)$ and the structure of the neural network used for solving it, is shown in Figure 2. In fact, all the gray-colored neurons representing product units belong to the same layer (their specific arrangement in Figure 2 is only for viewing purposes) and therefore the proposed neural network structure consists of twenty three neurons grouped to four different layers with the neuron activation function to be again the identity function. Due to the great complexity of this network it is not possible to label each synapse of the net as in Figure 1. However, the assignment of the synaptic weights follows the same approach. Therefore, after training, the components (x, y, z) of the identified root are the weights $W_{12} = x$, $W_{13} = y$ and $W_{14} = z$. The weights of all the input synapses to all the product units are set to the fixed value of unity to generate the non linear terms of the algebraic system, while, the weights of the synapses connecting the hidden with the three output neurons are set as follows:

$$\begin{array}{lll}
W_{12,21} = \alpha_{101} & W_{12,22} = \alpha_{201} & W_{12,23} = \alpha_{301} \\
W_{16,21} = \alpha_{102} & W_{16,22} = \alpha_{202} & W_{16,23} = \alpha_{302} \\
W_{20,21} = \alpha_{103} & W_{20,22} = \alpha_{203} & W_{20,23} = \alpha_{303} \\
W_{13,21} = \alpha_{104} & W_{13,22} = \alpha_{204} & W_{13,23} = \alpha_{304} \\
W_{14,21} = \alpha_{105} & W_{14,22} = \alpha_{205} & W_{14,23} = \alpha_{305} \\
W_{15,21} = \alpha_{106} & W_{15,22} = \alpha_{206} & W_{15,23} = \alpha_{306} \\
W_{17,21} = \alpha_{107} & W_{17,22} = \alpha_{207} & W_{17,23} = \alpha_{307} \\
W_{18,21} = \alpha_{108} & W_{18,22} = \alpha_{208} & W_{18,23} = \alpha_{308} \\
W_{19,21} = \alpha_{109} & W_{19,22} = \alpha_{209} & W_{19,23} = \alpha_{309} \\
W_{09,21} = \alpha_{110} & W_{09,22} = \alpha_{210} & W_{09,23} = \alpha_{310} \\
W_{05,21} = \alpha_{111} & W_{05,22} = \alpha_{211} & W_{05,23} = \alpha_{311} \\
W_{10,21} = \alpha_{112} & W_{10,22} = \alpha_{212} & W_{10,23} = \alpha_{312} \\
W_{06,21} = \alpha_{113} & W_{06,22} = \alpha_{213} & W_{06,23} = \alpha_{313} \\
W_{07,21} = \alpha_{114} & W_{07,22} = \alpha_{214} & W_{07,23} = \alpha_{314} \\
W_{11,21} = \alpha_{115} & W_{11,22} = \alpha_{215} & W_{11,23} = \alpha_{315} \\
W_{08,21} = \alpha_{116} & W_{08,22} = \alpha_{216} & W_{08,23} = \alpha_{316} \\
W_{02,21} = \alpha_{117} & W_{02,22} = \alpha_{217} & W_{02,23} = \alpha_{317} \\
W_{03,21} = \alpha_{118} & W_{03,22} = \alpha_{218} & W_{03,23} = \alpha_{318} \\
W_{04,21} = \alpha_{119} & W_{04,22} = \alpha_{219} & W_{04,23} = \alpha_{319}
\end{array}$$

The proposed neural network shown in Figure 2 has been designed in such a way, that the total input O_i sent to i_{th} neuron to be equal to the left hand part of the i_{th} equation of the nonlinear algebraic system ($i = 1, 2, 3$). This fact means that if the network trained by the back propagation algorithm and by using the constant coefficients of the nonlinear system β_1, β_2 and β_3 as the desired outputs, then, after training, the weights W_{12}, W_{13} and W_{14} will contain the values (x, y, z) of one of the roots of the nonlinear algebraic system.

5.3 The complete $n \times n$ nonlinear system

The common feature of the proposed neural network solvers presented in the previous sections, is their structure, characterized by the existence of four layers: (a) an input layer with only one neuron whose output synaptic weights will hold after training the components of one of the system roots, (b) a hidden layer of summation neurons that generate the linear terms x, y in the first case and x, y, z in the second case, (c) a hidden layer of product units that generate the nonlinear terms by multiplying the linear quantities received from the previous layer, and (d) an output layer of summation neurons, that generate the left hand parts of the system equations and estimate the training error by comparing their outputs to the values of the associated fixed coefficients of the nonlinear system under consideration. It has to be mentioned however that even though this structure has been designed to reproduce the algebraic system currently used, it is not nessecarily the optimum one and the possible optimization of it, is a subject of future research. In this project the focus is given to the complete nonlinear system that contains not only the terms with a degree of s but also all the smaller degrees with a value of m ($1 \leq m \leq s$).

