*Appl. Math. and Comp. Sci., 1994, vol.4, No.3, 309–330*

# APPLICATIONS OF NEURAL-TYPE STRUCTURED NETWORKS FOR SOLVING ALGEBRAIC MATRIX EQUATIONS AND COMPUTATION OF THE DRAZIN INVERSE

ANDRZEJ CICHOCKI*, TADEUSZ KACZOREK**

An overview of the methods of solving algebraic matrix equations and computation of the Drazin inverse of a singular matrix with the use of neural-type structured networks is presented. Algebraic matrix equations of the form $A_1XB_1 + A_2XB_2 + \ldots + A_nXB_n = C$ and the algebraic Riccati equation $A^TX + XA - XWX + Q = 0$ with one unknown matrix $X$ and the matrix equations $AX - YB = C$ with two unknown matrices $X, Y$ are considered. An extension for polynomial matrix equations of the form $AXB = C$ is also presented. The presented algorithms are based on the gradient optimization technique and the standard back-propagation learning algorithm.

## 1. Introduction

Recently, there has been great interest in massively parallel computing for various linear algebra problems, specially those of high computational complexity (Charlier and Van Dooren, 1989; Cichocki and Unbehauen, 1992; 1993; Osowski, 1993; Polycarpon and Ioannou, 1992; Wang and Mendel, 1992). Modern scientific and engineering computing is searching for fast, efficient and robust algorithms which are to a large extent massively parallel algorithms in the sense that a system of non-linear ordinary differential or difference equations describing a specific learning algorithm is solved simultaneously by a suitable VLSI network consisting of highly interconnected processing units (artificial neurons) (Cichocki and Unbehauen, 1993; Distante *et al.*, 1991; Flaherty and Michell, 1990; Lillo *et al.*, 1993; Polycarpon and Ioannou, 1992; Wang and Mendel, 1992; Wang and Wu, 1993).

There is extensive ongoing research on neural networks and their representation on array processors. General multilayer networks with supervising learning algorithms which can be mapped relatively easily onto massively parallel digital architectures are especially promising (Distante *et al.*, 1991).

The main purpose of this paper is to show how a large variety of linear matrix algebra problems of high computational complexity can be solved in almost real-time

* Institute of Theory of Electrical Engineering and Measurements, Warsaw University of Technology, ul. Koszykowa 75, 00–662 Warsaw, Poland

** Institute of Control and Industrial Electronics, Warsaw University of Technology, ul. Koszykowa 75, 00–662 Warsaw, Poland

by using parallel structured networks. Parallel structured networks constitute a simplified class of linear feedforward neural networks, where the activation function is replaced by a simple identity function (Cichocki and Unbehauen, 1992; 1993; Wang and Mendel, 1992). In fact, structured networks can be defined as multilayer feedforward neural networks with linear neurons in which weights of neurons give the solution to the problem. Linear artificial neurons are often considered uninteresting processing units, mainly for two reasons: firstly, only linear functions can be computed in linear networks, and secondly, a linear multilayer network with several layers can formally be transformed to a network with only one layer of linear processing units by multiplying and summing the weight matrices of the corresponding layers. However, we should emphasize that multilayer linear structured networks are very useful in many computational tasks and are still of great interest due to the internal representation of the input data which are processed and linear transformation which occur in different layers during the learning process (Cichocki and Unbehauen, 1993).

Algebraic and polynomial matrix equations play an important role in various systems, control and filtering problems. Algebraic Riccati equations are extensively used in control and system theory, for example in the very recent $H_2$ or $H_\infty$ control, as well as the standard LQG optimal control and Kalman filtering (Charlier and Van Dooren, 1989; Gardiner and Laub, 1991; Kenney *et al.*, 1989; Kučera, 1972; Laub, 1979; Wei and Yeh, 1991).

## 2. Set of Linear Algebraic Equations

Let $\mathbb{R}^n$ be the set of $n$-dimensional real vectors and $\mathbb{R}^{m\times n}$ be the set of $m \times n$ dimensional real matrices. Consider the set of linear algebraic equations (Cichocki and Unbehauen, 1993)

$$Ax = b \tag{1}$$

where $A = [a_{ij}] \in \mathbb{R}^{m\times n}$, $b = [b_i] \in \mathbb{R}^m$ are given and $x = [x_i] \in \mathbb{R}^n$ is unknown.

The number of equations $m$ can be less than, equal to or greater than the number of variables (components of $x$) $n$. From a practical point of view it is usually required to find an approximate solution which comes as close as possible to the exact one subject to a suitable optimality criterion. In the ordinary least squares (LS) approach to the problem the measurements in the matrix $A$ are assumed to be free from error and all errors are confined to the observation vector $b$, $b_t = b + n_e$, where $n_e \in \mathbb{R}^n$ is an unknown vector of the measurement noise. In this case the problem can be stated as follows. Find a vector $x \in \mathbb{R}^n$ which minimizes the scalar function (optimality criterion)

$$E(x) \doteq \|r(x)\|_p, \quad p \geq 1, \quad r(x) \doteq Ax - b \tag{2}$$

where $\|r(x)\|_p$ denotes the $L_p$ norm of the vector $r(x) \doteq [r_1(x), \ldots, r_m(x)]^T$ and the upper index $T$ denotes the transposition. For $p = 1, 2, \infty$ we have

$$\|r(x)\|_1 \doteq \sum_{i=1}^{m} |r_i(x)|, \quad \|r(x)\|_2 \doteq \left[r^T(x)r(x)\right]^{\frac{1}{2}}, \quad \|r(x)\|_\infty \doteq \max_i |r_i(x)|$$

The minimal norm $L_p$ solution $x^*$ satisfies the equation

$$Ax^* = b + r(x^*) \tag{3}$$

and

$$||Ax^* - b||_p \leq ||Ax - b||_p \quad \text{for all } x \in \mathbb{R}^n \tag{4}$$

If $m = n$ and $\det A \neq 0$, then $x^* = A^{-1}b$ and $||r(x^*)||_p = 0$ for $p = 1, 2, \infty$.

For $p = 2$ eqn. (2) takes the form

$$E(x) \doteq \frac{1}{2}(Ax - b)^T(Ax - b) \tag{5}$$

and we have (Cichocki and Unbehauen, 1993)

1) the unique exact solution $x^* = A^{-1}b$ $\left(E(x^*) = 0\right)$ if $m = n$ and $\det A \neq 0$;

2) the approximate least squares solution

$$x^* = (A^TA)^{-1}A^Tb \quad \left(E(x^*) = b^T(I - AA^+)b \geq 0\right) \quad \text{if } \operatorname{rank} A = n < m$$

where $A^+$ is the Moore-Penrose pseudoinverse matrix;

3) the minimal norm $L_2$ solution, which is unique

$$x^* = A^T(AA^T)^{-1}b \qquad \left(E(x^*) = 0\right) \quad \text{if } \operatorname{rank} A = m < n.$$

Using the gradient method (Cichocki and Unbehauen, 1993) to minimize (5), we obtain

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\mu\nabla E(x), \quad \nabla E(x) \doteq A^T(Ax - b) \tag{6}$$

where $\mu = [\mu_{ij}]$ is an $n \times n$ positive definite matrix $\mu > 0$. If $\mu$ is positive definite, then (6) is stable since

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \left[\frac{\partial E}{\partial x}\right]\frac{\mathrm{d}x}{\mathrm{d}t} = -\left[\nabla E(x)\right]^T\mu\nabla E(x) \leq 0$$

From (2), (5) and (6) we have

$$r_i(x) = \sum_{k=1}^{n} a_{ik}x_k - b_i, \qquad i = 1, \ldots, m \tag{7}$$

$$\frac{\partial E(x)}{\partial x_j} = \sum_{k=1}^{m} a_{kj}r_k(x), \qquad j = 1, \ldots, n \tag{8}$$

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\sum_{k=1}^{n} \mu_{jk}\frac{\partial E(x)}{\partial x_k} \tag{9}$$

The linear network shown in Fig. 1 follows from (7)–(9). The network consists of integrators and adders with associated connection weights denoted by $a_{ij}$ and $\mu_{ij}$. It consists of three layers of artificial neurons. Each neuron produces its output by computing the inner product of its input signals and its appropriate weight vector and passing the result through a non-linear sigmoid function. The first layer computes the actual residual errors $r_i(x)$ defined by (7). In the second layer the gradient components (8) are computed. The third layer constitutes the proper adaptive system. The selection of appropriate weights $\mu_{ij}$ is crucial. A suitable choice of $\mu_{ij}$ should ensure an appropriate convergence speed to the equilibrium state of the network. The outputs $x_1, x_2, \ldots, x_n$ of the integrators are components of the desired solution $x$.

