HIERARCHICAL MODELING OF MULTI-SCALE DYNAMICAL SYSTEMS USING ADAPTIVE RADIAL BASIS FUNCTION NEURAL NETWORKS: APPLICATION TO SYNTHETIC JET ACTUATOR WING

A Thesis

by

HEE EUN LEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2004

Major Subject: Aerospace Engineering

HIERARCHICAL MODELING OF MULTI-SCALE DYNAMICAL SYSTEMS USING ADAPTIVE RADIAL BASIS FUNCTION NEURAL NETWORKS: APPLICATION TO SYNTHETIC JET ACTUATOR WING

A Thesis

by

HEE EUN LEE

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by

John L. Junkins (Chair of Committee)	Othon K. Rediniotis (Member)
Don R. Halverson (Member)	Walter E. Haisler (Head of Department)

May 2004

Major Subject: Aerospace Engineering

ABSTRACT

Hierarchical Modeling of Multi-scale Dynamical Systems Using Adaptive Radial Basis Function Neural Networks: Application to Synthetic Jet Actuator Wing. (May 2004)

Hee Eun Lee, B.S., Yonsei University

Chair of Advisory Committee: Dr. John L. Junkins

To obtain a suitable mathematical model of the input-output behavior of highly nonlinear, multi-scale, nonparametric phenomena, we introduce an adaptive radial basis function approximation approach. We use this approach to estimate the discrepancy between traditional model areas and the multiscale physics of systems involving distributed sensing and technology. Radial Basis Function Networks offers the possible approach to nonparametric multi-scale modeling for dynamical systems like the adaptive wing with the Synthetic Jet Actuator (SJA). We use the Regularized Orthogonal Least Square method (Mark, 1996) and the RAN-EKF (Resource Allocating Network-Extended Kalman Filter) as a reference approach. The first part of the algorithm determines the location of centers one by one until the error goal is met and regularization is achieved. The second process includes an algorithm for the adaptation of all the parameters in the Radial Basis Function Network, centers, variances (shapes) and weights. To demonstrate the effectiveness of these algorithms, SJA wind tunnel data are modeled using this approach. Good performance is obtained compared with conventional neural networks like the multi layer neural network and least square

algorithm. Following this work, we establish Model Reference Adaptive Control (MRAC) formulations using an off-line Radial Basis Function Networks (RBFN). We introduce the adaptive control law using a RBFN. A theory that combines RBFN and adaptive control is demonstrated through the simple numerical simulation of the SJA wing. It is expected that these studies will provide a basis for achieving an intelligent control structure for future active wing aircraft.

TO MY FAMILY

ACKNOWLEDGMENTS

First and foremost I am very thankful to my advisor, Professor John L. Junkins, whose knowledge is seemingly endless, for honoring me as his student during the past two years. He is the role model I have always wanted to be in professional life, as well as in personal life, because of his distinct wisdom, creativity, and warmness. He always guided and supported me in developing my research skills and writing this thesis. I thank Professor Othon Rediniotis and Professor Don R. Halverson, my committee members, for advising and inspiring me to complete my thesis.

I am also grateful to my lovely family who made my dreams come true. Their sacrifice to send me here is the most precious thing which cannot be bought.

Finally I dedicate this thesis to everyone who supported me academically, mentally and financially.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF FIGURES

Page

Page

Page

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

Background and Motivation

The desire to achieve multi-scale autonomous, intelligent, shape controllable aircraft has been attempted in many ways throughout the years. The difficult-to-model nonlinear relationship of distributed actuator commands of the resulting aerodynamics lies at the heart of the difficulties.[1-7] One possible approach is to use an adaptive neural network algorithm to model such systems, especially nonparametric and highly nonlinear behavior like that of the SJA wing.

The dynamic properties of the system to be controlled can be developed to design automatic control systems. We need to overcome the problem of the nonlinear dynamics and the uncertainties provoked by differences between desired and actual dynamic models.[8] If model parameters vary during the system operation and the system is modified its behavior in the presence of higher degree of uncertainty, this is said to be an adaptive structure. Reconfigurability of the controlled system is achieved if it can adapt to the system failures such as sensor or actuation failures in near real time- to rely on a subset of the sensors and actuators. Since it determines future performance, the adaptive controller behavior is continuous with the effectiveness of the function approximator of choice.[9]

Journal model is *Journal of Guidance, Control and Dynamics.*

Neural networks in principle enable reconfigurablity, robustness, and adaptation in that it can learn in any mode to deal with the high multidimensionality and nonlinear behaviors. Furthermore they can approximate arbitrary continuous nonlinear function with any errors. While neural networks enable a high degree of feasibility, many difficult issue arise, associated with learning sets, controllability, observability and stability.[6,7,10]

There have been several methods introduced to achieve converged input/output representation using neural networks. The distinguishing features of adaptation mentioned above in neural networks make them an appealing approach for nonlinear control.[9,11] It has been shown (Hornik, et al.,1989) that any densely measured function of practicable interest can be approximated precisely with a neural network having enough neurons, at least one hidden layer and an appropriate set of weights.[9,12] However, the efficiency of this approach, as regards the number of free parameters, speed of learning and the validity of prediction using a given network remain open questions. Furthermore, if the architecture and learning locus are fixed, its ability to represent input/output behavior is implicitly constrained and may not work well for any given problem.[13-15] There are two kinds of neural networks. One is sigmoidal neural networks which are composed of one or more layers of sigmoidal transfer functions with a fixed shape and another popular neural networks implement is radial-basis neural networks with radial basis transfer functions.

