Recursive Orthogonal Least Squares Learning with Automatic Weight Selection for Gaussian Neural Networks

Meng H. Fun, mhfun@yahoo.com, Oklahoma State University
Martin T. Hagan, mhagan@master.ceat.okstate.edu, Oklahoma State University

Abstract

Gaussian neural networks have always suffered from the curse of dimensionality; the number of weights needed increases exponentially with the number of inputs and outputs. Many methods have been proposed to solve this problem by optimally or sub-optimally selecting the weights or centers of the Gaussian neural network [1],[2]. However, most of these attempts are not suitable for online implementation. In this paper, we develop a Recursive Orthogonal Least Squares learning with Automatic Weight Selection (ROLS-AWS) for a two-layered Gaussian neural network. This ROLS-AWS algorithm is capable of selecting useful weights sub-optimally and recursively. In doing so, we will not only reduce the growth of the size of the weights but also minimizes the number of weights used. Due to the recursive nature of this algorithm, it can be applied to any online system, as in control and signal processing applications.

Introduction

The Radial Basis Function (RBF) network often requires many hidden nodes due to their localized character. For practical purposes, it is desirable to construct the smallest possible RBF network. Many applications that use the RBF network have opted to use fixed centers or fixed grid size to limit the number of nodes used [6],[7]. However, this approach has often leads to large weights. On the other hand, the Orthogonal Least Squares (OLS) learning algorithm proposed by S. Chen [2], is a simple and efficient algorithm for fitting the RBF network. It also has the capability to select smaller weight and to create a parsimonious network model. However, one drawback with this algorithm is that the training is done in batch mode only.

In this paper, we will develop the Recursive Orthogonal Least Squares Learning with Automatic Weight Selection (ROLS-AWS) algorithm, which is based on batch orthogonal least squares learning. Generally, there are three methods to choose the centers and the variances: the fixed centers method, the self-organized learning method (e.g. *k* mean clustering method), and the stochastic gradient method [3]. Here we will only explore the fixed centers method. Such an approach is first described by Broomehead & Lowe [4], and was then extended by S. Chen, who developed the orthogonal least squares learning method [2]. The fixed centers method considered here is a little different than the standard method, to accommodate for the recursion in time. We first assume that each available data point will be a potential centers for the RBF network. When a data point is available, the center for that data point is available as well. Meanwhile, the variances are fixed. Such an RBF network is linear in the parameters, since all RBF centers and non-linearities in the hidden layer are fixed.

In the ROLS-AWS algorithm, we will show that by combining the QR Decomposition Recursive Least Squares algorithm (QRD-RLS) and the forward subset selection method, we can create a Recursive Orthogonal Least Squares learning algorithm that can automatically select the centers and the weights. We prove that, if $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are the old and new weight vectors, respectively, then the algorithm will select the optimal new weight vector $\mathbf{x}_{i+1}$ given the old weight vector $\mathbf{x}_i$. We also show that this sub-optimal solution is the same solution obtained by the batch forward selection method. This algorithm not only allows the Gaussian network to learn recursively in time but also adds necessary centers and weights when needed. Furthermore, the weight size is kept to a minimum, as the RBF network begins with just a bias, but grows in size with useful centers and weights that are selected sub-optimally.

Radial Basis Function (RBF) Network

The RBF network considered in this paper is a standard two-layered neural network with Gaussian non-linearity in the hidden layer and linear transfer function in the output layer. The output of the RBF network is computed according to the following linear equation:

$$\hat{y} = f(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{w}_i \mathbf{A}_i(-\|\mathbf{p} - \mathbf{c}_i\|^2/\sigma^2) + b \qquad (1)$$

where

σ	is the width of the Gaussian function;
$\mathbf{p}$	is the network input vector;
$\mathbf{c}_i$	is the center vector of the i^{th} node;
$\|.\|$	is the Euclidean norm;
N	is the number of nodes;
$\mathbf{A}_i$	is the Gaussian node output;
$\mathbf{w}_i$	is the i^{th} weight;
b	is the bias.