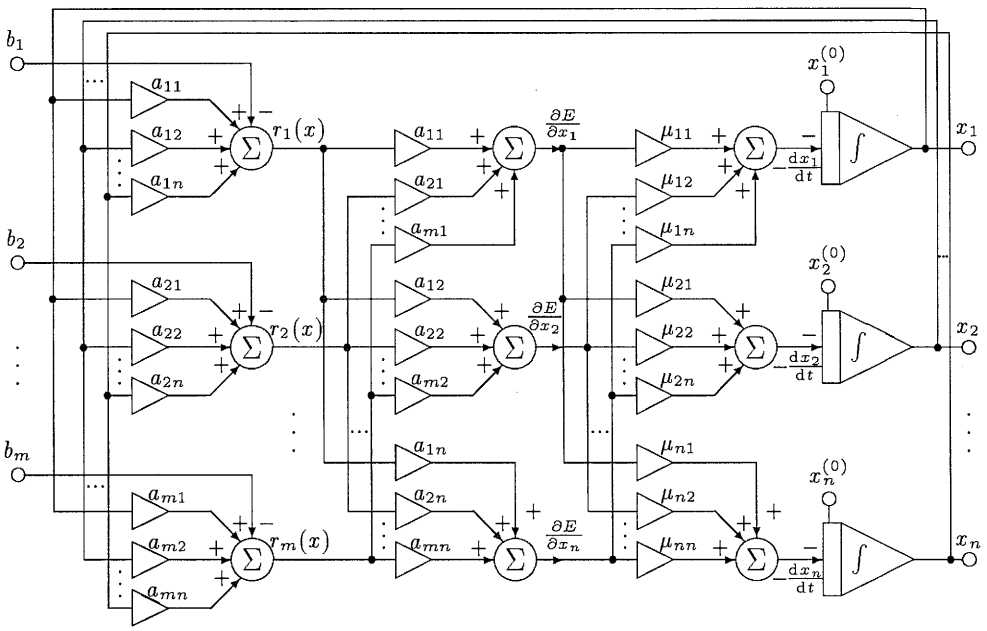


Fig. 1. General structure of an artificial neural network for solving a system of linear equations.

## 3. Algebraic Matrix Equations

### 3.1. Algebraic Equations with One Unknown Matrix

Consider the algebraic matrix equation (Cichocki and Kaczorek, 1992b)

$$\sum_{i=1}^{n} A_i X B_i = C \tag{10}$$

where $A_i \in \mathbb{R}^{m \times k}$, $B_i \in \mathbb{R}^{p \times q}$, $i = 1, \ldots, n$, $C \in \mathbb{R}^{m \times q}$ are given and $X = [x_{ij}] \in \mathbb{R}^{k \times p}$ is unknown.

For $n = 2$, $A_1 = A$, $B_1 = I$ and $A_2 = I$, $B_2 = B$ from (10) we have

$$AX + XB = C \tag{11}$$

and for $B = A^T$ the well-known Lyapunov equation. The equations

$$AX_1 + BX_2 + C \tag{12}$$

$$X_1A + X_2B + C \tag{13}$$

are particular cases of (10) for $A_1 = [A, B]$, $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$, $B_1 = I$ and $A_1 = I$, $X = [X_1, X_2]$, $B_1 = \begin{bmatrix} A \\ B \end{bmatrix}$, respectively.

It is assumed that equation (10) has a solution $X$. But equation (10) has a solution if and only if (Kaczorek, 1984)

$$\operatorname{rank}\left[\sum_{i=1}^{n} A_i \otimes B_i^T\right] = \operatorname{rank}\left[\sum_{i=1}^{n} A_i \otimes B_i^T, c\right]$$

where $\otimes$ denotes the Kronecker product and $c \in \mathbb{R}^{mq}$ is the vector consisting of the rows of $C$.

Postmultiplying (10) by a non-zero, time-variable excitation vector $u \doteq [u_1, \ldots, u_q]^T$ we obtain

$$\sum_{i=1}^{n} A_i X B_i u = Cu \tag{14}$$

Equation (14) can be represented by a multilayer neural network shown in Fig. 2. The layers corresponding to $A_i$, $B_i$, $i = 1, \ldots, n$ and $C$ represent neurons with fixed and known weights, while the layers corresponding to $X$ represent neurons with adjustable weights which must be determined during the optimization (learning) procedure. The network can be described by the equations

$$d_j = -\sum_{i=1}^{q} c_{ji} u_i, \quad e_j = \sum_{r=1}^{m} y_j^r + d_j, \quad y_j^r = \sum_{i=1}^{k} a_{ji}^r z_i^r$$

$$z_i^r = \sum_{j=1}^{p} x_{ij} \nu_j^r, \quad \nu_j^r = \sum_{i=1}^{q} b_{ji}^r u_i \tag{15}$$

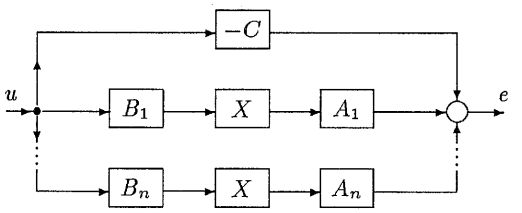


Fig. 2. A multilayer network for solving of the matrix eqn. (10).

where

$$A_r = [a_{ij}^r], \quad B_r = [b_{ij}^r], \quad X = [x_{ij}], \quad C = [c_{ij}], \qquad r = 1, \ldots, n$$

$$\nu^r = [\nu_1^r, \ldots, \nu_p^r]^T, \quad z^r = [z_1^r, \ldots, z_k^r]^T, \quad y^r = [y_1^r, \ldots, y_m^r]^T$$

As the error (cost) function in the optimization procedure we assume

$$E = \frac{1}{2} e^T e, \quad e \doteq \sum_{i=1}^{n} (A_i X B_i - C) u \tag{16}$$

To minimize (16) we shall use the gradient method (Cichocki and Unbehauen, 1993) based on the equation

$$\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} = -\mu \frac{\partial E}{\partial x_{ij}} \quad \text{for} \quad i = 1, \ldots, k \quad \text{and} \quad j = 1, \ldots, p \tag{17}$$

where $\mu > 0$ is the optimization (learning) rate.

Using the chain rule and taking into account eqns. (15) we obtain

$$\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} = -\mu \sum_{r=1}^{n} \Big( \sum_{h=1}^{m} a_{hi}^r e_h \Big) \nu_j^r \quad \text{for} \quad i = 1, \ldots, k \quad \text{and} \quad j = 1, \ldots, p \tag{18}$$

Differential eqns. (18) constitute the basic learning algorithm which can be considered as a modification of the back-propagation (delta rule) algorithm (Cichocki and Unbehauen, 1993). A realization of the algorithm is shown in Fig. 3. To check the validity and performance of the neural network it has been simulated extensively on a computer. Very good agreement with the theoretical considerations has been obtained.