Between these two neural network approaches, we will choose the method of Radial Basis Function Network (RBFN) as a candidate best approximator. Multi-layer perceptron networks, especially with a fixed architecture, have shown many defects like

slow convergence of weights and difficulty in modeling differential responses.[10] The key feature of the simplest type of RBFN is that the output layer is only a linear combination of the hidden units. The RBFN therefore has much simpler weight updating procedure.[14] Furthermore, these basis functions lies in the input space and dominate locally near their center so heuristic localization of learning is possible. Since approximation is based on a limited number of centers, which do not have to be placed on a grid through the domain, an RBFN with adaptive logic used to place centers appears attractive for high dimensional nonparametric problems. Finally the shape, size, and orientation of local RBFs can be adapted to capture on the small number of functions the major features of the input/output behavior.[16-17] However, they do not generalize local information very well. When the large data sets are available their performance is better.

The RBFN is especially attractive in real-time approximation because we can derive globally and locally optimal solution via simple linear optimization of the weight parameters; and good estimates for the center locations and shape parameters can be deriving directly from the residual errors. The computational simplicity is excellent since only one layer is involved in supervised training; the truth results in significant advantage compared to other Neural Network functions.

Research Objectives

The ultimate goal of this Thesis is to develop an intelligent control structure integrating all the usual functions of flight controllers, along with learning and adaptation. Further, we will study this approach to adaptive control with different flight

regimes to evaluate the merits of the neural network controller. We will use the neural controller in the context of a Model Reference Adaptive Controller.[12,18-19] For the Synthetic Jet Actuator, a relationship between inputs (angle of attack (AOA), actuator frequency, slot width, Mach number etc.) and outputs (lift, drag, moment aerodynamic forces) is not obvious and it is difficult to model with existing methods, so the neural approximation approach is one possible choice.[20] Using the linearized aerodynamics, even when using static measurements, integrated flight dynamic equations in SJA are fairly well matched for the case of small angles. However, with the expanded flight envelopes being considered for future maneuvering aircraft and for the non static case of interest the problem dimensionality grows quickly and a priori learning strategies are difficult to implement due to the difficulty of dynamic experiments. This means we can expect any method to be challenged to predict the highly nonlinear input-output behaviors at high AOA, over a range of flight conditions. For this problem we will consider a Radial Basis Function Network as a candidate approach, with both a priori and real-time learning.[21] Followed by this work, as a future work, we will try to establish a closed loop model reference adaptive control law which can apply in real time starting with the approximation from offline RBFN modeling to achieve the final control objective which is tracking a prescribed trajectory, angle of attack and pitching moment using the synthetic jet actuation frequency as the control input.[22-24]

Thesis Organization

Five Chapters are included in the main body. In Chapter II we introduce a brief

history and foundations of the Synthetic Jet Actuation followed by its characteristic and recent progress in Texas A&M University. Chapter III supplies the linear and nonlinear least square fitting algorithms to compare with the efficiency of the neural network structure and its candidate algorithm. This classical approach is also well match to the data when the dimensions are less than three. Chanter IV and V, we will go further and develop a general the neural network algorithms. In Chapter IV we provide the theoretical background of the neural network and radial basis neural network for basic foundation. Chapter V contains several algorithms and this theoretical development; these are the algorithms we will apply to tune the radial basis neural network to adapt the inputs in the simulations. First part is, for more insight on neural networks, we will try a traditional sigmoidal basis functions instead of radial basis functions network algorithm for the purpose of comparison. The second part is composed of implementation of the two radial basis function network algorithms, Forward Selection with Regularized Orthogonal Least Square (FS-ROLS) Neural Networks and Resource Allocating Neural Network with Extended Kalman Filter. In this Chapter we will discuss details of the function approximation procedure that can be applied to a wide range of nonlinear problems. Chapter VI describes how the neural controller works when either pre-trained or adapted on-line. This Chapter shows the model reference adaptive controller can be used effectively in on-line learning procedure with using SJA experimental wing data and other analytical test surfaces.

In appendices, some formulation used in the thesis will be derived and the final section contains the reference for this Thesis.

CHAPTER II

INTRODUCTION TO THE SYNTHETIC JET ACTUATOR (SJA)

Synthetic Jet Actuation Fundamentals

The several distinguishing features of Synthetic Jet Actuators (SJA) for control of flow separation over lifting surfaces at high angles of attack has been a great attraction in recent years. In particular, the ability to modify the aerodynamics lift drag and moment, by up to 20%, has been demonstrated experimentally, as discussed in reference. Hingeless SJA wing experimental set up has been developed in Texas A&M Aerospace Engineering Department. The main object of SJA's experiment is control of all of the wing's parameters.

Before engaging in control law development, the first order of business is to experimentally measure the input/output behavior of the SJA airfoil, over a range of operating conditions, mach numbers, angle of attack(AOA), and actuation inputs. Both static (fixed AOA) and dynamic experiments have been conducted. As the test wing profile, NACA 0015 airfoil is used for the dynamic pitch test of the synthetic jet actuator. This model was chosen due to the ease with which the wing could be manufactured and the available interior space for accommodating the synthetic jet actuator (SJA). The chord length of the wing is 0.420 meters and a span of 0.430 meters. There are total 32 holes of pressure tapping, 16 for upper wings and 16 for lower wings. Figure 2.1 and 2.2 shows the profile of the wing and its specific scale.