Since a multi-output RBF network can always be separated into a group of single-output RBF networks, we will consider the single-output case only. The Gaussian function width σ is fixed, while the centers $\mathbf{c}_i$ are selected from the data points. Every time a data point is presented to the network, the network creates a center based on that data point. These centers become potential centers that may be used by the RBF network. Let $\underline{\mathbf{A}}$ denote all possible centers presented by the data points, then

$$\underline{\mathbf{A}} \in \{\mathbf{A}, \bar{\mathbf{A}}\} , \tag{2}$$

where $\mathbf{A}$ contains the centers selected by the RBF network and $\bar{\mathbf{A}}$ contains the centers not selected by the RBF network. Also, note that when a new center, $\mathbf{a}_i$, is added to $\underline{\mathbf{A}}$, $\mathbf{a}_i \in \bar{\mathbf{A}}$ and $\mathbf{a}_i \notin \mathbf{A}$. Only when the algorithm determines that $\mathbf{a}_i$ is the necessary center, then $\mathbf{a}_i \notin \bar{\mathbf{A}}$ and $\mathbf{a}_i \in \mathbf{A}$.

Since the centers and the Gaussian width are fixed, we can view the RBF network as a linear regression model

$$\mathbf{d} = \mathbf{A}\mathbf{x} + \mathbf{e} \tag{3}$$

where

$$\mathbf{d} = \begin{bmatrix} d(1) & \dots & d(N) \end{bmatrix}^T , \tag{4}$$

$$\mathbf{x} = \begin{bmatrix} w_1 & \dots & w_M & b \end{bmatrix}^T , \tag{5}$$

$$\mathbf{e} = \begin{bmatrix} e(1) & \dots & e(N) \end{bmatrix}^T , \tag{6}$$

$$\mathbf{A} = \begin{bmatrix} a_1(1) & a_2(1) & \cdots & a_M(1) \\ a_1(2) & & & \\ \vdots & & \ddots & \vdots \\ a_1(N) & & \dots & a_M(N) \end{bmatrix} \tag{7}$$

is the desired output vector, the parameter vector, the error vector and the regressor matrix, respectively.

Recursive Orthogonal Least Squares with Automatic Weight Selection Algorithm

In this section, we show that by combining the QR Decomposition Recursive Least Squares (QRD-RLS) algorithm with the forward subset selection algorithm, we can create an algorithm that can recursively select the centers and recursively update the weights. We call this algorithm the Recursive Orthogonal Least Squares with Automatic Weight Selection (ROLS-AWS). This algorithm consists of two steps: a time update and an order update. In the time update, when a new observation $\mathbf{a}_i^T(k)$ and a new desired response $d(k)$ are added to equation (3)

$$\begin{bmatrix} \mathbf{A}_i(k-1) \\ \mathbf{a}_i^T(k) \end{bmatrix} \mathbf{x}_i(k) = \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix} \tag{8}$$

then the updated solution $\mathbf{x}(k)$ satisfies

$$\begin{aligned} (\mathbf{A}_i^T(k-1)\mathbf{A}_i(k-1) + \mathbf{a}_i(k)\mathbf{a}_i^T(k))\mathbf{x}_i(k) = \\ \mathbf{A}_i^T(k-1)\mathbf{d}(k-1) + \mathbf{a}_i(k)d(k) \end{aligned} . \tag{9}$$

The above equation can be solved using the Recursive Least Squares (RLS) [8],[9],[10]. Although RLS is fast, it does have some numerical stability problems [3]. A more accurate solution to equation (9) can be obtained with the QR Decomposition Recursive Least Squares algorithm (QRD-RLS) in pre-array and post-array form. This algorithm has better numerical stability properties than RLS [5], [3].

The QRD-RLS algorithm begins by initializing

$$\mathbf{R}_i^T(0) = \mathbf{0} , \mathbf{g}_i(0) = \mathbf{0} \tag{10}$$

where

$$\mathbf{A}_i(k) = \mathbf{Q}_i(k)\mathbf{R}_i(k) , \tag{11}$$

$$\mathbf{g}_i^T(k) = \mathbf{x}_i^T(k)\mathbf{R}_i^T(k) \tag{12}$$

and $\mathbf{R}_i(k)$ is a upper triangular matrix.