**Example 1.** Find a solution $X$ of (10) for $n = 2$ with

$$A_1 = \begin{bmatrix} -4 & 3 & 1 \\ -1 & 4 & 5 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 2 & 0 & -2 \\ -2 & 3 & -4 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -1 & -2 & 4 \\ -2 & 2 & -1 \end{bmatrix},$$

$$B_2 = \begin{bmatrix} -3 & -3 & -2 \\ 4 & 2 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} -39 & 4 & 20 \\ 14 & -7 & -10 \end{bmatrix}$$

On the basis of the system of differential equations (18) and the network architecture of Fig. 2, an appropriate circuit has been designed and simulated. As excitation signals orthogonal (uncorrelated) sine waves of the form $u_1(t) = \sin \omega t$, $u_2(t) = \sin 2\omega t$, $u_3(t) = \sin 3\omega t$, with $\omega = 10^6$ rad/s, have been employed. The learning rate was chosen equal to $\mu = 10^5$ which means integration time constants of integrators $\tau = 1/\mu = 10^{-5}$ seconds. Representative computer simulated trajectories for zero initial conditions illustrating convergence behaviour of the network are shown in Fig. 4. The network was able to find the correct solution in time less than $10^{-4}$ seconds independently of initial conditions. The resulting matrix $X$ after $10^{-4}$

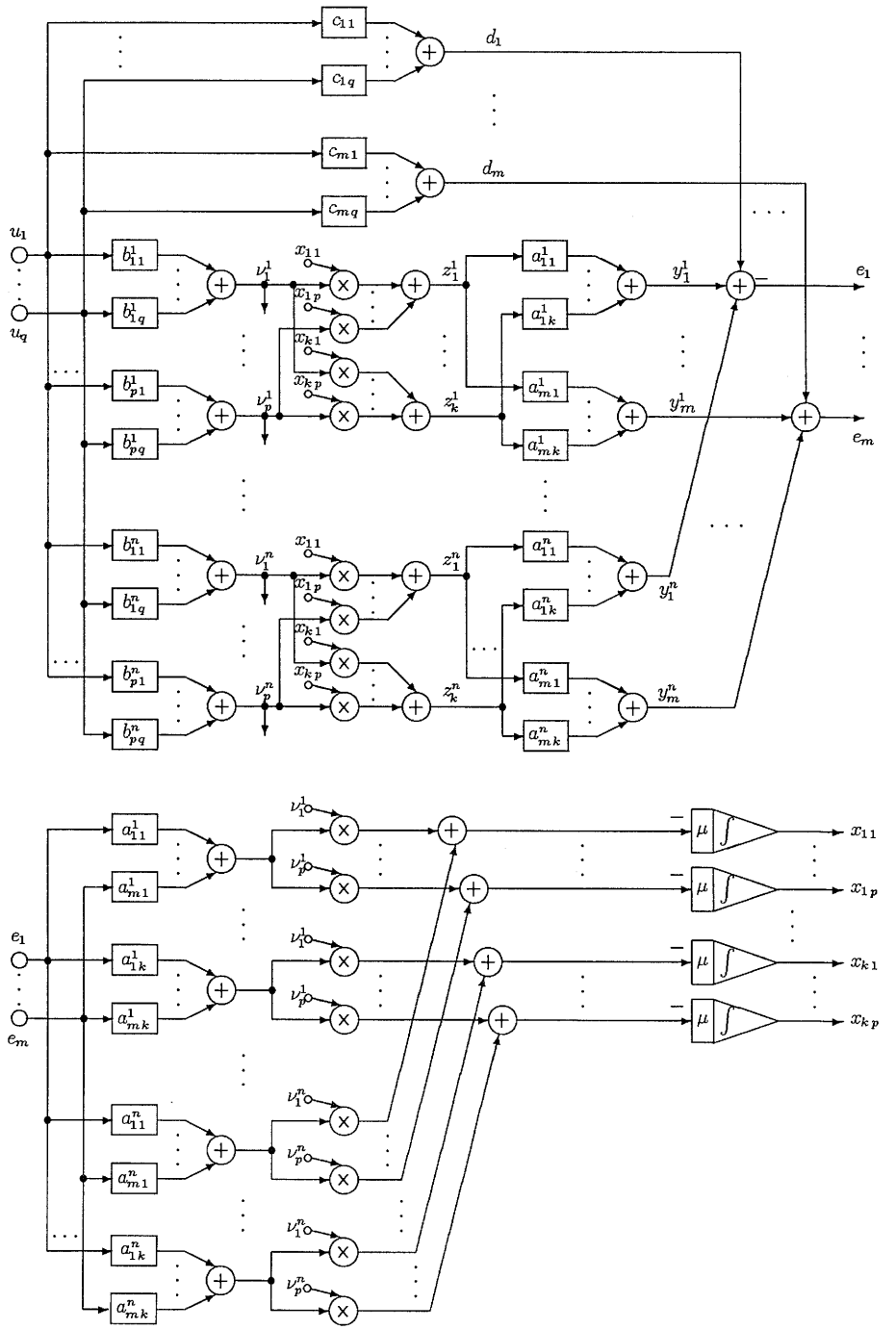


Fig. 3. A detailed architekture of the neural network for solving matrix eqn. (10).

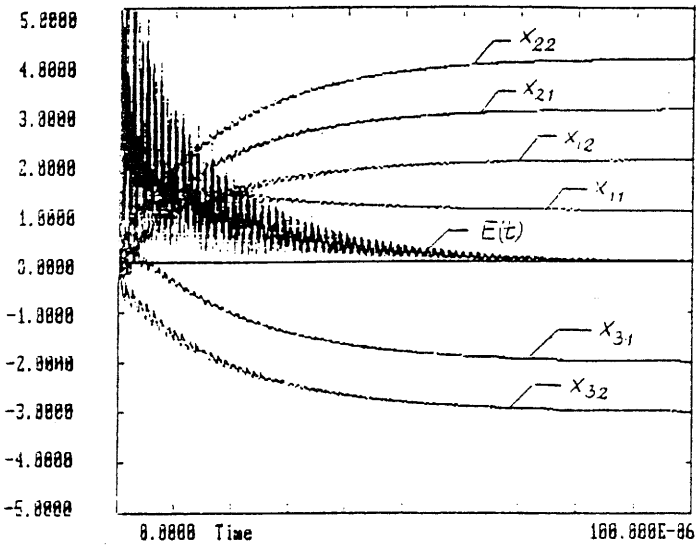


Fig. 4. Convergence behaviour of the neural network for Example 1.

seconds was

$$X = \begin{bmatrix} 1.0003 & 1.9991 \\ 2.9997 & 3.9992 \\ -1.9995 & -2.9994 \end{bmatrix}$$

which is in good agreement with the exact result

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ -2 & -3 \end{bmatrix}$$

The convergence speed of the network can be further increased by increasing the learning rate and the angular frequency.

### 3.2. Algebraic Riccati Equations

Consider an algebraic Riccati equation (Cichocki and Kaczorek, 1992a)

$$A^T X + XA - XWX + Q = 0 \tag{19}$$

where $A = [a_{ij}] \in \mathbb{R}^{n \times n}$, $W = [w_{ij}] \in \mathbb{R}^{n \times n}$, $Q = [q_{ij}] \in \mathbb{R}^{n \times n}$ are given and $X \in \mathbb{R}^{n \times n}$ is unknown.

It is assumed that $W$ and $Q$ are symmetric and positive semi-definite with $(A, W)$ stabilizable and $(Q, A)$ detectable. Under these assumptions eqn. (19) has a

unique symmetric positive semi- definite solution and the matrix $A - WX$ is stable (Laub, 1979). Postmultiplying (19) by a non-zero excitation vector $u = [u_1, \ldots, u_n]^T$ we obtain

$$A^T X u + X(A - WX)u + Qu = 0 \tag{20}$$

A multilayer branching neural network shown schematically in Fig. 5 follows from (20). Each block box represents one layer of linear neurons (processing units). The layers corresponding to $A, W$ and $Q$ represent neurons with fixed and known weights, while the layers corresponding to $X$ consist of neurons with adjustable weights which must be determined during the optimization process. As the error function in the optimization procedure we assume

$$E \doteq \frac{1}{2} e^T e, \quad e \doteq A^T X u + X(A - WX)u + Qu = [e_1, \ldots, e_n]^T \tag{21}$$

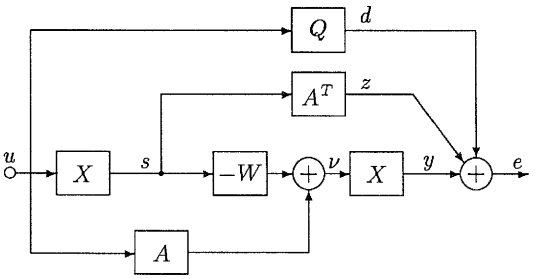


Fig. 5. Multilayer neural-type structured network for solving eqn. (20).