The previously described structure, characterizes also the neural solver for the complete $n \times n$ nonlinear algebraic system. Based on the fundamental theory of nonlinear algebraic systems presented in previous sections, it is not difficult to see that for a system of n equations with n unknowns characterized by a common degree $s_1 = s_2 = \cdots = s_n = s$, the total number of the linear as well as of the nonlinear terms of any degree m ($1 \leq m \leq s$) is equal to

$$N_{n|s} = \sum_{m=1}^{s} M_{n|m} = \frac{1}{(n-1)!} \sum_{m=1}^{s} \frac{(n+m-1)!}{m!} \quad (30)$$

Table 1 presents the number of those terms for the values $n = 1, 2, 3, 4, 5$ and for a degree $s = n$, a property that characterizes the nonlinear algebraic systems studied in this research. From this table, one can easily verify, that for values $n = 2$ and $n = 3$, the associated number of the system terms is equal to $N_{2|2} = 5$ and $N_{3|3} = 19$, in complete accordance with the number of terms of the analytic equations for the complete 2×2 and 3×3 systems, presented in the previous sections.

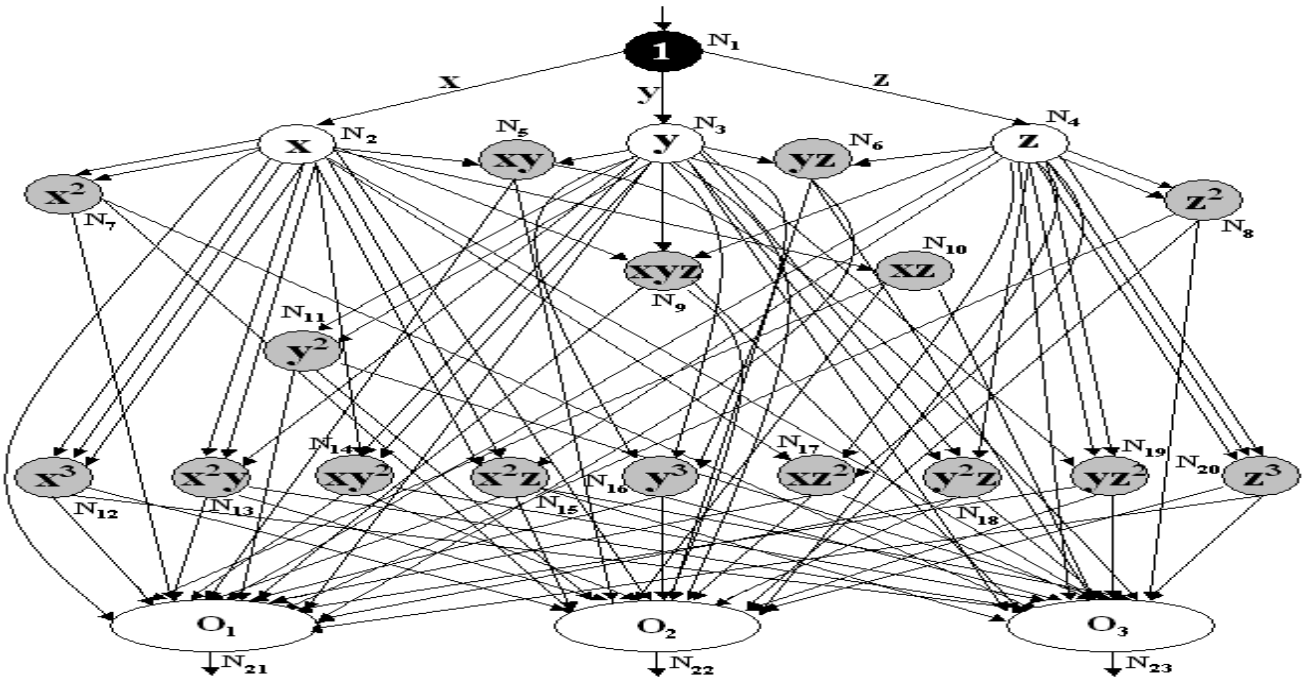

Figure 2. The structure of the 3×3 nonlinear system solver

For any given pair (n, s), the value of $N_{n|s}$ identifies the number of the neurons that belong to the two hidden layers of the neural solver. In fact, the number of product units belonging to the second hidden layer is equal to $P = N_{n|s} - n$, since, n of those neurons are the summation units of the first hidden layer that generate the linear terms $z_1, z_2, \ldots, z_n$. Regarding the total number of neurons in the network is is clearly equal to $N = N_{n|s} + n + 1$, since the input layer contains one neuron with a constant input equal to the unity, and the output layer contains n neurons corresponding to the n fixed parameters of the nonlinear system. Therefore, the structure of the neural simulator, contains the following layers: (a) an input layer with one summation unit, (b) a hidden layer of n summation units corresponding to the linear terms with a degree $i = 1$, (c) a hidden layer with $N_{n|s} - n$ product units corresponding to the nonlinear terms with a degree i $(2 \leq i \leq s)$, and (d) an output layer with n summation units.