Then, for $k = 1, 2 \dots$ calculate ,

$$\begin{bmatrix} \mathbf{R}_i^T(k-1) & \mathbf{a}_i(k) \\ \mathbf{g}_i^T(k-1) & d^T(k) \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{Q}_i(k) = \begin{bmatrix} \mathbf{R}_i^T(k) & \mathbf{0} \\ \mathbf{g}_i^T(k) & k^{-T/2}(k)x^T(k) \\ \mathbf{a}_i^T(k)\mathbf{R}_i^{-1}(k) & k^{-1/2}(k) \end{bmatrix} \tag{13}$$

where

$$x(k) = d(k) - \mathbf{a}_i^T(k)\mathbf{x}_i(k-1) , \tag{14}$$

$$k(k) = 1 + \mathbf{a}_i^T(k)\mathbf{R}_i^{-1}(k-1)\mathbf{R}_i^{-T}(k-1)\mathbf{a}_i(k) , \tag{15}$$

and

$$\mathbf{x}_i^T(k) = \mathbf{g}_i^T(k)\mathbf{R}_i^{-T}(k) \tag{16}$$

solves the weights update.

This QRD-RLS pre-array and post-array formulation presents a nice framework for solving the RLS problem. Specifically, the $\mathbf{Q}_i(k)$ is a Given rotation that operates on the elements of the input vector $\mathbf{a}_i(k)$ in the pre-array matrix. It annihilates the elements one by one to produce a block zero entry in the top block row of the post-array. The lower triangular structure of $\mathbf{R}_i^T(k)$, is preserved in its exact form before and after the transformation. By completing the annihilation on the pre-array matrix, we obtained the post-array matrix which consists of the new $\mathbf{R}_i^T(k)$ and $\mathbf{g}_i^T(k)$. The least squares weight vector $\mathbf{x}_i(k)$ is solved using the methods of back substitution with equation (16). The whole process of QRD-RLS is repeated by substituting $\mathbf{R}_i^T(k)$, $\mathbf{g}_i^T(k)$, the new observation $\mathbf{a}_i(k)$ and the new desired tar-

get $d(k)$ into the pre-array and repeat the annihilation process again. To accommodate for the order update, the QRD-RLS checks to see if an order update is necessary before it repeats the annihilation process.

For the following order update derivation, we will drop the time index k since the time is fixed. For the order update, a new center is created simultaneously when a data point is created. This data point, $\bar{\mathbf{a}}_i$, is first added to the set of centers not selected by the RBF network.

$$\bar{\mathbf{A}}(k) = \left[\bar{\mathbf{a}}_1 \ \ldots \ \bar{\mathbf{a}}_i \ \ldots \ \bar{\mathbf{a}}_M\right] \tag{17}$$

Only when the algorithm determines that $\bar{\mathbf{a}}_i$ is the center that minimizes the network error, is $\bar{\mathbf{a}}_i$ incorporated into the RBF network.

To choose which center will minimize the network error, let $\mathbf{a}_i$ be one of the center vectors that are added to $\mathbf{A}_i$,

$$\mathbf{A}_{i+1} = \left[\mathbf{A}_i \ \mathbf{a}_i\right]. \tag{18}$$

Also, let $\mathbf{x}_i$ be the solution to the normal equation

$$\mathbf{A}_i^T\mathbf{A}_i\mathbf{x}_i = \mathbf{A}_i^T\mathbf{d}, \tag{19}$$

and $\mathbf{x}_{i+1}$ be the new normal equation solution with $\mathbf{A}_{i+1}$, then the solution can be written as

$$\begin{bmatrix}\mathbf{A}_i^T\mathbf{A}_i & \mathbf{A}_i^T\mathbf{a}_i \\ \mathbf{a}_i^T\mathbf{A}_i & \mathbf{a}_i^T\mathbf{a}_i\end{bmatrix}\mathbf{x}_{i+1} = \begin{bmatrix}\mathbf{A}_i^T\mathbf{d} \\ \mathbf{a}_i^T\mathbf{d}\end{bmatrix}. \tag{20}$$

