3-22-1996

Neural network architecture for solving the algebraic matrix Riccati equation

Fredric M. Ham

Emmanuel G. Collins

Neural network architecture for solving the algebraic matrix Riccati equation

Fredric M. Ham
Emmanuel G. Collins

A neural network architecture for solving the algebraic matrix Riccati equation

Fredric M. Ham and Emmanuel G. Collins

Florida Institute of Technology
Electrical Engineering Program
150 West University Boulevard
Melbourne, Florida 32901-6988
E-mail: fmh@ee.fit.edu

Florida A&M University-Florida State University
Department of Mechanical Engineering
2525 Pottsdamer Street
Tallahassee, Florida 32310-6046
E-mail: ecollins@eng.fsu.edu

ABSTRACT

This paper presents a neurocomputing approach for solving the algebraic matrix Riccati equation. This approach is able to utilize a good initial condition to reduce the computation time in comparison to standard methods for solving the Riccati equation. The repeated solutions of closely related Riccati equations appears in homotopy algorithms to solve certain problems in fixed-architecture control. Hence, the new approach has the potential to significantly speed-up these algorithms. It also has potential applications in adaptive control. The structured neural network architecture is trained using error backpropagation based on a steepest-descent learning rule. An example is given which illustrates the advantage of utilizing a good initial condition (i.e., initial setting of the neural network synaptic weight matrix) in the structured neural network.

Keywords: adaptive control, convergence, fixed-architecture control, homotopy, Riccati equation, steepest descent, structured neural network

1. INTRODUCTION

One of the more practical areas of modern control is the design of fixed-architecture (e.g., reduced-order and decentralized), robust controllers. The synthesis of these controllers results in relatively complex optimization problems. In recent years the solutions to these optimization problems have been effectively approached by using homotopy algorithms[1,2]. These methods essentially work by posing a related but easier problem, and deforming the easier problem (and its associated solution) into the original problem.

Each iteration in the homotopy algorithms involve the solution of one or more algebraic Riccati equations. At each iteration, the parameters of a given Riccati equation are deformations of the parameters of the corresponding Riccati equation from the previous iteration. Hence, the solution P_k of the current Riccati equation should be "close" to the solution P_{k-1} of the Riccati equation of the previous iterate. Using P_{k-1} as an initial condition to a Riccati solver for P_k seems to be an excellent way for reducing the computation time of such algorithms. However, many standard solution techniques are not able to take advantage of this information. This paper presents a neurocomputing approach for solving the algebraic Riccati equation that is able to use a "good" initial condition (i.e., synaptic weights from the previous iterate) to reduce the computation time in comparison to standard methods. This approach may also find applications in full-order, adaptive H_2 and H_∞ control.

2. DEVELOPMENT OF THE NEUROCOMPUTING APPROACH

We consider the standard Riccati equation

$$A^T X + XA - XRX + Q = 0 \tag{1}$$

0-8194-2141-3/96/$6.00

where $A \in \Re^{nxn}$, $R \in \Re^{nxn}$, $R^T = R$ (symmetric), $R > 0$ (positive definite), $Q \in \Re^{nxn}$, $Q^T = Q$, and $Q \geq 0$ (non-negative definite). Therefore, we seek the non-negative definite solution to (1), i.e., $X \in \Re^{nxn}$, $X^T = X$, and $X \geq 0$. A structured neural network[3,4,5] for computing the solution, X, to (1) is shown in block diagram form in Fig. 1. The instantaneous error vector in Fig. 1 is given as

$$e = b - y = XRv - A^T Xz - XAz - Qz \tag{2}$$

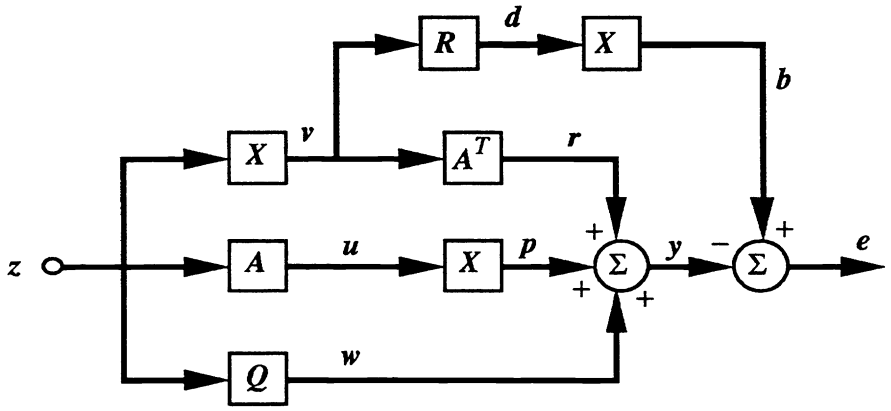

Figure 1. Network architecture for solving the algebraic matrix Riccati equation.