On the other hand, the synaptic weights of the neural network are configured as follows: (a) the n synapses between the input neuron and the neurons of the first hidden layer have variable weights that after training will contain the components of the estimated root. (b) The synaptic weights between the summation units of the first hidden layer and the product units of the second hidden layer have a fixed value equal to the unity in order to contribute to the generation of the various nonlinear terms, and (c) the weights

Table 1. Number of terms of the complete nonlinear algebraic system for $n = 1, 2, 3, 4, 5$ and for the case $s = n$

System Dimension n=s	Total Number of Terms $N_{n\|s}$
1	001
2	005
3	019
4	069
5	251

of the synapses connecting the product units of the second hidden layer and the summation units of the output layer are set to the system coefficients, namely, to the components of the tensorial quantity, A. The same is true for the weights of the synapses connecting the summation units of the first hidden layer with the output neurons. This structure characterizes the complete nonlinear system, and if one or more terms are not present in the system to be solved, the corresponding synaptic weights is set to the zero value.

After the identification of the number of layers and the number of neurons per layer, let us know estimate the total number of synapses of the neural solver. Starting from the neurons of the output layer, each one of them, has of course $N_{n|s}$ input synapses, whose fixed weight values are set to the coefficients of the system tensor, A. It is not dif-

ficult to note, that n of those synapses come from the n summation units of the first hidden layer and are associated with the linear terms of the system with a degree of $m = 1$, while, the remaining $N_{n|s} - n$ input synapses, come from the product units of the second hidden layer and are associated with the nonlinear terms of some degree m $(2 \leq m \leq s)$. Since there are n output neurons, the total number of those synapses is equal to $L_o = n \cdot N_{n|s}$.

On the other hand, the number of synapses with a fixed weight value $w = 1$ connecting the summation units of the first hidden layer with the product units of the second hidden layer to form the various nonlinear terms, can be estimated as follows: as is has been mentioned, each product unit generates some non linear term of some degree m $(2 \leq m \leq s)$; this quantity is called from now, the degree of the product unit. Since each such unit gets inputs from summation units generating linear terms, it is clear, that a product unit with a degree of m, has m input synapses. Therefore, the number of input synapses for all the product units with a degree of m is equal to $L_{n|m} = m \cdot M_{n|m}$, while, the total number of input synapses for all the product units is given by the equation

$$L_{n|s} = \sum_{m=2}^{s} m \cdot M_{n|m} = \frac{1}{(n-1)!} \sum_{m=2}^{s} \frac{(n+m-1)!}{(m-1)!} \quad (31)$$

The third group of synapses that appear in the network, is the synapses with the variable weights connecting the input neuron with the n summation units of the first hidden layer. The number of those synapses is clearly, $L_n = n$. Therefore, the total number of synapses of the neural solver is equal to

$$\begin{aligned} L &= L_o + L_{n|s} + L_n = \\ &= n\left(\frac{1}{(n-1)!} \sum_{m=1}^{s} \frac{(n+m-1)!}{m!} + 1 \right) + \\ &\quad + \frac{1}{(n-1)!} \sum_{m=2}^{s} \frac{(n+m-1)!}{(m-1)!} \end{aligned} \quad (32)$$

Table 2 presents the values of the quantities L_o, $L_{n|s}$, L_n and L for $n = 1, 2, 3, 4, 5$ and for a degree $s = n$.

5.4 Creating the neural solver

After the description of the structure that characterizes the neural solver for the complete $n \times n$ nonlinear algebraic system, let us now describe the algorithm that creates its structure. The algorithm is given in pseudo-code format, and the following conventions are used:

- The network synapses are supposed to be created by the function AddSynapse with a prototype in the form

Table 2. Number of synapses L_o, $L_{n|s}$, L_n and L of the $n \times n$ neural solver for $n = 1, 2, 3, 4, 5$ and for the case $s = n$

$n = s$	L_o	$L_{n\|s}$	L_n	L
2	0010	0006	0002	0018
3	0057	0042	0003	0102
4	0276	0220	0004	0500
5	1255	1045	0005	2305

  AddSynapse (N_{ij},N_{kl},Fixed/Variable,N/rand). This fuction creates a synapse between the j_{th} neuron of the i_{th} layer and the l_{th} neuron of the k_{th} layer with a fixed weight equal to N or a variable random weight.

- The symbol L_i $(i = 1, 2, 3, 4)$ describes the i_{th} layer of the neural network.

The algorithm that creates the complete $n \times n$ neural solver is composed by the following steps:

1. Read the values of the parameters n and s and the coefficients of the tensor A.
2. Set $N_1 = 1$, $N_2 = n$, $N_3 = N_{n|s} - n$ and $N_4 = n$
3. Create the network layers as follows:
   - Create Layer L_1 with N_1 summation units
   - Create Layer L_2 with N_2 summation units
   - Create Layer L_3 with N_3 product units
   - Create Layer L_4 with N_4 summation units
4. Create the L_n variable-weight synapses between the input and the first hidden layer as follows:

```
for i=1 to n step 1
  AddSynapse (N11,N2i,variable,rand)
```

5. Create the $L_{n|s} - n$ input synapses to the $M_{n|s}$ product units of the second hidden layer with a fixed weight value $w = 1$ as follows:

   Each nonlinear term with some degree of m $(2 \leq m \leq s)$ generated by some product unit N_{3u} $(1 \leq u \leq M_{n|s})$ is the product of m linear terms, some of them are equal to each other; for those terms the corresponding product unit gets more than one synapses from the summation unit of the first hidden layer associated with that terms. Therefore, to draw the correct number of synapses, one needs to store for each nonlinear term, the position of the neurons of the first hidden layer, that form it. To better understand this situation, let us consider the case $m = 5$, and let us

suppose that the product unit N_{3u} generates the term $x_1x_2^3x_4$. This term can be expressed as the product $x_1x_2x_2x_2x_4$, and therefore, to create it, we need one synapse from neuron N_{21}, three synapses from neuron N_{22}, and one synapse from neuron N_{24}. This can be done in three steps: (a) we store in a buffer of m cells the current values of the indices of the m nested loops used to generate an m degree nonlinear term as follows:

```
for j1=1 to n step 1
    for j2=1 to n step 1
    ......................
        for jm=1 to n step 1
            InitBuffer (buf,j1..,jm)
```

In the above code, the function InitBuffer it is supposed to store the current value of the indices of the m nested loops to the buffer 'buf' with a length of m cells. (b) we perform a histogram analysis by identifying all the different indices stored in the buffer, and by counting for each one of them the number of its occurrences. The result of this procedure is a two dimensional matrix C of r rows and 2 columns where r is the number of different indices found in the buffer. In the case of the previous example, the three rows of the matrix C are the $\{1,1\}$, $\{2,3\}$, $\{4,1\}$, meaning that the term x_1 should be used one time, the term x_2 should be used three times, and the term x_4 should be used one time. But this means that we need one synapse from neuron N_{21}, three synapses from neuron N_{23}, and one synapse from neuron N_{24}. (c) In the last step we create the actual synapses by running the following code segment:

```
for i=1 to r step 1
    k=A[i][1]
    s=A[i][2]
    for j=1 to s step 1
        AddSynapse(L1k,L2u,fixed,1)
```

In the above pseudo-code the loop indices are varied from 1 to n and the two dimensional matrices are supposed to be stored in a row-wise fashion as in C programming language.

To create all the synapses with a fixed weight value $w = 1$, this procedure has to be applied for each product unit of the second hidden layer and for each degree m $(2 \leq m \leq s)$. Since during this procedure repeated terms are going to be generated (as for example the terms xy and yx), the synapse has to be created only if it does not exists (if the synapse exists it is characterized as non valid; this existence means that the nonlinear term under consideration has already been generated and the corresponding synapses have been created and assigned to it during a past cycle of the loop).

At this point, auxiliary data structures have to be allocated and initialized to help the creation of the synapses from the hidden to the output layer. For each degree m $(2 \leq m \leq s)$ of the nonlinear terms of the system, the number $N_{n|m}$ of the associated product units is estimated, and a two dimensional lookup table with dimensions $N_{n|m} \times (m+1)$ is allocated dynamically. Each time a new valid index combination is generated, it is stored to the next empty row of the lookup table together with the position of the associated product unit in the second hidden layer. In this way, we have the ability for a given nonlinear term to identify the product unit associated with it. The insertion procedure of a valid index combination to the system lookup table for parameter values $n = 3$ and $s = 6$ is shown in Figure 3. In Figure 3, the current content of the buffer are the indices values $j_1 = 1$, $j_2 = 1$, $j_3 = 2$, $j_4 = 3$, $j_5 = 3$ and $j_6 = 3$, and therefore, the nonlinear term generated by them is the term $x \cdot x \cdot y \cdot z \cdot z \cdot z = x^2yz^3$, where, for sake of simplicity, we have set $z_1 = x$, $z_2 = y$, and $z_3 = z$. Let us suppose also, that the product unit that is going to be associated with this index combination, is the fifth product unit of the layer L_3. Therefore, the record that is going to be added to the system lookup table, is the $\{5,1,1,2,3,3,3\}$. These records are stored from the first to the last row of the system lookup table in the order they appear. It is clear that the maximum number of those records for a given degree m $(2 \leq m \leq s)$ is equal to $N_{n|m}$.

6. Create the L_o input synapses to the output neurons whose fixed weights are set to the values of the system coefficients. These synapses can be divided into two groups: (a) synapses coming from the summation units of the first hidden layer and are associated with the linear terms with a degree $m = 1$, and (b) synapses coming from the product units of the second hidden layer and are associated with the nonlinear terms with degrees in the interval $[2, s]$. These synapse groups contain n^2 and $n(L_{n|s} - n)$ synapses, respectively.