Let $\mathbf{A}_i = \mathbf{Q}_i\mathbf{R}_i$ where $\mathbf{Q}_i$ is an orthogonal matrix and $\mathbf{R}_i$ is an upper triangular matrix, then

$$\begin{bmatrix}\mathbf{R}_i^T\mathbf{R}_i & \mathbf{R}_i^T\mathbf{Q}_i^T\mathbf{a}_i \\ \mathbf{a}_i^T\mathbf{Q}_i\mathbf{R}_i & \mathbf{a}_i^T\mathbf{a}_i\end{bmatrix}\mathbf{x}_{i+1} = \begin{bmatrix}\mathbf{A}_i^T\mathbf{d} \\ \mathbf{a}_i^T\mathbf{d}\end{bmatrix}. \tag{21}$$

The sum of squares error of the above equation is

$$\mathbf{e}_{i+1}^T\mathbf{e}_{i+1} = \mathbf{e}_i^T\mathbf{e}_i - \rho_i^{-2}((\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)^T\mathbf{d})^2 \tag{22}$$

where

$$\mathbf{e}_i = \mathbf{d} - \mathbf{A}_i\mathbf{x}_i = \mathbf{d} - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{d} \tag{23}$$

and

$$\rho_i = \sqrt{(\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)^T(\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)}. \tag{24}$$

If we divide both sides of equation (22) by the desired sum of squares $\mathbf{d}^T\mathbf{d}$, which is a constant, then we obtain

$$\frac{\mathbf{e}_{i+1}^T\mathbf{e}_{i+1}}{\mathbf{d}^T\mathbf{d}} = \frac{\mathbf{e}_i^T\mathbf{e}_i}{\mathbf{d}^T\mathbf{d}} - \frac{((\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)^T\mathbf{d})^2}{(\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)^T(\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)\mathbf{d}^T\mathbf{d}}. \tag{25}$$

This has a solution that is identical to the forward subset selection algorithm [11]. The error measurement term

$$err_i = \frac{((\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)^T\mathbf{d})^2}{(\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)^T(\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i)\mathbf{d}^T\mathbf{d}} \quad 1 \le i \le M \tag{26}$$

is the error reduction ratio in [2]. Geometrically, if we compare this error reduction ratio to the definition of the principle cosine angle between two vectors $\mathbf{y}$ and $\mathbf{z}$ [12]:

$$\cos^2(\theta) = \frac{(\mathbf{y}^T\mathbf{z})^2}{\mathbf{y}^T\mathbf{y}(\mathbf{z}^T\mathbf{z})}. \tag{27}$$

We can see that the err_i measures the cosine squared angle between the projected $\mathbf{a}_i$, $\mathbf{a}_i - \mathbf{Q}_i\mathbf{Q}_i^T\mathbf{a}_i$, and the desired vector $\mathbf{d}$. Maximizing err_i will maximize the angle between the projected $\mathbf{a}_i$ and $\mathbf{d}$. Essentially, err_i provides a way to measure how much does $\mathbf{a}_i$ (the center that has not been selected) contribute to the total error in the RBF network. This calculation is applied to each center not selected $\bar{\mathbf{A}}$ by the RBF network. We are guaranteed to optimally minimize the new sum of squares error, $\mathbf{e}_{i+1}^T\mathbf{e}_{i+1}$, by choosing the $\mathbf{a}_i$ that maximizes err_i.

Furthermore, by setting a threshold value υ on this term, we can decide whether to update or not to update the order. If all the err_i values fall below this threshold value υ, then the RBF network is adequate with the present weights and no order update is necessary. On the other hand, if some err_i values fall above the threshold value υ, then that implies that the network is not adequate with the present weights and an order update is necessary. Hence, the maximum err_i is picked, and the corresponding center is incorporated into the RBF network.