Experimental Results (Rediniotis, 2000)

In the experiment, the wing angle of attack (AOA) is controlled by the sinusoidal reference signal. Oscillation frequency is given by the 0.2Hz interval from 0.2Hz to 2Hz. Freestream velocity is 25m/sec. From the surface pressures, the lift and pitching moment coefficients were calculated via integration. Static experiments results tell us that the repeatability of the pressure measurement is achieved successfully. The AOA of airfoil is forced to oscillate from 0^o to 25^o at a given frequency The experimental data collected were the time histories of the pressure distribution on the wing surface (at 32 locations). In recent experiment, more holes are added in the tail part of the SJA wing surface and the region of oscillation of AOA of airfoil is increased to from 0 to 30 degree. The data was also integrated to generate the time histories of the lift coefficient and the pitching moment coefficient. Data was collected with the SJA on and with the SJA off (i.e both with and without active flow control).

We observed that SJA showed its maximum effectiveness only in some of the separation region. In other words, its operation extends the AOA which does not exhibit the flow separation and so increases the maximum lift coefficient. However when the airfoil is forced to oscillate, the AOA at which flow separation occurs becomes lower as the AOA motion frequency increases and thus the SJA improves the aerodynamic performance at far below at which the AOA that showed the flow separation in static experiment (figure 2.3).

Unfortunately, for the present configuration, the various operating conditions of SJA do not alter the lift and moment coefficients meaningfully at the lower angle of

attack (figure 2.3) such that the sufficient control authority is not obtained for the free pivoting so as a consequence the current SJA configuration is not an effective control actuator at small AOA. However, it is observed that the SJA modifies the aerodynamic coefficients significantly at the high angle of attack, especially above the stall angle of attack (when SJA is off). Therefore, we need to consider the role of the SJA under the incursion of the wing across the high angle of attack, including in the stall region.

The variation range of center of pressure when the permissible operating parameters (SJA frequency and slot width) are changing should be large enough to cover the range of that for the entire angle of attack. Therefore, for the current setup, it does not seem possible to have the good control authority for small angle of attack just by altering the SJA operating conditions. From the recent experiment, we can know that there are several possible approaches that can be used to achieve control using SJA actuation. Thus we can try that the Neural Network algorithm which is non-parametric and suitable for poorly known input/output models, where a physical parameterization is not easily derived. Figure 2.3-2.5 show the variation of C_L and C_M, for different pitching frequencies, with and without SJA actuation. As shown especially in figure 2.3, in SJA actuation has a significant effect on the maximum value of the dynamics C_L.