Also from Fig. 1, we see that $v = Xz$, therefore, (2) can be written as

$$e = \left[XRX - A^T X - XA - Q\right]z \tag{3}$$

and inside the brackets is the negative of the left-hand side of the Riccati equation given in (1). Therefore, as X converges to the solution to (1), the error vector will approach zero given an appropriate external excitatory input signal $z(t)$ (this is discussed below). An error performance cost function can be formulated using the error vector in (2) as

$$\mathrm{E}(X) = \frac{1}{2}\|e\|_2^2 \tag{4}$$

where $\|\bullet\|_2$ is the L_2 (or Euclidean) norm. Our objective is the minimization of the error performance cost function in (4) according to the gradient steepest-descent method[6] leading to a set of first-order matrix differential equations, i.e.,

$$\frac{\mathrm{d}X(\mathrm{t})}{\mathrm{dt}} = -\mu\nabla_X \mathrm{E}(X) = -\mu\frac{\partial \mathrm{E}(X)}{\partial X} \tag{5}$$

where $\mu > 0$ is the learning rate parameter, which must be set appropriately to ensure convergence. Therefore, from (5) we see that the gradient of the error performance cost function must be computed, i.e., $\nabla_X \mathrm{E}(X)$. The error performance cost function can be written using (2) as

$$\mathrm{E}(X) = \frac{1}{2}\|e\|_2^2 = \frac{1}{2}e^T e = \frac{1}{2}\left(XRv - A^T Xz - XAz - Qz\right)^T \left(XRv - A^T Xz - XAz - Qz\right) =$$
$$\frac{1}{2}(v^T R^T X^T XRv - v^T R^T X^T A^T Xz - v^T R^T X^T XAz - v^T R^T X^T Qz -$$
$$z^T X^T AXRv + z^T X^T AA^T Xz + z^T X^T AXAz + z^T X^T AQz -$$
$$z^T A^T X^T XRv + z^T A^T X^T A^T Xz + z^T A^T X^T XAz + z^T A^T X^T Qz -$$
$$z^T Q^T XRv + z^T Q^T A^T Xz + z^T Q^T XAz + z^T Q^T Qz). \tag{6}$$

This approach can be considered as a *batch* formulation of the problem which will lead to the continuous-time vector-matrix form for the learning rule in (5). In order to compute the gradient of (6), two general results must be applied from matrix calculus[7], i.e.,

$$\frac{\partial}{\partial A}\mathrm{trace}(BAC) = B^T C^T \tag{7}$$

and

$$\frac{\partial}{\partial A}\mathrm{trace}\left(BA^T C\right) = CB. \tag{8}$$

From (6) 24 derivative terms will result from computing the gradient, however, because (6) is a quadratic form there are actually only 12. We will only calculate the partial derivative of the first term to show how the *batch* approach can be taken and then state the final result. The partial derivative of the first term in (6) is given as

$$\frac{1}{2}\frac{\partial}{\partial X}\mathrm{trace}\left(v^T R^T X^T XRv\right) = \frac{1}{2}XRvv^T R^T + \frac{1}{2}XRvv^T R^T \tag{9}$$

where the general results in (7) and (8), along with the appropriate chain rule, were applied. Carrying out this procedure for the remaining terms in (6) results in the gradient of $\mathrm{E}(X)$ as

$$\nabla_X \mathrm{E}(X) = [\underbrace{XRv - A^T Xz - XAz - Qz}_{e}]v^T R^T -$$
$$[\underbrace{XRv - A^T Xz - XAz - Qz}_{e}]z^T A^T - A[\underbrace{XRv - A^T Xz - XAz - Qz}_{e}]z^T. \tag{10}$$

Therefore, the gradient can be written as ($R = R^T$)

$$\nabla_X \mathrm{E}(X) = ev^T R - ez^T A^T - Aez^T \tag{11}$$

and the continuous-time vector-matrix form for the learning rule in (5) can now be written as

$$\frac{\mathrm{d}X(t)}{\mathrm{dt}} = \mu\nabla_X \mathrm{E}(X) = \mu\left[Ae(t)z^T(t) + e(t)z^T(t)A^T - e(t)v^T(t)R\right] \tag{12}$$