Note that to obtain this error reduction calculation, we need to obtain $\mathbf{Q}_i$ from the $\mathbf{R}_i$ matrix of the QRD-RLS, by solving the following equation via back substitution,

$$\mathbf{Q}_i = \mathbf{A}_i\mathbf{R}_i^{-1}. \tag{28}$$

If an order update is necessary, (compare err_i to the threshold value υ) then we will select the $\mathbf{a}_i$ that maximizes the err_i and will include that into $\mathbf{A}_{i+1}$:

$$\mathbf{A}_{i+1} = \left[\mathbf{A}_i \ \mathbf{a}_i\right]. \tag{29}$$

Also, the new $\mathbf{R}_{i+1}$ is calculated through

$$\mathbf{R}^T{}_{i+1} = \begin{bmatrix}\mathbf{R}_i^T & 0 \\ \mathbf{a}_i^T\mathbf{Q}_i & \rho_i\end{bmatrix}, \tag{30}$$

and the new weight is updated through

$$\mathbf{x}_{i+1} = \begin{bmatrix}\mathbf{x}_i - \mathbf{R}_i^{-1}\mathbf{Q}_i^T\mathbf{a}_i\rho_i^{-2}\mathbf{a}_i^T\mathbf{e}_i \\ \rho_i^{-2}\mathbf{a}_i^T\mathbf{e}_i\end{bmatrix}. \tag{31}$$

Lastly, to complete the ROLS-AWS algorithm, we calculate

$$\mathbf{g}_{i+1}^T(k) = \mathbf{x}_{i+1}^T(k)\mathbf{R}_{i+1}^T(k). \tag{32}$$

Once equation (29), (30), (31), and (32) are updated, these equations are converted to the pre-array matrix, equation (13), of the QRD-RLS algorithm and the whole process is repeated.

In practice, this RBF network begins with a bias and grows in size by adding centers and weights into the network one

at a time. The ROLS-AWS method is very similar to the batch form OLS method. However, they do have their differences.

- The ROLS-AWS operates recursively without recalculating the whole OLS calculation.
- The potential centers in the ROLS-AWS algorithm are limited by the data points presented to the RBF network, while the batch OLS method has the full range of data points to work with at start. So, it is natural that the ROLS-AWS method selected more centers (still smaller in size) than the batch OLS.

Result

In this section, simulation results for a simple function approximation problem are presented. The RBF network will be trained to approximate the following function:

$$y(k) = \sin(k) + \cos(2k) \quad k = 0{:}\pi/20{:}8.85\pi . \qquad (33)$$

The RBF network and the ROLS-AWS have the following fixed constants $\sigma = 1$, and $\upsilon = 10^{-4}$. Also, we trained the function with the OLS algorithm for comparison.

Both results are plotted in Figure 1 and Figure 2. The circles represent the locations of the centers used by the RBF network, the pluses are the target, and the solid line is the output of the RBF network. The lower graph on each figures shows the error residual of the RBF network.

Comparing both figures, the ROLS-AWS is as good as the batch OLS algorithm. As expected, the ROLS-AWS uses a few more centers; 51 centers versus 44 centers out of a possible 178 centers.