   To describe the algorithm of creating those synapses let us write down the i_{th} equation of the complete nonlinear system $(i = 1, 2, \ldots, n)$ with the whole set of linear and nonlinear terms with degrees $1 \leq m \leq s$:

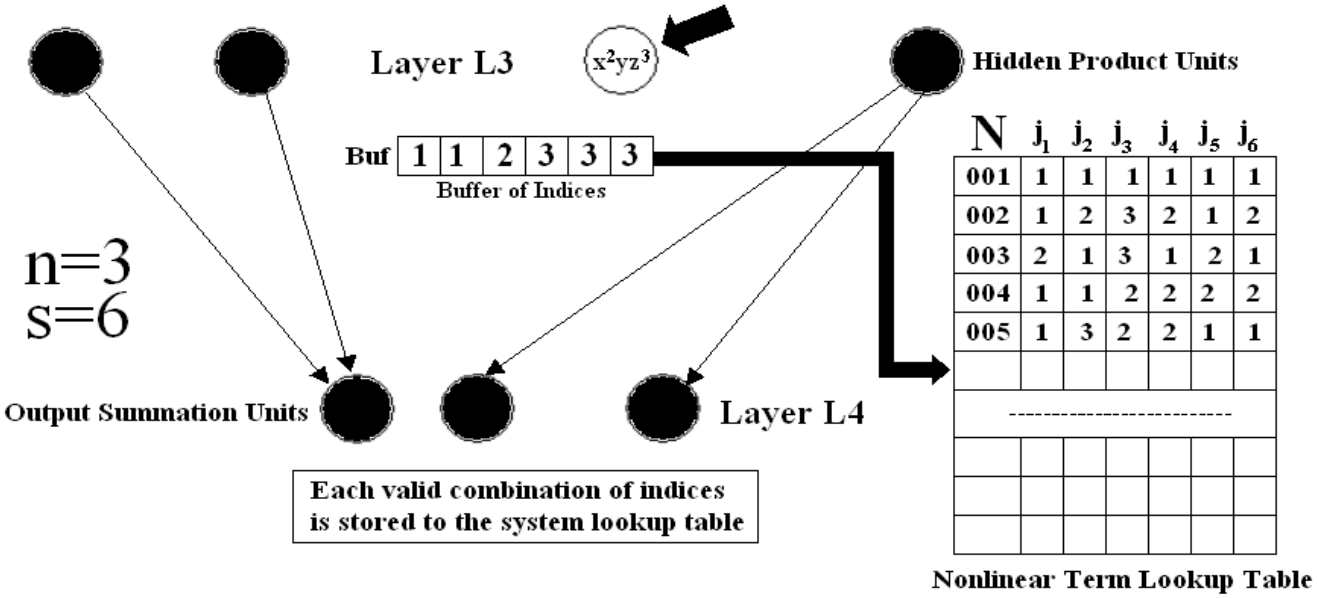

Figure 3. The insertion of a valid index combination to the system lookup table.

$$A_i(\vec{z}) = \sum_{j_1=1}^{n} A_{i(1)}^{j_1} z_{j_1} + \sum_{j_1=1}^{n}\sum_{j_2=1}^{n} A_{i(2)}^{j_1 j_2} z_{j_1} z_{j_2} + \\ + \sum_{j_1=1}^{n}\sum_{j_2=1}^{n}\sum_{j_3=1}^{n} A_{i(3)}^{j_1 j_2 j_3} z_{j_1} z_{j_2} z_{j_3} + \cdots + \\ + \sum_{j_1=1}^{n}\sum_{j_2=1}^{n}\cdots\sum_{j_s=1}^{n} A_{i(s)}^{j_1 j_2 j_3 \cdots j_s} z_{j_1} z_{j_2} z_{j_3} \cdots z_{j_s} \quad (33)$$

In this equation, the n terms of the first sum are the n linear terms of the system, while, the remaining terms are the nonlinear factors with degrees $2 \leq m \leq s$. From the above description it is clear that the complete identification of the system requires the initialization of a set of s tensors, $A_{(1)}, A_{(2)}, \ldots, A_{(m)}, \ldots, A_{(s)}$, with the tensor $A_{(m)}$ to have dimensions $n \times n \times \cdots \times n$ ($m+1$ times). From these tensors, the n^2 coefficients of the tensor $A_{(1)}$ will be assigned to the weights of the n^2 synapses connecting the n units of the layer L_2 with the n units of the layer L_4, while, the coefficients of the other tensors will be assigned to the $n(L_{n|s} - n)$ synaptic weights defined between the product units of the layer L_3 and the summation output units of the layer L_4.

Based on this discussion, the assignment of the coefficients of the tensor $A_{(1)}$ to the first group of synapses, will be performed as follows:

```
for i=1 to n step 1
    for j=1 to n step 1
        AddSynapse (L2i,L4j,
                fixed,A(1)[i][j])
```

On the other hand, the coefficients of the tensor $A_{(m)}$ for some degree m $(2 \leq m \leq s)$ are assigned to the weights of the synapses between the neurons of the layers L_3 and L_4 as follows:

```
for j1=1 to n step 1
  for j2=1 to n step 1
  ......................
    for jm=1 to n step 1 {
       InitBuffer (buf,j1..,jm)
       get k=lookup(buf)
       for i=1 to n step 1
         AddSynapse (L3k,L4i,fixed,
            A(m)[j1][j2]...[jm]) }
```

In other words, for each valid combination of the m indices, the buffer 'buf' is initialized and by traversing the system lookup table, we identify the product unit of the layer L_3, generating the corresponding nonlinear term. Then, synapses are created between this unit and each summation unit of the output neuron, with a fixed weight value $w = A_{(m)}^{j_1 j_2 \cdots j_m}$. This procedure has to be performed for all the product units and all the degrees m $(2 \leq m \leq s)$.