However, the learning rule in (12) does not satisfy the symmetry requirement for X, (i.e., $X = X^T$). Although, as in the case of the neurocomputing approach for solving the algebraic Lyapunov equation given by Cichocki and Unbehauen[8], if the transpose of each term in the gradient in (11) is included along with the other terms, and the learning rate parameter μ is replaced with $\mu/2$, the symmetry requirement for X is met. The resulting learning is given as

$$\frac{dX(t)}{dt} = \frac{\mu}{2}\nabla_X E(X) = \frac{\mu}{2}[Ae(t)z^T(t) + z(t)e^T(t)A^T + e(t)z^T(t)A^T + Az(t)e^T(t) - e(t)v^T(t)R - Rv(t)e^T(t)]. \tag{13}$$

In (13) we see that no divisions are required. Now the learning rule in (13) can be written in discrete-time form as

$$X(k+1) = X(k) + \frac{\mu}{2}[Ae(k)z^T(k) + z(k)e^T(k)A^T + e(k)z^T(k)A^T + Az(k)e^T(k) - e(k)v^T(k)R - Rv(k)e^T(k)] \tag{14}$$

where k is the time index, $e(k)$ is given in (2), and $v(k) = X(k)z(k)$ from Fig. 1. The scalar form for the discrete-time learning rule in (14) can be written as

$$x_{ij}(k+1) = x_{ij}(k) + \frac{\mu}{2}\left\{\left[\sum_{h=1}^{n} a_{ih}e_h(k)\right]z_j(k) + z_i(k)\left[\sum_{h=1}^{n} e_h(k)a_{hi}\right] + e_i(k)\left[\sum_{h=1}^{n} z_h(k)a_{hi}\right] + \left[\sum_{h=1}^{n} a_{ih}z_h(k)\right]e_j(k) - e_i(k)\left[\sum_{h=1}^{n} v_h(k)r_{hj}\right] - \left[\sum_{h=1}^{n} r_{ih}v_h(k)\right]e_j(k)\right\} \tag{15}$$

and

$$e_i(k) = \left[\sum_{h=1}^{n} x_{ih}(k)r_{hj}\right]v_i(k) - \left[\sum_{h=1}^{n} a_{hi}x_{hj}(k)\right]z_i(k) - \left[\sum_{h=1}^{n} x_{ih}(k)a_{hj}\right]z_i(k) - \left[\sum_{h=1}^{n} q_{ih}z_h(k)\right] \tag{16}$$

where $v_i(k) = \sum_{h=1}^{n} x_{ih}(k)z_h(k)$, for $i, j = 1, 2, \cdots, n$. Figure 2 shows the multi-layer neural network architecture for computing the solution to the algebraic matrix Riccati equation. From (15) and (16) we see that the learning rule is an error backpropagation algorithm for computing the Riccati equation solution, X.

The external excitatory vector input signals, z, shown in Figs. 1 and 2, can be selected as a set of linearly independent bi-polar vectors given as

$$z^{(1)} = [1\ -1\ \cdots\ -1]^T,\ z^{(2)} = [-1\ 1\ \cdots\ -1]^T,\ \cdots,\ z^{(n)} = [-1\ -1\ \cdots\ 1]^T. \tag{17}$$

All vectors in (17) are randomly presented to the neural network every training epoch. Therefore, for N (an integer) training epochs there is a total of Nxn presentations of the bi-polar excitatory input vectors.