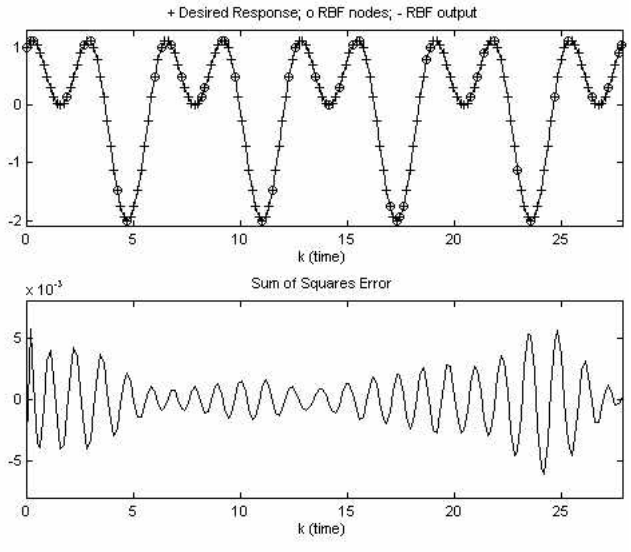

Figure 1: Batch Orthogonal Least Squares Learning

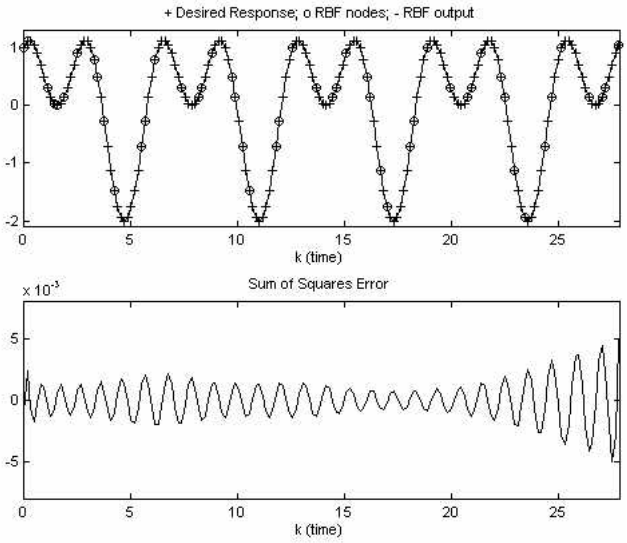

Figure 2: Recursive Orthogonal Least Squares Learning

Interestingly, the error residual for the ROLS-AWS seems to be higher near the newer data points. However, this seems to be normal, as the algorithm has not been fully adapted near the new data points.

Conclusion

This paper presents a promising algorithm that can be used to recursively train the RBF network while sub-optimally selecting the centers and weights. However, there is a limitation to this algorithm. The ROLS-AWS requires $\mathbf{A}$ to be stored in memory. As the data points increase, the size of $\mathbf{A}$ increases. Fortunately, a windowing method can be used to overcome this limitation. This windowing algorithm is currently under development. Also, regularization [13] can be easily incorporated into this ROLS-AWS.

References

[1] Chen S., Grant P.M., Cowan C.F.N. (1992). Orthogonal least squares algorithm for training multi-output radial basis function networks. *Proceedings IEE Second International Conference on Artificial Neural Networks*, Bournemouth, UK, pp.336-339.

[2] Chen S., Cowan C.F.N., Grant P.M. (1991). Orthogonal least squares algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, Vol.2, No.2, pp.302-309.

[3] Haykin S. (1996). *Adaptive Filter Theory*. Prentice Hall, 3rd Editions, NJ

[4] Broomhead D.S., Lowe D. (1988). Multi-variable functional interpolation and adaptive networks. *Complex Sys-*

tem, Vol.2, pp.269-303.

[5] Sayed A.H., Kailath T. (1994). A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, Vol.11, pp.18-60.

[6] Fabri S., Kadirkamanathan V. (1996). Dynamic structure neural networks for stable adaptive control of nonlinear systems, *IEEE Transactions on Neural Networks*, Vol.7, No.5, pp.1151-1167.

[7] Sanner R., Slotine J.-J.E. (1992). Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks*, Vol.3, No.6.

[8] Bjorck A. (1996). *Numerical Methods for Least Squares Problems*, SIAM PA.

[9] Golub G.H., VanLoan C.F. (1996). *Matrix Computations*. 3rd Edition, The John Hopkins University Press, London.

[10] Trefethen L.N., Bau D. (1997) *Numerical Linear Algebra*. SIAM PA.

[11] Miller A.J. (1990) *Subset Selection in Regression*. Chapman and Hall, NY.

[12] Overschee P.V., De Moor B. (1996) *Subspace Identification for Linear Systems. Kluwer*, Academic Publishers, MA.

[13] Chen S., Chng E.S., Alkadhimi K. (1996) Regularised orthogonal least squares algorithm for constructing radial basis function networks. *Int. J. Control*, Vol.64, No.5, pp.829-837.