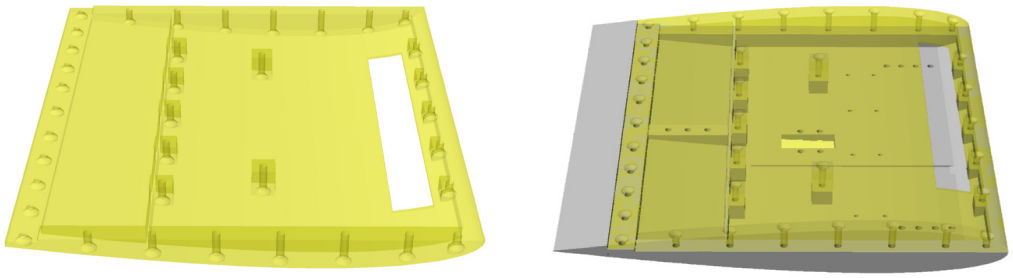

Figure 2.1 Rendered views of upper wing half panel (left) and assembled wing

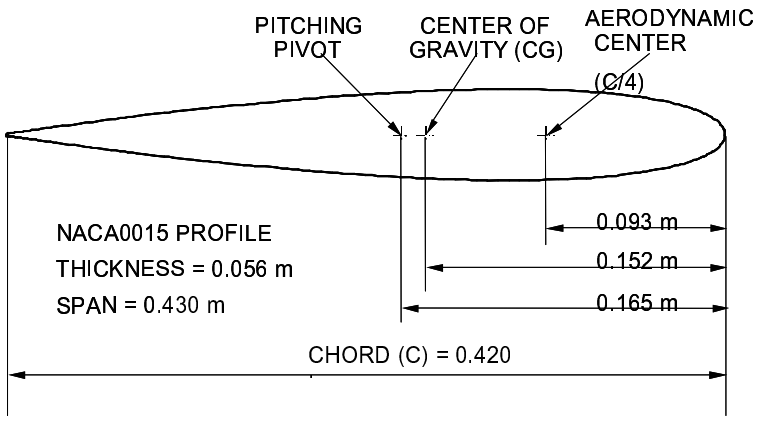

Figure 2.2 NACA 0015 wing profile

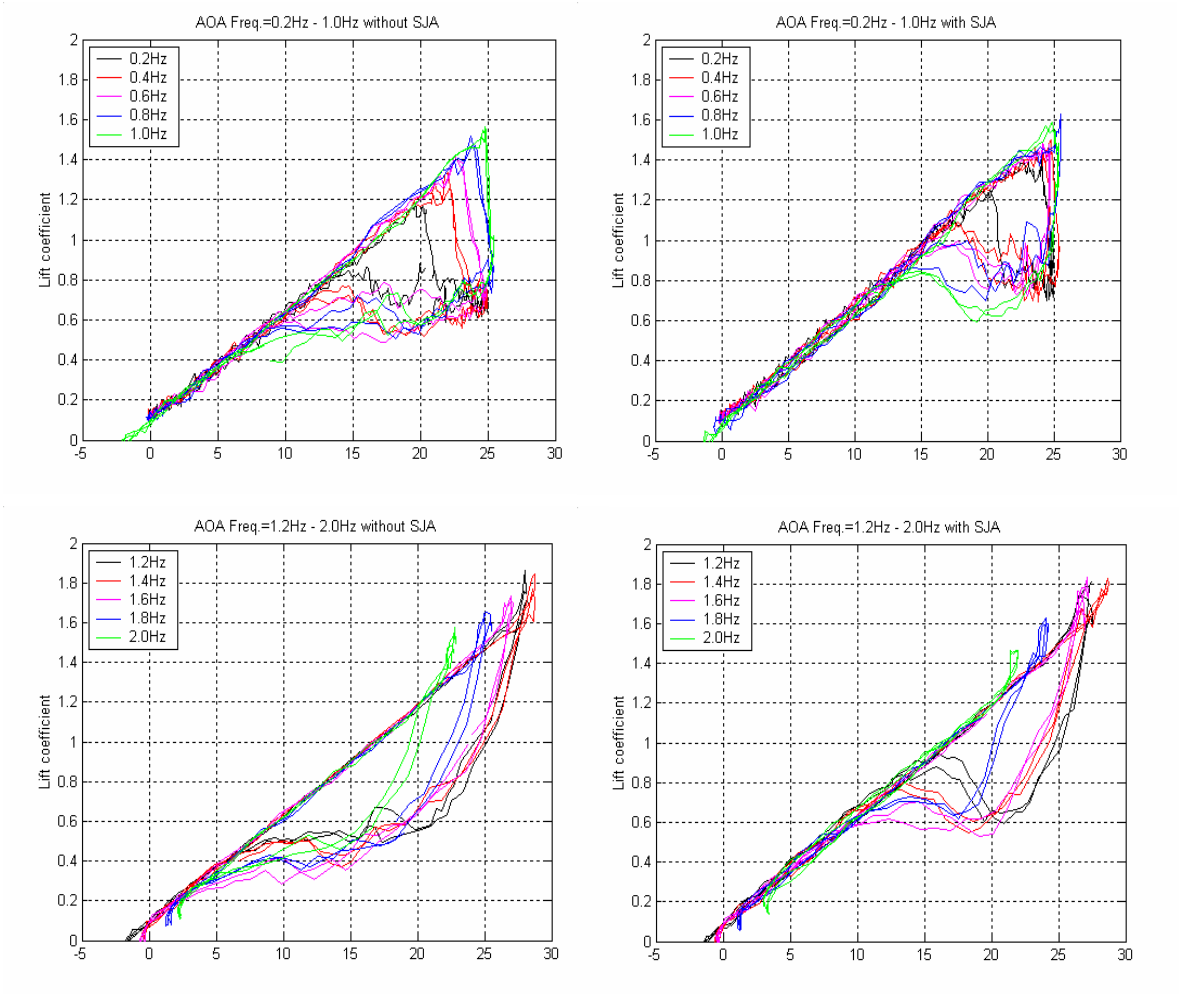

Figure 2.3 C_L variation without SJA (left) and C_L variation with SJA (right)