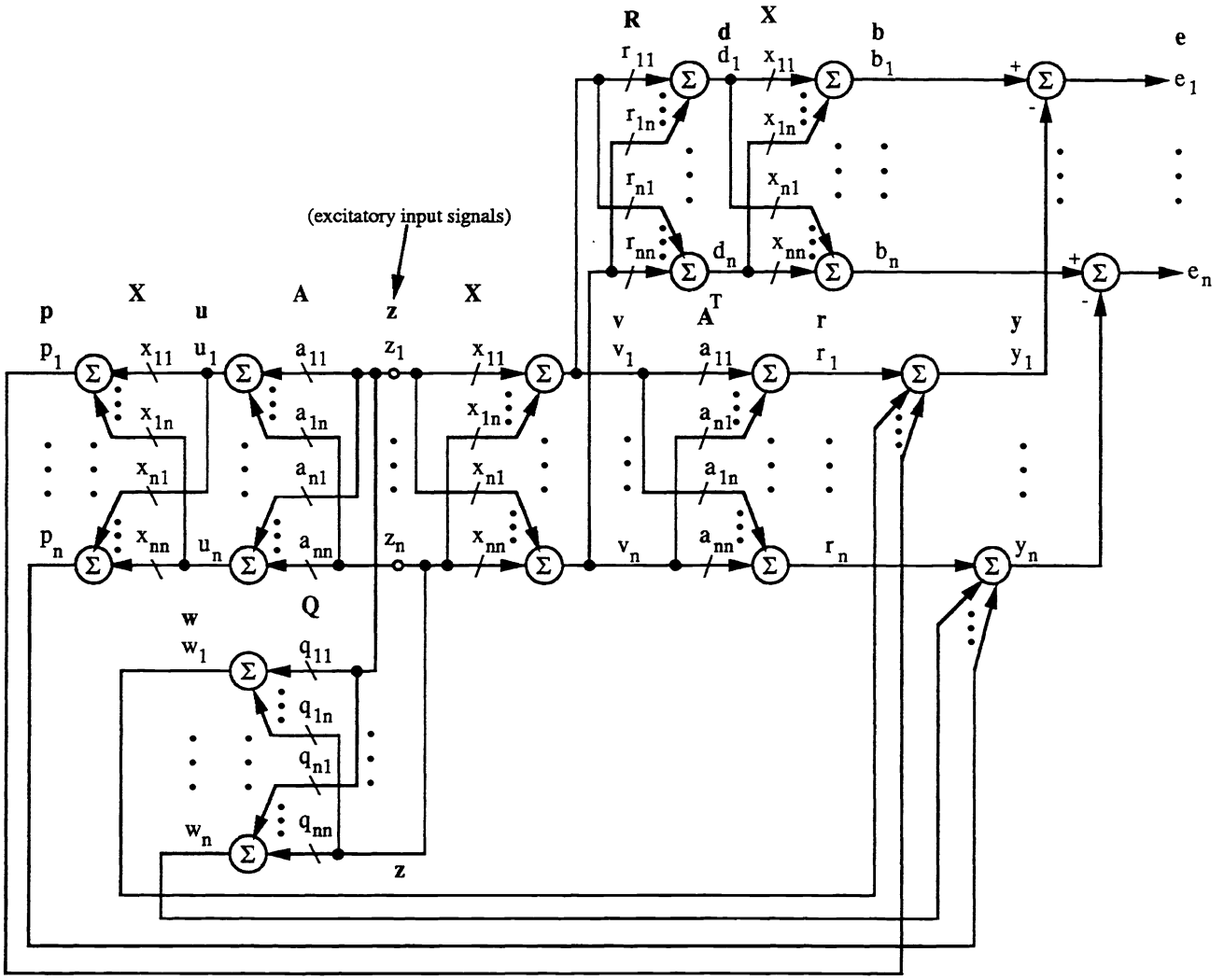

Figure 2. Multi-layer neural network architecture for solving the algebraic matrix Riccati equation. The a_{ij}, q_{ij}, and r_{ij} for $i, j = 1, 2, \cdots, n$ elements are fixed and x_{ij} for $i, j = 1, 2, \cdots, n$ are adaptive.

3. SIMULATION EXAMPLE

The following example was run on a Pentium-P5-90 PC using Matlab 4.2c.1.

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix}, \boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 3.3333 & 0 \\ 0 & 0 & 14.2857 \end{bmatrix} \tag{18}$$

The Matlab Control Systems Toolbox function "lqr2" was first used to solve the steady-state algebraic matrix Riccati equation. The "lqr2" function solves the linear quadratic regulator problem using the Schur decomposition method. Therefore, this function requires as inputs the matrices $\boldsymbol{A}$, $\boldsymbol{Q}$, $\tilde{\boldsymbol{R}}$, and $\boldsymbol{B}$, where

$\tilde{R} = R^{-1}$ (given above) and the "R" matrix is internally computed as $B\tilde{R}^{-1}B^T$, where $B = I_{nxn}$ (identity matrix). The solution yielded by the Schur decomposition method for solving the Riccati equation in Matlab, X_M, is given by

$$X_M = \begin{bmatrix} 0.3324 & 0.1094 & -0.0123 \\ 0.1094 & 0.3790 & -0.0059 \\ -0.0123 & -0.0059 & 0.0388 \end{bmatrix} \tag{19}$$

and required 270 ms. The neural network solution, X_{NN}, using a learning rate $\mu = 0.00275$, N=500 (or Nxn =1500 iterations, where n=3), and with the initial conditions $X_{NN}(k=0) = 0$, the null matrix), yielded the same results as the Matlab solution (out to 4 decimal places) as shown in (19). The neurocomputing approach required 4.73 sec to converge, and the elements of the error vector e in (2) are plotted separately in Fig. 3a.