6 Experimental Results

The proposed neural solver was tested in practice for nonlinear systems with two and three unknowns. The training was based to the back propagation algorithm with a learning rate $n = 0.7$, a momentum $m = 0$ and without bias units. The average number of iterations for the network to converge was a few thousands iterations (5000-10000) and the global training error was equal to $10^{-7} - 10^{-8}$. The nonlinear systems solved by the network, and the training results are presented below:

- Nonlinear system 1: it is given by the equations

$$\begin{aligned}
&+0.20x^2+0.35y^2-0.1z^2+0.72xy-0.4xz- \\
&\quad -0.025yz+0.5x+0.68y-0.47z=0.130 \\
&-0.48x^2-0.33y^2+0.64z^2+0.08xy-0.01xz+ \\
&\quad +0.15yz+0.92x-0.27y+0.39z=0.450 \\
&-0.62x^2+0.43y^2-0.21z^2+0.07xy+0.12xz+ \\
&\quad -0.17yz-0.38x+0.94y+0.55z=0.940
\end{aligned}$$

  This system has eight roots in the form (x, y, z) whose values as estimated by Mathematica are the following:

$$\begin{aligned}
(x_1, y_1, z_1) &= (-1.12802, +2.07847, -3.20031) \\
(x_2, y_2, z_2) &= (-0.51219, -2.16791, +1.68004) \\
(x_3, y_3, z_3) &= (-2.10449, +2.67732, +2.96830) \\
(x_4, y_4, z_4) &= (+0.15248, +0.60549, +0.74952)
\end{aligned}$$

  These four roots are real, while, furthermore, the system has two double complex roots with values

$$\begin{aligned}
x_5 &= -0.236017 + 1.472720\, i \\
y_5 &= -1.857080 - 1.423440\, i \\
z_5 &= +0.036698 - 0.795817\, i
\end{aligned}$$

  and

$$\begin{aligned}
x_6 &= +0.333446 + 1.879550\, i \\
y_6 &= -0.896393 + 0.412340\, i \\
z_6 &= -0.751127 + 1.556630\, i
\end{aligned}$$

  The root of this system estimated by the neural solver was the $(x, y) = (+0.15248, +0.60549, +0.74952)$ and therefore the neural network reached the fourth root (x_4, y_4, z_4).

- Nonlinear system 2: it is given by the equations

$$\begin{aligned}
&0.1x^2 + 0.1y^2 + 0.3xy - 0.1x + 0.1y = +0.8 \\
&0.1x^2 - 0.3y^2 + 0.2xy - 0.1x - 0.3y = +0.4
\end{aligned}$$

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

$$\begin{aligned}
(x_1, y_1) &= (+12.23410, -03.82028) \\
(x_2, y_2) &= (-04.11479, +01.29870) \\
(x_3, y_3) &= (-01.40464, -01.07718) \\
(x_4, y_4) &= (+02.28537, +00.59876)
\end{aligned}$$

  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).

The main advantage of the proposed neural solver is its ability to solve practically any type of non linear system and not only systems with polynomial equations. These equations are reproduced by the network, since the activation function of the summation units of the first hidden layer is the identity function. However, this function can be any primitive or user defined function, allowing thus the construction of a nonlinear system with arbitrary structure. Supposing for example that the activation function of the L_1 units is the sinus function, the following nonlinear system can be solved:

$$\begin{aligned}
&0.1\sin^2 x + 0.1\sin^2 y + 0.3\sin x \sin y \\
&\qquad -0.1\sin x + 0.1\sin y = 0.126466 \\
&0.1\sin^2 x - 0.1\sin^2 y + 0.2\sin x \sin y \\
&\qquad -0.1\sin x - 0.1\sin y = 0.147451
\end{aligned}$$

The solution of this system estimated by the neural solver is the pair $(x, y) = (0.314162, 0.628318)$. On the other hand, Mathematica was not able to solve this system, a fact that demonstrates the power of the proposed method.

7 Conclusions and future work

The objective of this research was the implementation of nonlinear algebraic system solvers by using back-propagation neural networks. The proposed models are four-layered feed forward networks with summation and product units whose structure has been designed in such a way, that the total input to a summation unit of the output layer to be equal to the left-hand side of the corresponding equation of the nonlinear system. In this way, the network is trained to generate the constant coefficients of the system, with the components of the estimated root to be the weights of the synapses joining the neuron of the input layer with the n neurons of the first hidden layer, where n is the dimension of the non linear system.

There are many open problems regarding the accuracy and the performance of the proposed simulator, the most important of them are the following:

- If the nonlinear system under consideration can be solved, it has, in general, more than one roots. However, during training, the network estimates only one of them, and always the same root, even though the initial conditions are very close to other roots, different than the estimated one. Therefore, we need to investigate, why the network converges only to the specific root, and how the solver has to be configured in order to find and the other roots, too. Furthermore, the network structure and maybe the algorithm creating it, may need modifications to deal with complex system roots.