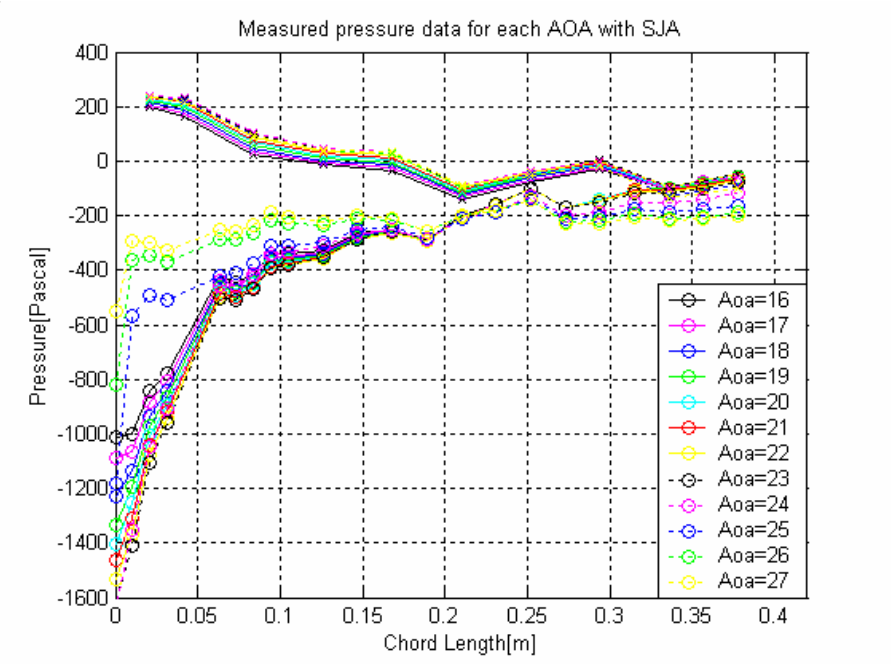

Figure 2.4 Measured pressure data for each AOA with SJA

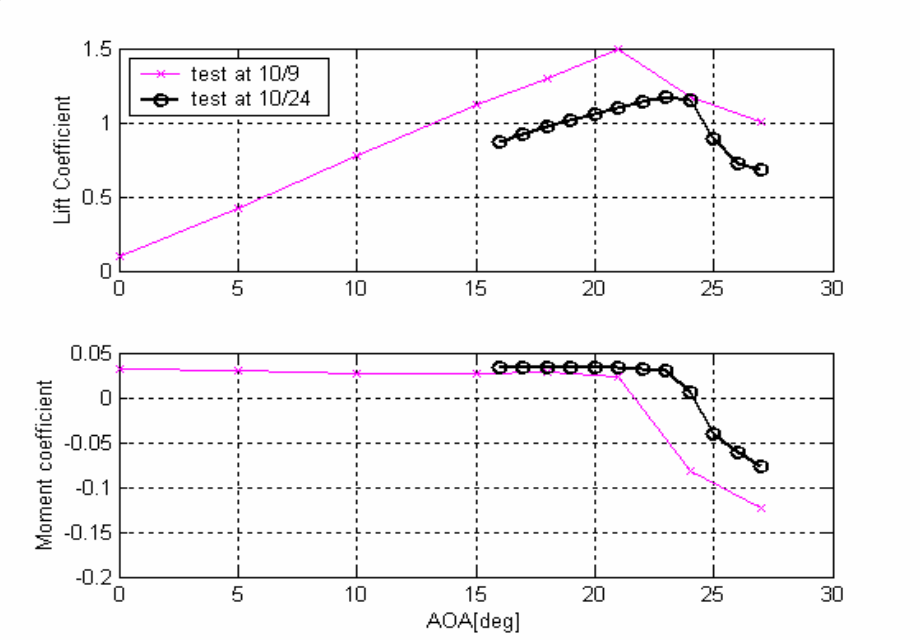

Figure 2.5 Lift and moment coefficient variation with AOA

Control Block Diagram for SJA Wing

The control objective in SJA wing is to achieve a commanded lift and moment by modulating the actuation frequency, slot width, Mach number and the slot opening rate. We note that varying SJA frequency changes, for example, C_L, C_D, and C_M, so we may anticipate difficulty achieving a prescribed change in C_L, without experiencing variations also in C_D, and C_M. By increasing the number of actuator inputs, however, we may be able to achieve prescribed $C_L, C_D,$ and C_M variations simulation only. In Chapter V we estimate the blanketed part in the block diagram in that actuator input output mapping will be determined and control law is derived in Chapter VI.

Several practical challenges are present in this experiment: the flow is unsteady, the pitch moment control authority at low angle of attack is poor, the problem is very sensitive. All of these challenges will present and the results of this thesis represent a first step in addressing them.

Figure 2.6 shows the block diagram of the final model for the intelligent wing.

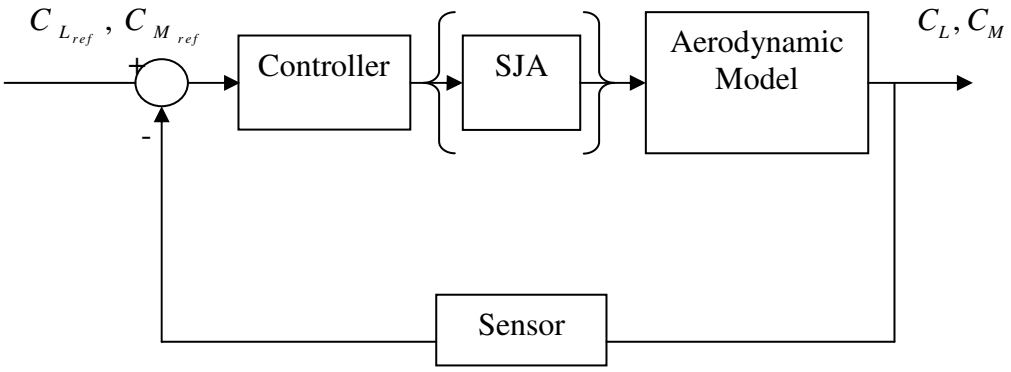

Figure 2.6 Block diagram model for control approach

CHAPTER III

LEAST SQUARES FOR FUNCTION APPROXIMATION

In this section result from Least Square method will be presented based upon various approaches to fitting the SJA data. Later on we compare this result with the neural network algorithms, to see their relative merits in input/output model accuracy and learning ability compared to the without learning algorithm.

Linear Least Squares without SJA

Using well known linear batch least square optimal estimate equation, we can estimate the each aerodynamics forces C_L, C_D, C_M by solving the normal equation.

$$\hat{\mathbf{x}} = (\mathbf{H}^{\mathbf{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathbf{T}}\tilde{\mathbf{y}} \tag{1}$$

Vector $\mathbf{H}$ is calculated here with Angle of Attack whose range is from 0 to 25 degree and SJA frequency and measurements $\tilde{\mathbf{y}}$'s are coefficients of the aerodynamic forces from the experiment. eqn(1) is useful, of course, for the batch estimation case in which all data is available, as in off-line or a priori learning. First, the key issue in any curve fitting problem, of course, is the choice of the fitting function. While we do not address "model optimality", we give two representative models in the present discussion. For the linear least squares problem, the $\hat{\mathbf{x}}$ vector is the set of coefficients linearly combine a chosen set of basis function. In a Model 1 and 2 least squares fitting will be performed without SJA effects. In figure 3.1-3.3, Model 1 is estimated using simple polynomials.