To illustrate how the neurocomputing approach for solving the Riccati equation can take advantage of prior information when the need arises to recompute the solution for slowly varying parameters in A, a perturbed matrix A is now given as

$$A_p = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -10.9945 & -6 \end{bmatrix} \tag{20}$$

where the *32* element in A_p represents a 0.05% change (reduction) relative to the original (nominal) plant, A, given in (18). The neural network only required N=29 (or Nxn =87) iterations to converge to the new solution (or 220 ms), where the previous learning parameter was used, however, the nominal plant Matlab Riccati solution, X_M, given in (19) was used as the initial X matrix. The resulting neural network solution for the perturbed plant, A_p, is given as

$$X_{NNp} = \begin{bmatrix} 0.3324 & 0.1093 & -0.0123 \\ 0.1093 & 0.3789 & -0.0059 \\ -0.0123 & -0.0059 & 0.0388 \end{bmatrix} \tag{21}$$

and the elements of the error vector e are plotted separately in Fig. 3b. The Matlab Schur decomposition method again required 270 ms to solve the Riccati equation and the solution, X_{Mp}, was the same as that obtained by the neural network (out to 4 decimal places) as shown in (21). Therefore, the neurocomputing approach was able to compute the Riccati solution 50 ms faster than the Matlab Schur decomposition method.

The learning rule that was implemented in Matlab for the neurocomputing approach to solve the algebraic matrix Riccati equation is given by (15) (the discrete-time form). The convergence times achieved in the simulations shown above would be significantly decreased with a parallel implementation of the learning rule (see Fig. 2). Also, the steepest-descent gradient method used for the learning rule is perfectly adequate for this application because the desired repeated solutions to the related Riccati equations will always be very "close" to each other. Therefore, only a relatively small number of iterations is necessary to compute the "new" solution. That is, a more sophisticated gradient method such as a conjugate gradient technique [6] would not be necessary. In fact, because of the additional computations necessary in a conjugate gradient

method associated with the line search, this approach would require more computations than the steepest descent method.