To minimize (21) we shall use the gradient method based on the equation

$$\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} = -\mu \frac{\partial E}{\partial x_{ij}} \quad \text{for} \quad i, j, \ldots, n \tag{22}$$

Using the chain rule we obtain

$$\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} = -\mu \left( e_i \nu_j + \frac{\partial E}{\partial s_i} u_j \right) \tag{23}$$

and

$$\frac{\partial E}{\partial s_i} = \sum_{p=1}^{n} \frac{\partial E}{\partial z_p} \frac{\partial z_p}{\partial s_i} = \sum_{p=1}^{n} e_p a_{ip} \tag{24}$$

where

$$\nu_i = \sum_{k=1}^{n} [a_{ik} u_k - w_{ik} s_k], \quad s_i = \sum_{k=1}^{n} x_{ik} u_k, \quad z_j = \sum_{i=1}^{n} a_{ij} s_i \tag{25}$$

A realization of (23)–(25) by neural network is shown in Fig. 6. The solution computed on the basis of (20) has no guarantee of symmetry and positive semi-definiteness (Cichocki and Kaczorek, 1992a). In order to find the unique stabilizing

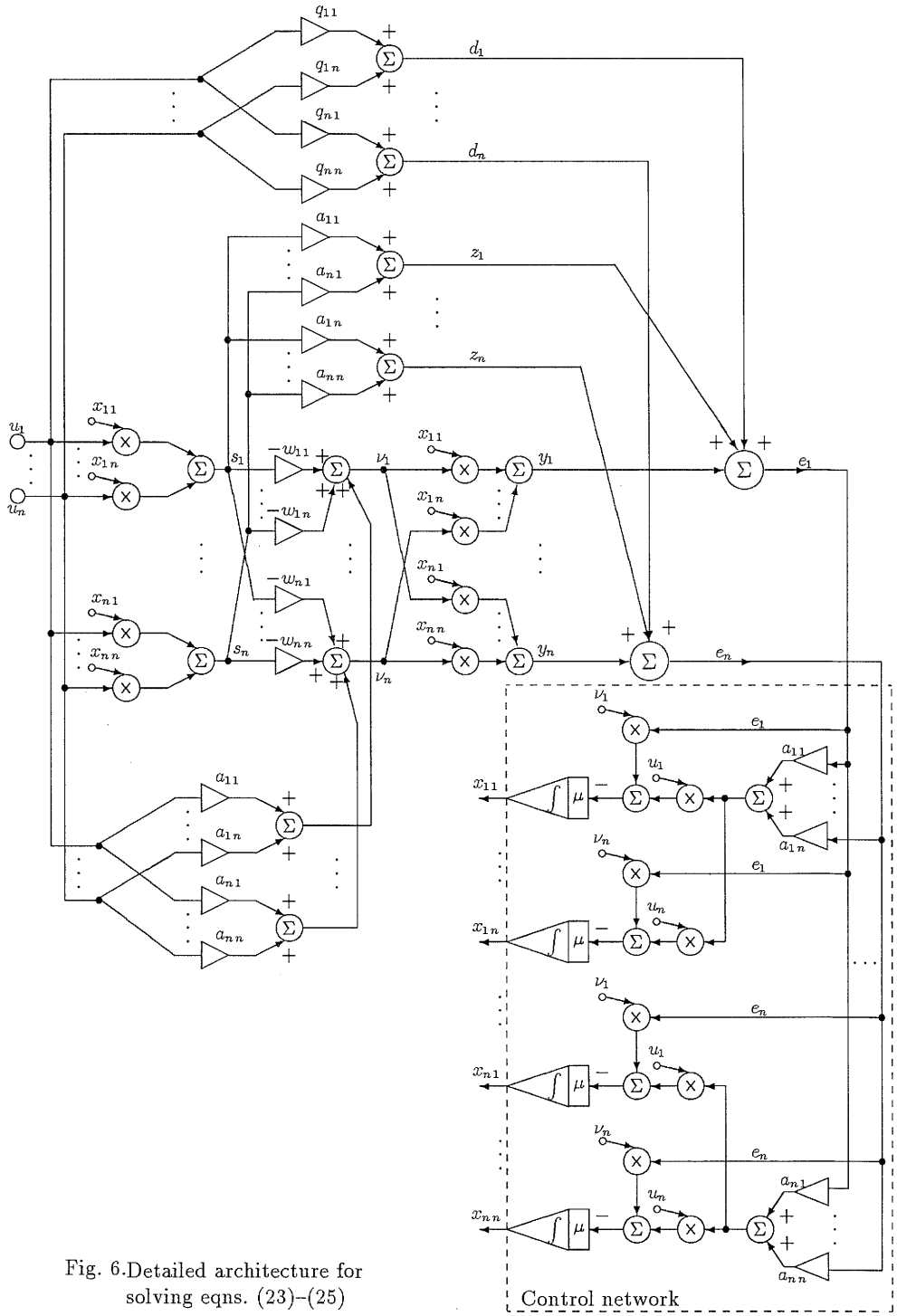


Fig. 6.Detailed architecture for solving eqns. (23)–(25)

solution of (20) we formulate the following constrained optimization problem: minimize $E = e^T e/2$ subject to the constraint $u^T(A - WX)u = u^T\nu < 0$ for any $u$ where $\nu = (A - WX)u$.

The constrained optimization problem can be mapped into an unconstrained one by using the Lagrange multiplier method (Bertsekas, 1982; Wierzbicki, 1981; Cichocki and Unbehauen, 1993). The unconstrained performance function has the form

$$E = \frac{1}{2}e^T e + \lambda\left[u^T\nu\right]_+ - \alpha\frac{\lambda^2}{2} \tag{26}$$

where $\lambda \geq 0$ is the Lagrange multiplier, $\left[u^T\nu\right]_+ \doteq \max(0, u^T\nu)$, $\alpha \geq 0$ is a stabilizing parameter.

Minimization of (26) by using the gradient method yields a system of differential equations (Cichocki and Kaczorek, 1992a)

$$\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} = -\mu\left[e_i\nu_j + \Big(\sum_{p=1}^{n} e_p a_{ip}\Big)u_j - \lambda\Big(\sum_{p=1}^{n} w_{pi}u_p\Big)u_j\right], \quad i, j, \ldots, n \tag{27}$$

$$\frac{\mathrm{d}\lambda}{\mathrm{d}t} = \gamma\Big\{\left[u^T\nu\right]_+ - \alpha\lambda\Big\} \tag{28}$$

where $\mu > 0$, $\gamma > 0$ are learning rates. The stabilizing parameter $\alpha$ is added to eliminate parasitic oscillations and improve the convergence behaviour of the network.

Algorithm (27)–(28) assures a stabilizing solution $X$ of (20) but still there is no guarantee of the symmetry of $X$. To avoid this disadvantage the algorithm eqns. (27)–(28) can be modified as follows

$$\frac{\mathrm{d}x_{ij}(t)}{\mathrm{d}t} = \frac{\mathrm{d}x_{ji}(t)}{\mathrm{d}t} = -\frac{\mu}{2}\left\{e_i\nu_j + e_j\nu_i - \lambda\left[\Big(\sum_{p=1}^{n} w_{pi}u_p\Big)u_j + \Big(\sum_{p=1}^{n} w_{pj}u_p\Big)u_i\right]\right\},$$

$$i, j = 1, 2, \ldots, n \tag{29}$$

$$\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} = \gamma\Big\{\left[u^T\nu\right]_+ - \alpha\lambda(t)\Big\} \tag{30}$$

The functional block diagram for the improved algorithm (29)–(30) is shown in Fig. 7. The network architecture shown in Fig. 7 may be viewed as an analog multivariable control feedback-loop system. This viewpoint makes it possible to use many powerful concepts and techniques from control theory in order to improve the convergence properties of the basic learning algorithm, for example by the use of optional PID controllers (Cichocki and Kaczorek, 1992a).