Model 2 is selected by trial and errors and simulated in figure 3.4-3.6

Model 1

$$C_L(\alpha) = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + a_4\alpha^4$$
$$C_D(\alpha) = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3 + b_4\alpha^4$$
$$C_M(\alpha) = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3 + c_4\alpha^4$$

Model 2

$$C_L(\alpha) = -a_1^*\lambda_1\alpha - a_2^*\lambda_2\alpha^2 + a_3^* e^{-\lambda_3\alpha} + a_4^* e^{-\lambda_4\alpha} + a_5^* e^{-\lambda_5\alpha} + a_6^*\cos(\lambda_6\alpha)$$
$$C_D(\alpha) = -b_1^*\lambda_1\alpha - b_2^*\lambda_2\alpha^2 + b_3^* e^{-\lambda_3\alpha} + b_4^* e^{-\lambda_4\alpha} + b_5^* e^{-\lambda_5\alpha} + b_6^*\cos(\lambda_6\alpha)$$
$$C_M(\alpha) = -c_1^*\lambda_1\alpha - c_2^*\lambda_2\alpha^2 + c_3^* e^{-\lambda_3\alpha} + c_4^* e^{-\lambda_4\alpha} + c_5^* e^{-\lambda_5\alpha} + c_6^*\cos(\lambda_6\alpha)$$

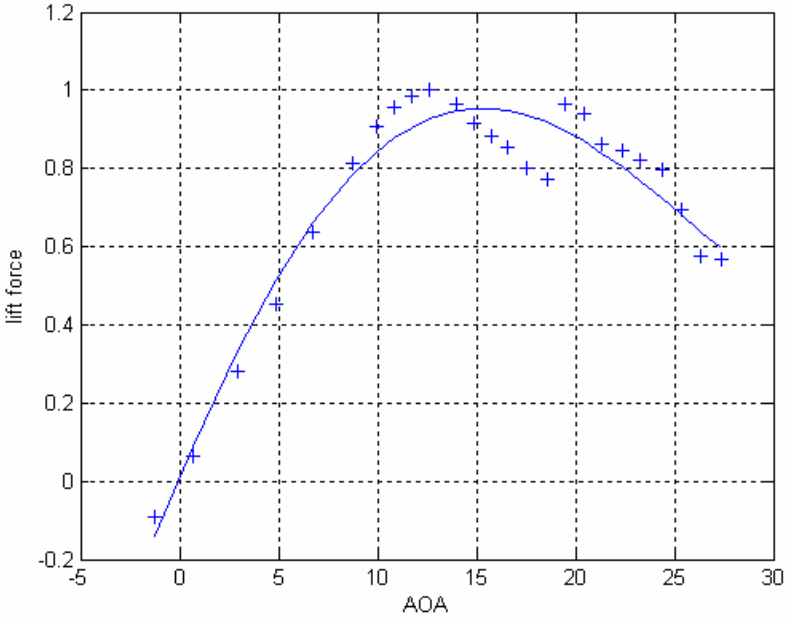

Figure 3.1 Lift coefficient fitting of Model 1.

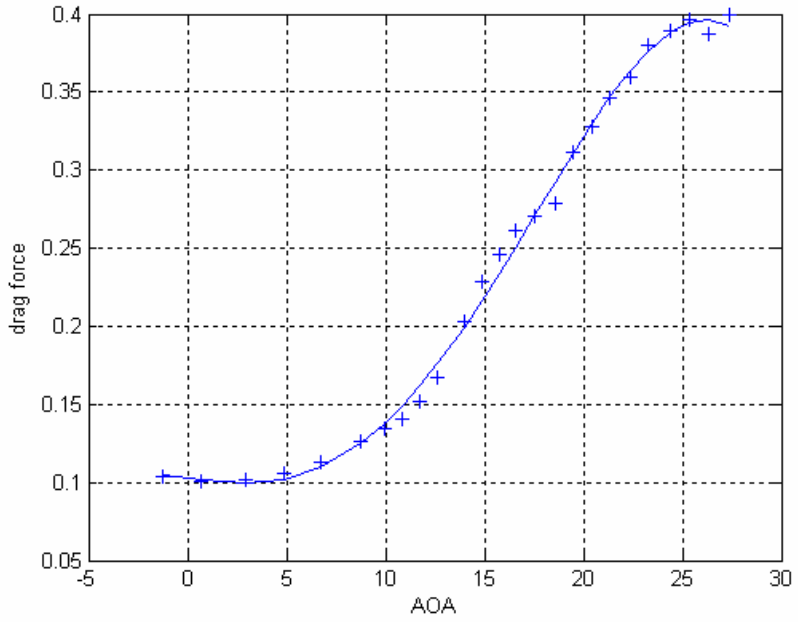

Figure 3.2 Drag coefficient fitting of Model 1.

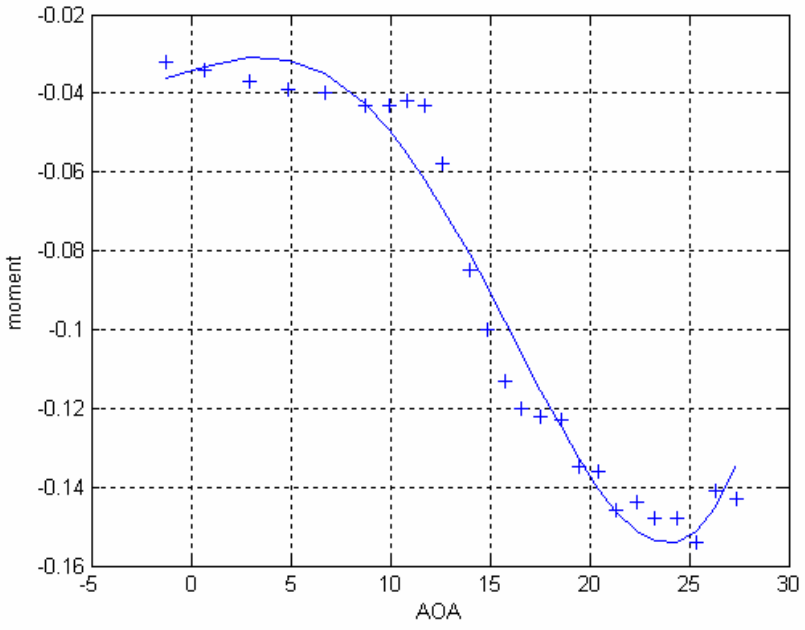

Figure 3.3 Moment coefficient fitting of Model 1.