- The neural solver has to be tested for many different nonlinear system types - such as canonical, over-determined and under-determined systems as well as for systems with no solution - to examine the way it behaves and measure its performance. A required step is to measure the speed of the process and the execution time for continually increased system size $n \times n$ $(n = 1, 2, \ldots, N)$ for a specific value of the maximum size, N, in order to examine how the complexity increases with the problem size. Its computational complexity has to be estimated in a theoretical as well as in an experimental lever, and its performance has to be compared with the performance of other similar algorithms such that the gradient descent algorithm and the nonlinear ABS methods.

- The proposed neural network structure has been designed to deal with the simplest complete non linear algebraic systems with real coefficients, real roots and real constant terms. However, its extension for dealing with complex quantities is straightforward. For example, in the case of complex roots, one should have to separate the real from the imaginary parts, a procedure that leads to the duplication of the number of the equations. To understand this feature, let us consider the complete 2×2 nonlinear system described by equations $\alpha_{i1}x^2 + \alpha_{i2}y^2 + \alpha_{i3}xy + \alpha_{i4x} + \alpha_{i5}y = \beta_i$ $(i = 1, 2)$ and let us suppose that its roots (x, y), are complex numbers in the form $x = \kappa + i\lambda$ and $y = \mu + i\nu$. By substituting these roots in the system equations and by separating the real and the imaginary parts, one can easily get the system of four equations

$$\begin{aligned} \alpha_{i1}(\kappa^2-\lambda^2)+\alpha_{i2}(\mu^2-\nu^2)+ & \\ +\alpha_{i3}(\kappa\mu-\lambda\nu)+\alpha_{i4}\kappa+\alpha_{i5}\mu &= \beta_i \\ 2\alpha_{i1}\kappa\lambda+2\alpha_{i2}\mu\nu+\alpha_{i3}(\kappa\nu+\lambda\mu)+\alpha_{i4}\lambda+\alpha_{i5}\nu &= 0 \end{aligned}$$

  $(i = 1, 2)$ with a solution in the form $(\kappa, \lambda, \mu, \nu)$.

  By following the procedure described above, a neural network for solving this 4×4 nonlinear algebraic system can be constructed and used. Regarding the general case of the complete $n \times n$ nonlinear systems with complex roots - as well as complex coefficients and constant terms - the associated theoretical analysis is straightforward and it is subject of future research. The structure of neural network for the case of the complete 2×2 nonlinear systems with complex roots is shown in Figure 4. In this figure the weights of the synapses joining the linear units of the second layer with the product units of the third layer have a constant value equal to the unity, the weights of the four synapses between the input neuron and the linear units of the second layer are set as follows

$$W_{12} = \kappa \quad W_{13} = \lambda \quad W_{14} = \mu \quad W_{15} = \nu$$

  while, the weights of the synapses between the third and the fourth layer are initialized as

$$\begin{aligned} W_{06,16} &= +\alpha_{11} & W_{06,18} &= +\alpha_{21} \\ W_{08,16} &= -\alpha_{11} & W_{08,18} &= -\alpha_{21} \\ W_{12,16} &= +\alpha_{12} & W_{12,18} &= +\alpha_{22} \\ W_{15,16} &= -\alpha_{12} & W_{15,18} &= -\alpha_{22} \\ W_{09,16} &= +\alpha_{13} & W_{09,18} &= +\alpha_{23} \\ W_{10,16} &= -\alpha_{13} & W_{10,18} &= -\alpha_{23} \\ W_{02,16} &= +\alpha_{14} & W_{02,18} &= +\alpha_{24} \\ W_{04,16} &= +\alpha_{15} & W_{04,18} &= +\alpha_{25} \\ W_{07,17} &= 2\alpha_{11} & W_{07,19} &= 2\alpha_{21} \\ W_{14,17} &= 2\alpha_{12} & W_{14,19} &= 2\alpha_{22} \\ W_{13,17} &= +\alpha_{13} & W_{13,19} &= +\alpha_{23} \\ W_{11,17} &= +\alpha_{13} & W_{11,19} &= +\alpha_{23} \\ W_{03,17} &= +\alpha_{14} & W_{03,19} &= +\alpha_{24} \\ W_{05,17} &= +\alpha_{15} & W_{05,19} &= +\alpha_{25} \end{aligned}$$

- Finally, a last issue that it has to be resolved is the ability of the neural solver to deal with arbitrary nonlinear algebraic systems. This ability has been demonstrated for the sinus function, but the general behavior of the solver has not been studied systematically and it is a subject of future research.