Now let us consider matrix equation (20) with complex coefficients. Substituting $A = A_1 + iA_2$, $W = W_1 + iW_2$, $Q = Q_1 + iQ_2$ and $X = X_1 + iX_2$, $i = \sqrt{-1}$, into (20) we obtain

$$(A_1 + iA_2)^*(X_1 + iX_2) + (X_1 + iX_2)(A_1 + iA_2) - (X_1 + iX_2)(W_1 + iW_2)(X_1 + iX_2)$$
$$+(Q_1 + iQ_2) = A_1^T X_1 + A_2^T X_2 + X_1 A_1 - X_2 A_2 + (X_2 W_2 - X_1 W_1)X_1$$

$$+(X_2W_1 - X_1W_2)X_2 + X_2 + Q_1 + i\Big[A_1^TX_2 - A_2^TX_1 + X_2A_1 + X_1A_2$$

$$-(X_2W_1 + X_1W_2)X_1 + (X_2W_2 - X_1W_1)X_2 + Q_2\Big] = 0$$

and

$$A_1^TX_1 + A_2^TX_2 + X_1A_1 - X_2A_2 + (X_2W_2 - X_1W_1)X_1$$

$$+(X_2W_1 - X_1W_2)X_2 + Q_1 = 0 \tag{31}$$

$$A_1^TX_2 + A_2^TX_1 + X_2A_1 - X_1A_2 - (X_2W_1 + X_1W_2)X_1$$

$$+(X_2W_2 - X_1W_1)X_2 + Q_2 = 0 \tag{32}$$

Note that the eqns. (31), (32) can be written in the form

$$[A_1^T, A_2^T]\begin{bmatrix}X_1\\X_2\end{bmatrix} + [X_1, X_2]\begin{bmatrix}A_1\\-A_2\end{bmatrix} + [X_1, X_2]\begin{bmatrix}-W_1W_2\\W_2W_1\end{bmatrix} + \begin{bmatrix}X_1\\X_2\end{bmatrix} + Q_1 = 0 \tag{33}$$

and

$$[-A_2^T, A_1^T]\begin{bmatrix}X_1\\X_2\end{bmatrix} + [X_1, X_2]\begin{bmatrix}A_2\\A_1\end{bmatrix} + [X_1, X_2]\begin{bmatrix}-W_2 - W_1\\-W_1\ W_2\end{bmatrix} + \begin{bmatrix}X_1\\X_2\end{bmatrix} + Q_2 = 0 \tag{34}$$

Therefore, eqn. (20) with complex coefficients has been reduced to eqns. (33), (34) of the same form with real coefficients.

The validity and performance of the proposed learning algorithm have been tested by computer simulations.

**Example 2.** Solve the algebraic Riccati equation (20) with

$$A = \begin{bmatrix}3 & 1 & -2\\-1 & 2 & -1\\1 & 3 & 1\end{bmatrix}, \quad W = \begin{bmatrix}2.0 & -0.5 & -0.5\\-0.5 & 1.0 & -0.5\\-0.5 & -0.5 & 1.0\end{bmatrix}, \quad Q = \begin{bmatrix}27.0 & -39.5 & -32.5\\-39.5 & 94.0 & 34.5\\-32.5 & 34.5 & 106.0\end{bmatrix}.$$

To solve the equation we employed the learning algorithm given by eqns. (29), (30) with $\mu = \gamma = 10^6$, $\alpha = 4$. The simulated network was able to find the solution

$$X = \begin{bmatrix}6.5162 & 2.5486 & 1.4319\\2.5486 & 19.5851 & 8.7127\\1.4319 & 8.7127 & 13.2978\end{bmatrix}$$

starting from zero initial conditions in a time of less than 70 microseconds. However, it was found that the network is able to compute the unique stabilizing solution

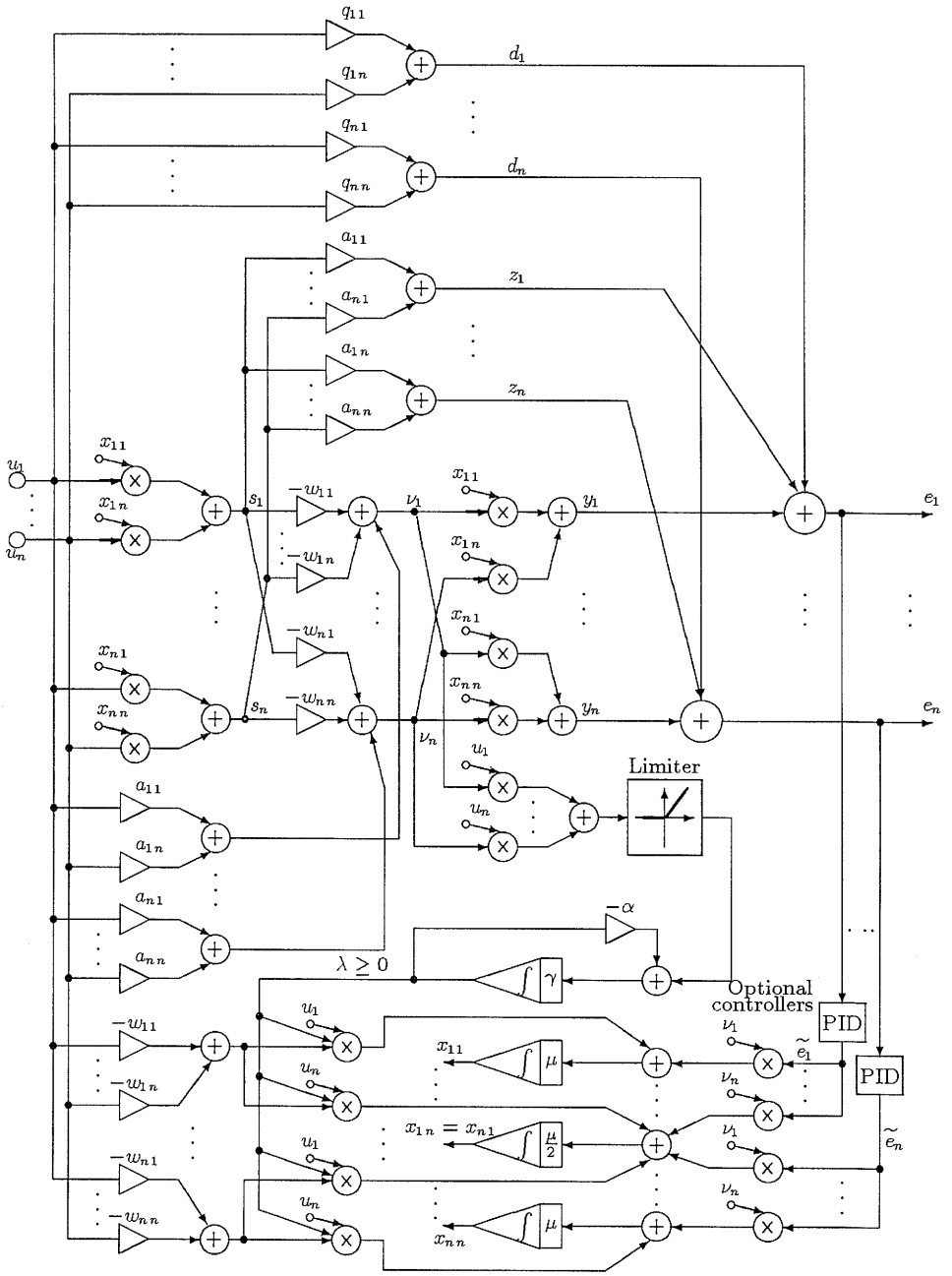


Fig. 7. Detailed architecture for solving the Riccati equations according to eqns. (29)–(30).

independent of the initial conditions. The results obtained are in good agreement with those found using MATLAB. The solution obtained from MATLAB is

$$X_{MATLAB} = \begin{bmatrix} 6.5118 & 2.5503 & 1.4338 \\ 2.5503 & 19.5874 & 8.7150 \\ 1.4378 & 8.7150 & 13.3004 \end{bmatrix}$$

### 3.3. Equations with Two Unknown Matrices

Consider the algebraic matrix equation (Cichocki and Kaczorek, 1992c)

$$AX - YB = C \tag{35}$$

where $A = [a_{ij}] \in \mathbb{R}^{m \times n}$, $B = [b_{ij}] \in \mathbb{R}^{q \times p}$ and $C = [c_{ij}] \in \mathbb{R}^{n \times p}$ are given and $X = [x_{ij}] \in \mathbb{R}^{n \times p}$, $Y = [y_{ij}] \in \mathbb{R}^{m \times q}$ are unknown.