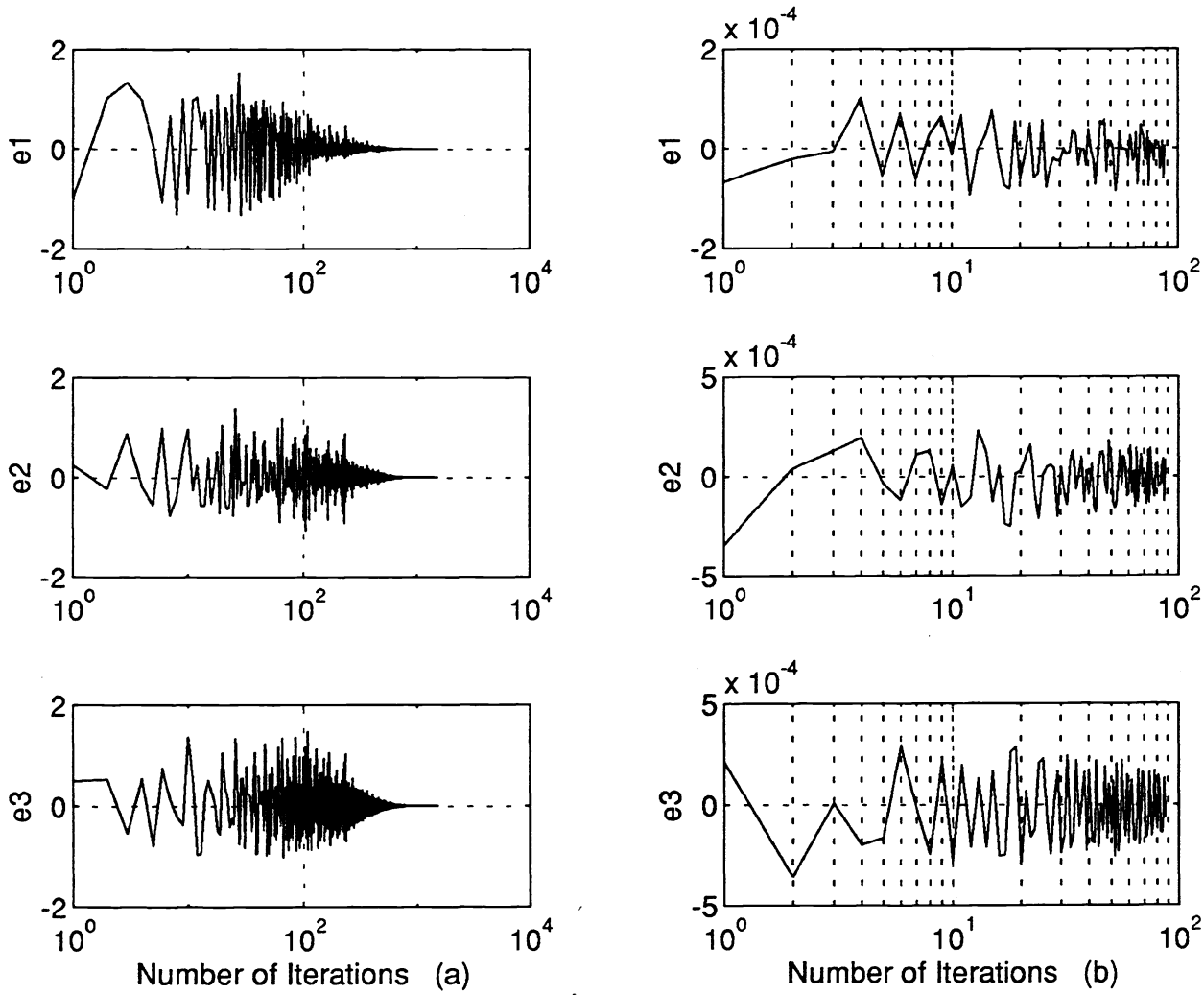

Figure 3. (a) Neural network learning error for the initial solution to the Riccati equation. (b) Neural network learning error for the perturbed system plant.

4. CONCLUSIONS

We have shown that a neurocomputing approach can be used to solve the algebraic matrix Riccati equation. When a good initial condition (i.e., prior synaptic weights) is provided, a situation that is common within certain homotopy algorithms for fixed-architecture control, a neurocomputing approach can yield a solution to the Riccati equation significantly faster than standard solution techniques. This is primarily because the standard methods do not have the ability to utilize prior information when computing the solution.

5. ACKNOWLEDGMENT

The authors would like to thank Dr. Wassim M. Haddad, Associate Professor, Department of Aerospace Engineering, Georgia Institute of Technology, for his helpful suggestions.

6. REFERENCES

1. Y. Ge, E.G. Collins, Jr., and L.T. Watson, "A homotopy algorithm for full and reduced order mixed norm H_2 / H_∞ synthesis," *Proceedings of the IEEE Conference on Decision and Control*, Lake Buena Vista, FL, pp. 2672-2677, Dec. 1994.

2. E.G. Collins, Jr., W.M. Haddad, and L.T. Watson, "Probability-one homotopy algorithms for robust controller analysis and synthesis with fixed-structure multipliers," *Proceedings of the IEEE Conference on Decision and Control*, New Orleans, LA, Dec. 1995 (to appear).

3. L. Wang and J.M. Mendel, "Structured trainable networks for matrix algebra," Proceeding of the *IJCNN*, Vol. II, pp. 125-132, Jun. 1990.

4. L. Wang and J.M. Mendel, "Parallel structured networks for solving a wide variety of matrix algebra problems," *Journal of Parallel and Distributed Computing*, Vol. 14, pp. 236-247, 1992.

5. M.M. Polycarpou and P.A. Ioannou, "Learning and convergence analysis of neural-type structured networks," *IEEE Transactions on Neural Networks*, Vol. 3, pp. 39-50, Jan. 1992.

6. D.G. Luenburger, *Linear and Nonlinear Programming*, 2nd Ed., Addison-Wesley Publ. Co., Reading, MA, 1984.

7. F.L. Lewis, *Optimal Control*, Wiley-Interscience, New York, 1986.

8. A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley and Sons, New York, 1993.