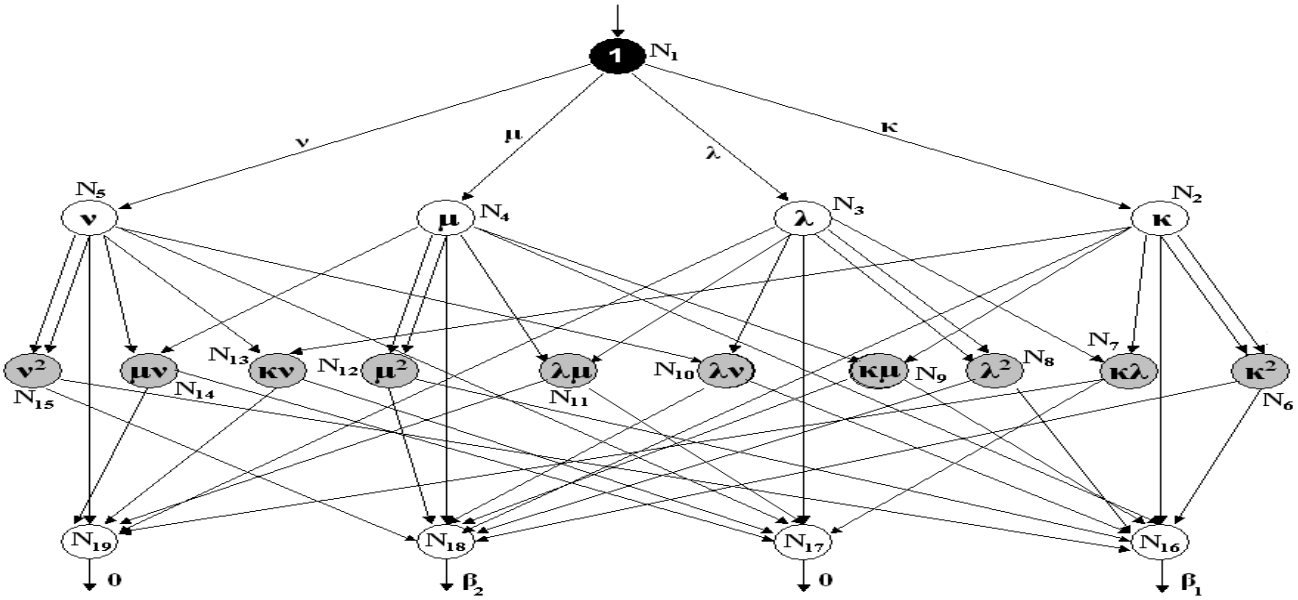

Figure 4. The structure of the 2×2 neural solver for the complete 2×2 nonlinear algebraic system with complex roots.

References

[1] W.Press, S.Teukolsky, W.Vetterling, B.Flannery, *Numerical Recipes in C - The Art of Scientific Programming*, Second Edition, Cambridge University Press, 1992.

[2] J.Abaffy, C.G.Broyden, and E.Spedicato, "A class of direct methods for linear systems", *Numerische Mathematik*, 1984, Vol. 45, pp. 361-376.

[3] J.Abaffy, and E.Spedicato, *ABS Projection Algorithms: Mathematical Techniques for Linear and Nonlinear Equations*, Ellis Horwood, Chichester, 1989.

[4] E.Spedicato, E.Bodon, A.Del Popolo, N. Mahdavi-Amiri, "ABS Methods and ABSPACK for Linear Systems and Optimization, a Review", *Proceedings of the* 3^{rd} *Seminar of Numerical Analysis*, Zahedan, November 15/17, 2000, University of Zahedan.

[5] J.Abaffy, and A.Galantai, "Conjugate Direction Methods for Linear and Nonlinear Systems of Algebraic Equations", Colloquia Mathematica Soc. Jfinos Bolyai, *Numerical Methods*, Miskolc, Hungary, 1986; Edited by P. R6zsa and D. Greenspan, North Holland, Amsterdam, Netherlands, 1987, Vol. 50, pp. 481-502.

[6] J.Abaffy, A.Galantai, and E.Spedicato, "The Local Convergence of ABS Methods for Nonlinear Algebraic Equations", 1987, *Numerische Mathematik*, Vol. 51, pp. 429-439.

[7] A.Galantai, and A.Jeney, "Quasi-Newton ABS Methods for Solving Nonlinear Algebraic Systems of Equations", *Journal of Optimization Theory and Applications*, 1996, Vol.89, No.3, pp. 561-573.

[8] V.N.Kublanovskaya, and V.N.Simonova, "An Approach to Solving Nonlinear Algebraic Systems. 2", *Journal of Mathematical Sciences*, 1996, Vol.79, No.3, pp. 1077-1092.

[9] I.Emiris, B.Mourrain, and M.Vrahatis, "Sign Methods for Counting and Computing Real Roots of Algebraic Systems", *Technical Report RR-3669*, 1999, INRIA, Sophia Antipolis.

[10] A.Engelbrecht, and A.Ismail, "Training product unit neural networks", *Stability and Control: Theory and Applications (SACTA)*, 1999, Vol 2, No 1-2, pp. 59-74, 1999.

[11] K.G.Margaritis K.G, M.Adamopoulos, K.Goulianas, "Solving linear systems by artificial neural network energy minimisation", 1993, *Universily of Macedonia Annals (volume XII)*, pp.502-525, (in Greek).

[12] V.Dolotin, and A.Morozov, "Introduction to Nonlinear Algebra v.2", online document found in the Arxiv repository (www.arxiv.org), ITEP, Moscow, Russia, September 2006.