It is assumed that eqn. (35) has a solution $(X, Y)$. Equation (35) has a solution if and only if (Cichocki and Kaczorek, 1992c)

$$(I - AA^{+})C(I - B^{+}B) = 0$$

where $A^{+}$ and $B^{+}$ are generalized Moore-Penrose pseudoinverse matrices of $A$ and $B$, respectively.

Postmultiplying (35) by a non-zero excitation vector $u = [u_1, \ldots, u_p]^T$ we obtain

$$AXu - YBu = Cu \tag{36}$$

Equation (36) can be represented by a multilayer neural network shown in Fig. 8. The layers corresponding to $A, B$ and $C$ represent neurons with fixed and known weights, while the layers corresponding to $X$ and $Y$ represent neurons with adjustable weights which must be determined during the optimization procedure.

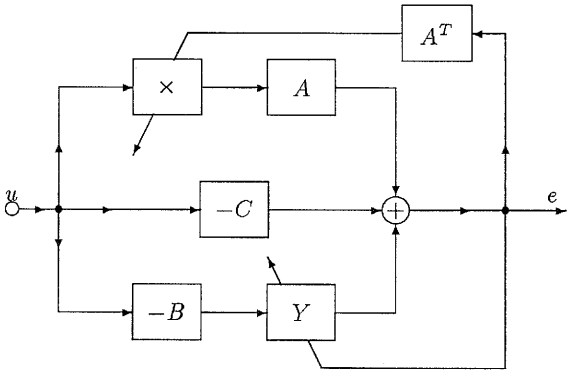


Fig. 8. Functional block diagram illustrating solving eqn. (36).

As the error function in the optimization procedure we assume

$$E = \frac{1}{2}e^T e, \quad e \doteq (AX - YB - C)u \tag{37}$$

To minimize (37) we shall use the gradient method based on the equations

$$\begin{aligned} \frac{\mathrm{d}x_{ij}}{\mathrm{d}t} &= -\mu\frac{\partial E}{\partial x_{ij}} \qquad \text{for} \quad i = 1,...,n, \;\; j = 1,...,p \\ \frac{\mathrm{d}y_{ij}}{\mathrm{d}t} &= -\mu\frac{\partial E}{\partial y_{ij}} \qquad \text{for} \quad i = 1,...,m, \;\; j = 1,...,q \end{aligned} \tag{38}$$

where $\mu > 0$ is the learning rate. Using the chain rule we obtain the non-linear differential equations

$$\begin{aligned} \frac{\mathrm{d}x_{ij}}{\mathrm{d}t} &= -\mu\Big(\sum_{r=1}^{m} a_n e_r\Big)u_j \qquad \text{for} \quad i = 1,...,n, \;\; j = 1,...,p \\ \frac{\mathrm{d}y_{ij}}{\mathrm{d}t} &= -\mu e_i \nu_j \qquad\qquad\qquad\quad \text{for} \quad i = 1,...,m, \;\; j = 1,...,q \end{aligned} \tag{39}$$

where $\nu_j = -\sum_{r=1}^{p} b_{jr}u_r$.

Equations (39) constitute the basic learning algorithm which can be considered a modification of the back-propagation algorithm.

If (35) has a solution $X, Y$, then, in general, it is not unique and therefore the neural network is able to find only one particular solution depending on the initial conditions. Starting from different initial conditions we can find a set of particular solutions of (35). In some applications additional requirements can be imposed on the solution, for example $X = X^T$ and $Y = Y^T$, positive definiteness, diagonal dominancy, etc. (Cichocki and Kaczorek, 1992c). In order to meet symmetry constraints for the matrices $X$ and $Y (X = X^T$ and $Y = Y^T)$, the learning algorithm given by (39) can be modified as follows:

$$\begin{aligned} \frac{\mathrm{d}x_{ij}}{\mathrm{d}t} &= -\frac{\mu}{2}\left[\Big(\sum_{r=1}^{m} a_n e_r\Big)u_j + \Big(\sum_{r=1}^{m} a_n e_r\Big)u_i\right], \\ \frac{\mathrm{d}y_{ij}}{\mathrm{d}t} &= -\frac{\mu}{2}(e_i\nu_j + e_j\nu_i), \end{aligned} \qquad i = 1,2,...,n, \;\; j = 1,2,...,p \tag{40}$$

To check the validity and performance of the proposed algorithm, we have simulated it extensively on computer. A very good agreement with the theoretical considerations has been obtained.

**Example 3.** Find a solution $X, Y$ of (35) such that $X = X^T$, $Y = Y^T$ with constraints $y_{11} = 1$, $y_{22} = 2$, $y_{33} = 3$. Assume

$$A = \begin{bmatrix} 1 & -1 \\ 2 & 1 \\ 3 & -1 \end{bmatrix}, \qquad B = \begin{bmatrix} -2 & -1 \\ -1 & -3 \\ 1 & -2 \end{bmatrix}, \qquad C = \begin{bmatrix} 4 & -8 \\ 8 & 4 \\ -5 & -5 \end{bmatrix}$$

On the basis of the system of differential equations (40) and the network architecture of Fig. 8 an appropriate circuit has been designed and simulated. As excitation signals the following ones have been employed $u_1(t) = \sin \omega t$, $u_2(t) = \sin 3\omega t$, $\omega = 8 \times 10^6$. The learning rate $\mu$ was chosen equal to $10^8$.

Computer simulated trajectories for zero initial conditions illustrating convergence behaviour of the network are shown in Fig. 9. The network was able to find the correct solution in time less than $50 \times 10^{-6}$ seconds independently of the initial conditions. The resulting matrices $X, Y$ after $4 \times 10^{-5}$ seconds were

$$X = \begin{bmatrix} 1.999 & 0.998 \\ 0.998 & 2.998 \end{bmatrix}, \qquad Y = \begin{bmatrix} 1.000 & -0.999 & -2.002 \\ -0.999 & 2.000 & -2.997 \\ -2.002 & -2.997 & 3.000 \end{bmatrix}$$

which are in good agreement with the exact solution

$$X = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \qquad Y = \begin{bmatrix} 1 & -1 & -2 \\ 1 & 2 & -3 \\ -2 & -3 & 3 \end{bmatrix}$$

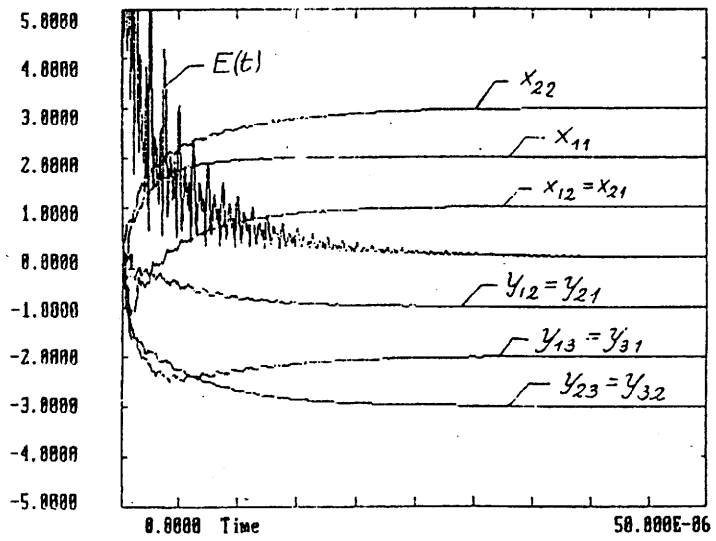


Fig. 9. Computer simulated trajectories for Example 3.