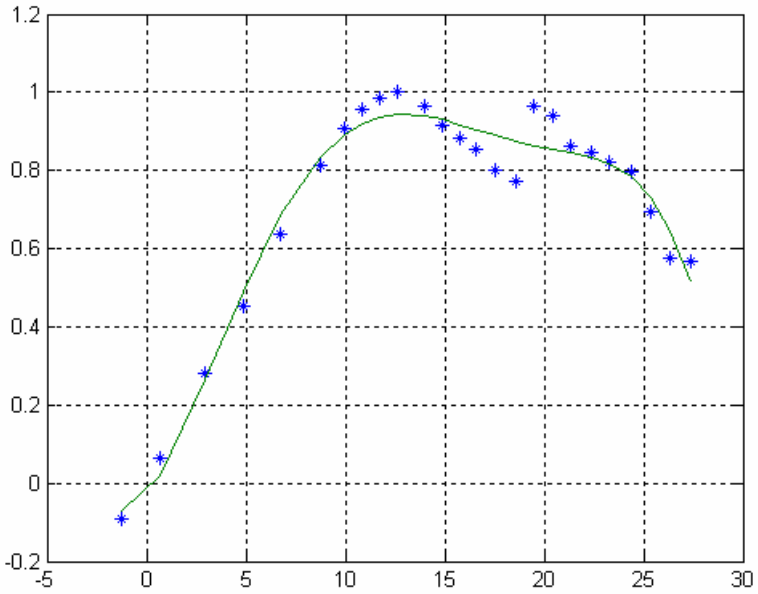

Figure 3.4 Lift coefficient fitting of Model 2.

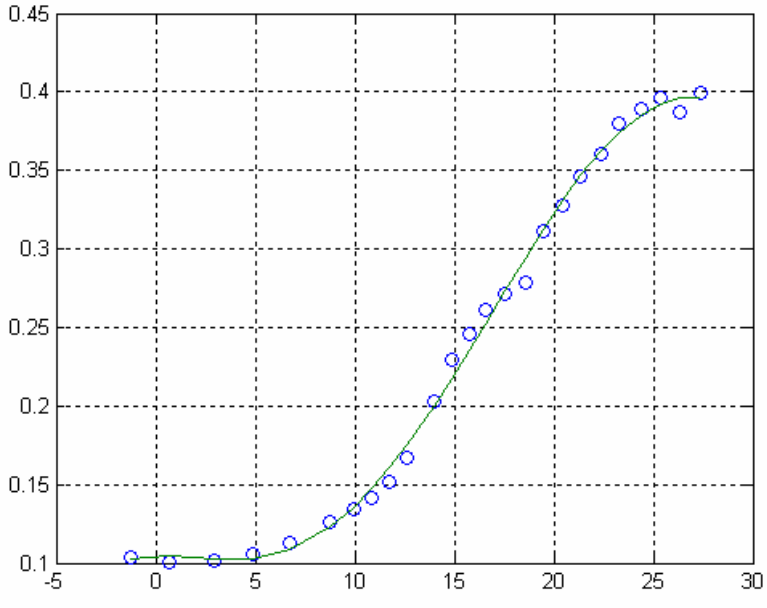

Figure 3.5 Drag coefficient fitting of Model 2

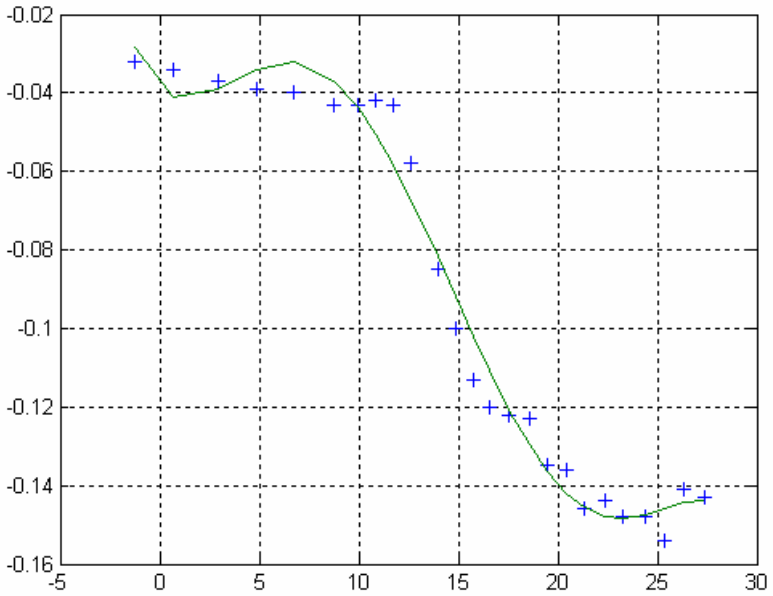

Figure 3.6 Moment coefficient fitting of Model 2

Table 3.1 Errors of linear least squares without SJA effect in Model 1.

Force	Lift force	Drag force	Moment
Force Sum	18.2610	6.0310	-2.3690
Error Sum	0.0011(0.01%)	0.1153(0.62%)	0.0017(0.07%)

Table 3.2 Errors of linear least squares without SJA effect in Model 2.