## 4. Polynomial Matrix Equations

Let $\mathbb{R}[s]$ be the ring of polynomials in $s$ with coefficients from the field of real numbers $\mathbb{R}$ and $\mathbb{R}^{m \times n}[s]$ be the set of $m \times n$ polynomial matrices in $s$ with coefficients from $\mathbb{R}$.

Consider the polynomial matrix equation (Cichocki and Kaczorek, 1993)

$$AXB = C \tag{41}$$

where $A \in \mathbb{R}^{m \times n}[s]$, $B \in \mathbb{R}^{p \times q}[s]$ and $C \in \mathbb{R}^{m \times q}[s]$ are given and $X \in \mathbb{R}^{n \times p}$ is unknown.

It is assumed that eqn. (41) has a real solution $X$. Using the Kronecker product of $A$ and $B$ denoted by $A \otimes B$ we can write (41) in the form (Kaczorek, 1984)

$$Dx = c \tag{42}$$

where

$$D \doteq A \otimes B^T = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_M \end{bmatrix} \in \mathbb{R}^{M \times N}[s], \quad M \doteq mq, \quad N \doteq np$$

$$x = [x_1, \ldots, x_n]^T, \qquad c = [c_1, \ldots, c_m]^T$$

$x_i$ and $c_i$ are the $i$-th rows of $X$ and $C$, respectively.

From comparison of the coefficients at $s$ of the same powers from (42) we obtain

$$\overline{D}x = \overline{c} \tag{43}$$

where

$$\overline{D} \doteq [D_{10}^T D_{11}^T \ldots D_{1N_{D1}}^T D_{20}^T D_{21}^T \ldots D_{2N_{D2}}^T D_{30}^T \ldots D_{MN_{DM}}^T]^T \in \mathbb{R}^{\overline{N} \times N}$$

$$\overline{N} \doteq \sum_{k=1}^{M} (N_{DK} + 1)$$

$$\overline{c} \doteq [\overline{c}_{10}\overline{c}_{11} \ldots \overline{c}_{1N_{D1}}\overline{c}_{20}\overline{c}_{21} \ldots \overline{c}_{2N_{D2}}\overline{c}_{30} \ldots \overline{c}_{MN_{DM}}]^T \in \mathbb{R}^{\overline{N}}$$

$$c \doteq \begin{bmatrix} \overline{c}_1 \\ \overline{c}_2 \\ \vdots \\ \overline{c}_M \end{bmatrix} \qquad \begin{aligned} D_i &\doteq D_{i0} + D_{i1}s + \ldots + D_{iN_{Di}}s^{N_{Di}} \\ \overline{c}_i &\doteq \overline{c}_{i0} + \overline{c}_{i1}s + \ldots + \overline{c}_{iN_{Di}}s^{N_{Di}} \end{aligned} \qquad i = 1, \ldots, M$$

By the Kronecker-Capelli theorem eqn. (43) has a real solution $x$ if and only if

$$\operatorname{rank}\overline{D} = \operatorname{rank}[\overline{D}, \overline{c}]$$

The problem of finding a real solution $X = \mathbb{R}^{n \times p}$ of (41) has been reduced to the problem of finding a real solution $x \in \mathbb{R}^N$ to the algebraic matrix equation (43) for given $\overline{D}$ and $\overline{c}$. To find $x$ the method presented in Section 2 can be applied.

Note that this approach can be easily extended for finding a polynomial solution $X \in \mathbb{R}^{n \times p}$ of (41).

## 5. Drazin Inverse of a Singular Matrix

Consider a singular matrix $A \in \mathbb{R}^{n \times n}$ with its known index $q$, i.e. the least positive integer $q$ such that $\text{rank}A^q = \text{rank}A^{q+1}$. A matrix $A^D \in \mathbb{R}^{n \times n}$ is called the *Drazin inverse* of the singular matrix $A \in \mathbb{R}^{n \times n}$ if it satisfies the conditions

$$\begin{aligned} &\text{(i)} \quad AA^D = A^D A \\ &\text{(ii)} \quad A^D A A^D = A^D \\ &\text{(iii)} \quad A^D A^{q+1} = A^q \end{aligned} \tag{44}$$

The Drazin inverse $A^D$ of a matrix $A \in \mathbb{R}^{n \times n}$ always exists and is unique. The problem under consideration can be stated as follows. Given a matrix $A \in \mathbb{R}^{n \times n}$, find its Drazin inverse $A^D$ using neural networks approach. Substituting in equations (44) $X = A^D$ and postmultiplying them by a non-zero excitation vector $u \in \mathbb{R}^n$ we obtain

$$\begin{aligned} &\text{(i)} \quad AXu = XAu \\ &\text{(ii)} \quad XAXu = Xu \\ &\text{(iii)} \quad XA^{q+1}u = A^q u \end{aligned} \tag{45}$$

Equations (45) can be represented by a multilayer neural network shown schematically in Fig. 10. The layers corresponding to $A$, $A^{q-1}$ and $A^q$ represent neurons with fixed and known weights, while the layers corresponding to $X$ represent neurons with adjustable weights which must be determined during the optimization procedure.

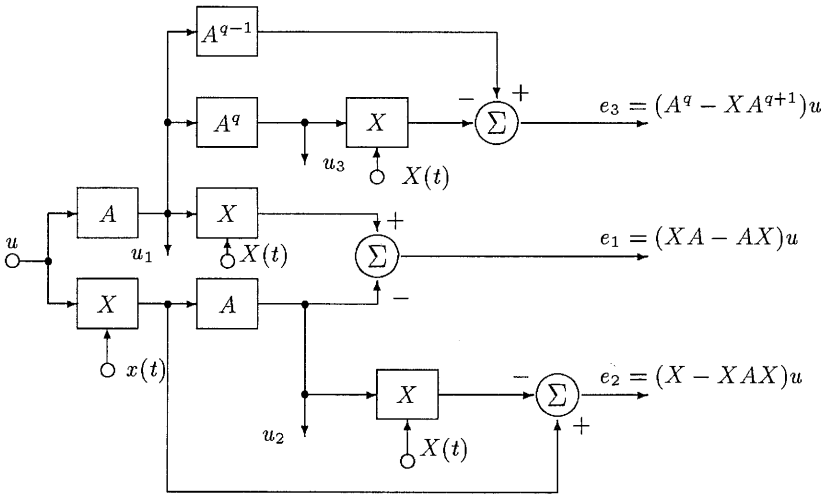


Fig. 10. Functional block diagram illustrating solving set of eqns. (45).

As the error function in the optimization procedure we assume

$$E = \frac{1}{2}(w_1 e_1^T e_1 + w_2 e_2^T e_2 + w_3 e_3^T e_3) \tag{46}$$

where

$$e_1 \doteq (XA - AX)u, \quad e_2 \doteq (X - XAX)u, \quad e_3 \doteq (A^q - XA^{q+1})u$$

and $w_1, w_2, w_3$ are weight positive coefficients (typically $w_1 = w_2 = w_3 = 1$). To minimize (46) we shall use the gradient method (Cichocki and Unbehauen, 1993) based on the equation

$$\frac{\mathrm{d}X}{\mathrm{d}t} = -\mu \frac{\partial E}{\partial X} \tag{47}$$

where $\mu > 0$ is the optimization rate. Using the chain rule we obtain

$$\frac{\mathrm{d}X(t)}{\mathrm{d}t} = -\mu \left[ e_1 u_1^T - A^T e_1 u^T - e_2 u_2^T + e_2 u^T - e_3 u_3^T \right] \tag{48}$$

Differential equation (48) constitutes the basic learning algorithm which can be considered a modification of the backpropagation algorithm (Cichocki and Unbehauen, 1993). The realization of the algorithm by a multilayer neuron – like network is shown in Fig. 11. To check the validity and performance of the neural network approach to the computation of Drazin inverse some computer simulations have been done. A very good agreement with the theoretical considerations has been obtained.

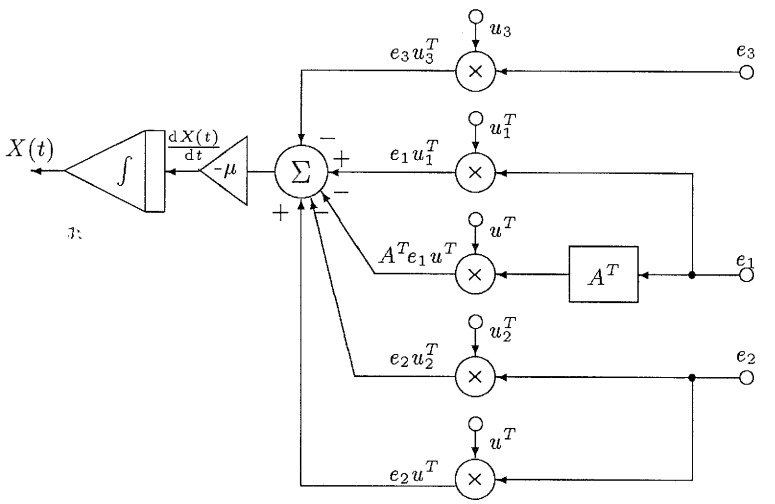


Fig. 11. Implementation of the learning algorithm (48).

**Example 4.** Find the Drazin inverse matrix of the singular matrix

$$A = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{49}$$

with the index $q = 1$.

It is easy to check that the Drazin inverse of matrix (49) has the form

$$A^D = \begin{bmatrix} 1 & -2 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

On the basis of the system of differential equations (48) and the neural network with block architecture of Fig. 10 a computer simulation on the IBM PC has been done. The following excitation signals have been used:

$$u = \begin{bmatrix} \sin(\omega t) \\ \sin(2\omega t) \\ \sin(3\omega t) \end{bmatrix}$$

The learning rate $\mu$ was chosen as $\mu = 10^7$ and $\omega = 10^5$. The following matrix after $10^{-5}$ seconds has been obtained

$$\begin{bmatrix} 1.00447 & -2.01459 & -1.0081 \\ 8 \times 10^{-5} & 1.00135 & 9 \times 10^{-4} \\ 5 \times 10^{-3} & -1.01220 & -6 \times 10^{-3} \end{bmatrix}$$

which is in good agreement with the exact solution.

## 6. Concluding Remarks

The key step in solving the above problems by the use of neural-type structured networks has been the construction of an appropriate error function $E$ such that its minimal value corresponds to the desired solution. The derivation of the error function enables us to transform the minimization problem into a set of differential equations on the basis of which a suitable neural-type structured network can be designed. The continuous-time (analog) formalism applied in these algorithms makes the proposed neural-type structured networks more suitable for implementation in VLSI technology. This follows from the fact that learning algorithms are expressed by systems of non-linear differential equations instead of the usually employed difference equations; this eliminates the timing, clocking and synchronization problems. Moreover, the learning rate $\mu$ may be sufficiently large without affecting the stability of the system in contrast to a corresponding discrete-time system where the learning rate must be small. Simulation experiments have fully confirmed the validity and correctness of the developed algorithms. The results obtained indicate that the proposed approach is an alternative and promising method of solving a large class of algebraic problems. The proposed method can be easily extended for 1) algebraic matrix equations with complex coefficients and slowly time-varying coefficients, 2) polynomial matrix equations with two (or more) independent variables. One of the open problems is to extend the method presented in Section 5 for computing the Drazin inverse of a singular matrix with its unknown index. Another open problem is

to extend the method for computing minimal degree solutions of polynomial matrix equations.

## References


Bertsekas D.P. (1982): *Constrained Optimization and Lagrange Multiplier Methods.* — New York: Academic Press.

Charlier J.P. and Van Dooren P. (1989): *A systolic algorithm for Riccati and Lyapunov equations.* — Mathematics of Control, Signals and Systems, v.2, No.2, pp.109–136

Cichocki A. (1992): *Neural network for singular value decomposition.* — Electronics Letters, v.28, No.8, pp.784–786.

Cichocki A. and Kaczorek T. (1992a): *Neural-type structured networks for solving algebraic Riccati equations.* — Arch. of Contr. Sci., v.XXXVI, No.3–4, pp.153–165.

Cichocki A. and Kaczorek T. (1992b): *Solving of algebraic matrix equations by use of neural networks.* — Bull. Pol. Acad. Techn. Sci., v.40, No.1, pp.61–67.

Cichocki A. and Kaczorek T. (1992c): *Solving of real matrix equations $AX - YB = C$ making use of neural networks.* — Bull. Pol. Acad. Techn. Sci., v.40, No.1, pp.273–279.

Cichocki A. and Kaczorek T. (1993): *Determination of real solutions to two-sided polynomial matrix equations making use of neural networks.* — Bull. Pol. Acad. Techn. Sci., v.40, No.3, pp.281–287.

Cichocki A., Kaczorek T. and Stajniak A. (1992): *Computation of the Drazin inverse of a singular matrix making use of neural networks.* — Bull. Pol. Acad. Techn. Sci., v.40, No.4,pp.336–394.

Cichocki A. and Unbehauen R. (1992): *Neural Networks for solving systems of linear equations and related problems, Part I and II.* — IEEE Trans. Circuits and Systems, v.CAS–39, No.2 and No.9, pp.124–138, and pp.619–633.

Cichocki A. and Unbehauen R. (1993): *Neural Networks for Optimization and Signal Processing.* — Chichester/Stuttgart: Teubner– Wiley.

Distante F., Sami M., Stefanelli R. and Storti–Gajami G. (1991): *Mapping neural nets onto a massively parallel architecture, a defect–tolerance solution.* — Proc. IEEE, v.79, No.4, pp.444–460.

Flaherty D.T. and Michell A.N. (1990): *A controlled gradient search for the backpropagative algorithm.* — Proc. Conf. *Decision Control*, Honolulu, pp.31–40.

Gardiner J.D. and Laub A.J. (1991): *Parallel algorithms for algebraic Riccati equations.* — Int. J. Control, v.54, No.6, pp.1317–1333.

Kaczorek T. (1984): *Matrices in Automation and Electrotechnics.* —Warsaw: WNT, (in Polish).

Kenney C., Laub A.J. and Jonckhheere E. (1989): *Positive and negative solutions of dual Riccati equations by matrix sign. function creation.* — System and Control Letters, v.13, No.3, pp.109–116.

Kučera V. (1972): *A contribution to matrix quadratic equation.* — IEEE Trans. Automat. Contr., v.17, No.3, pp.344–347.

Laub A.J. (1979): *A Shur method for solving algebraic Riccati equations.* — IEEE Trans. Automat. Contr., v.24, No.8, pp.913–921.

Lillo W.E., Loh M.–H., Hui S. and Żak S.H. (1993): *Neural networks for problems.* — Int. J. Circuit Theory and Applications, v.21, No.4, pp.385–399.

Osowski S. (1993): *Signal flow graphs and neural networks.* — Biological Cybernetics, v.70, No.4, pp.387–395.

Polycarpon M.M. and Ioannou P.A. (1992): *Learning and convergence analysis of neural–type structured networks.* — IEEE Trans. Neural Networks, v.3, No.1, pp.39–50.

Rumelhart D.E., Hinton G.E. and Willams R.J. (1986): *Learning internal representations by error propagation*, In: Parallel Distributed Processing (Rumelhart D.E. and McClelland J.L., Eds.), Reading: MIT Press.

Unbehauen R. and Cichocki A. (1989): *MOS Switched-Capacitor Integrated Circuits and Systems.* — New York: Springer-Verlag.

Wang L. and Mendel J.M. (1992): *Parallel structured networks for solving of wide variety of matrix algebra problems.* — J. Parallel and Distributed Computing, v.14, No.2, pp.236–247.

Wang J. and Wu G. (1993): *Recurrent neural networks for LU decomposition and Cholesky factorization.* — Math. Comput. Modelling, v.18, No.1, pp.1–8.

Wei L.–F. and Yeh F.–B. (1991): *A modified Shur method for solving algebraic Riccati equations.* — Systems and Control Letters, v.16, No.5, pp.295–297.

Wierzbicki A. (1981): *Methods of Mathematical Programming.* — Warsaw: Polish Scientific Publishers, (in Polish).