Force	Lift force	Drag force	Moment
Force Sum	18.2610	6.0310	-2.3690
Error Sum	0.0310 (0.5%)	0.2435(1.3%)	0.0303(1.2%)

Table 3.3 The coefficients of aerodynamic coefficients via linear least square technique in Model 1.

.	Lift($a_i, i = 0...4$)	Drag($b_i, i = 0...4$)	Moment($c_i, i = 0...4$)
α^4	0.000001	-0.000001	0.000003
α^3	-0.0001	0.0001	-0.000007
α^2	-0.0029	0.000003	-0.00002
α	0.1193	-0.0018	0.0016
Constant term	0.0108	0.1031	-0.0345

Observe, when unknowns other than linearly contained parameters (Model 1) must be estimated, we require a nonlinear optimization algorithm to estimate the model parameters. From the table 3.1 and 3.2, for all aerodynamic coefficient cases, the least squares with only polynomial terms results in smaller errors than with this particular choice of a nonlinear model. This implies when we consider the function with only AOA, we can successfully approximate the function with linear least squares with polynomial basis functions. However in real situation other difficulties may arise to change this conclusion, the above simply shows for two choices (of many possible) basis functions. There are many more variations that affect basis function choices the behavior of SJA which show highly nonlinear characteristic. We can observe the coefficients of aerodynamic coefficients via linear least square technique of Model 1 in table 3.3 and Model 2 in table 3.4. To get the λ optimal estimates in table 3.5, we used the MATLAB

function FMINS. The function FMIN minimizes the residuals to obtain a least square fit of data with a function of $y = \beta_1 \times e^{-\lambda(1)t} + \cdots + \beta_n \times e^{-\lambda(n)t}$.

Table 3.4 The coefficients of aerodynamic coefficients in Model 2

	Lift ($a_i^*, i = 1...6$)	Drag ($b_i^*, i = 1...6$)	Moment ($c_i^*, i = 1...6$)
$\lambda_1\alpha$	0.1084×10^{8}	-2.482×10^{8}	-2.810×10^{7}
$\lambda_2\alpha^2$	2.2810×10^{5}	-5.40029×10^{6}	-6.07601×10^{5}
$e^{-\lambda_3\alpha}$	4.4772×10^{9}	-1.0510×10^{11}	-1.1844×10^{10}
$e^{-\lambda_4\alpha}$	-2.7425×10^{9}	0.6431×10^{11}	0.72487×10^{10}
$e^{-\lambda_5\alpha}$	0.2891×10^{9}	-0.6896×10^{10}	-7.748×10^{8}
$\cos(\lambda_6\alpha)$	-2.0238×10^{9}	0.4769×10^{11}	0.5370×10^{10}

Table 3.5 λ values (nonlinear term) in Model 2.

Nonlinear term	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
value	-0.025	0.0039	0.0015	0.0017	-0.0014	0.00069

Linear Least Squares with SJA

Let us construct two another Model 3 and 4 which include SJA's actuation effects. In these Models the coupling of angle of attack α and actuation frequency ω are considered. Different from polynomial Model 3, in Model 4 nonlinear basis functions

are included. Model 4 are selected by trial and errors, and represent a “for example” nonlinear model. Figure 3.7-12 show the simulation results of these models.

Model 3

$$C_L(\alpha,\omega) = 1 + a_1'\alpha + a_2'\omega + a_3'\alpha^2 + a_4'\alpha\omega + a_5'\omega^2 + a_6'\alpha^2\omega + a_7'\alpha\omega^2 + a_8'\alpha^3 + a_9'\omega^3$$
$$C_D(\alpha,\omega) = 1 + b_1'\alpha + b_2'\omega + b_3'\alpha^2 + b_4'\alpha\omega + b_5'\omega^2 + b_6'\alpha^2\omega + b_7'\alpha\omega^2 + b_8'\alpha^3 + b_9'\omega^3$$
$$C_M(\alpha,\omega) = 1 + c_1'\alpha + c_2'\omega + c_3'\alpha^2 + c_4'\alpha\omega + c_5'\omega^2 + c_6'\alpha^2\omega + c_7'\alpha\omega^2 + c_8'\alpha^3 + c_9'\omega^3$$

Model 4

$$C_L(\alpha) = a_1''\lambda_1(\alpha+\omega) - a_2''\lambda_2(\alpha^2+\omega^2+\alpha\omega) - a_3''\lambda_3\alpha^3 + a_4''e^{-\lambda_4(\alpha+\omega)} + a_5''e^{-\lambda_5(\alpha+\omega)} + a_6''\cos(\lambda_6(\alpha+\omega))$$
$$C_D(\alpha) = b_1''\lambda_1(\alpha+\omega) - b_2''\lambda_2(\alpha^2+\omega^2+\alpha\omega) - b_3''\lambda_3\alpha^3 + b_4''e^{-\lambda_4(\alpha+\omega)} + b_5''e^{-\lambda_5(\alpha+\omega)} + b_6''\cos(\lambda_6(\alpha+\omega))$$
$$C_M(\alpha) = c_1''\lambda_1(\alpha+\omega) - c_2''\lambda_2(\alpha^2+\omega^2+\alpha\omega) - c_3''\lambda_3\alpha^3 + c_4''e^{-\lambda_4(\alpha+\omega)} + c_5''e^{-\lambda_5(\alpha+\omega)} + c_6''\cos(\lambda_6(\alpha+\omega))$$

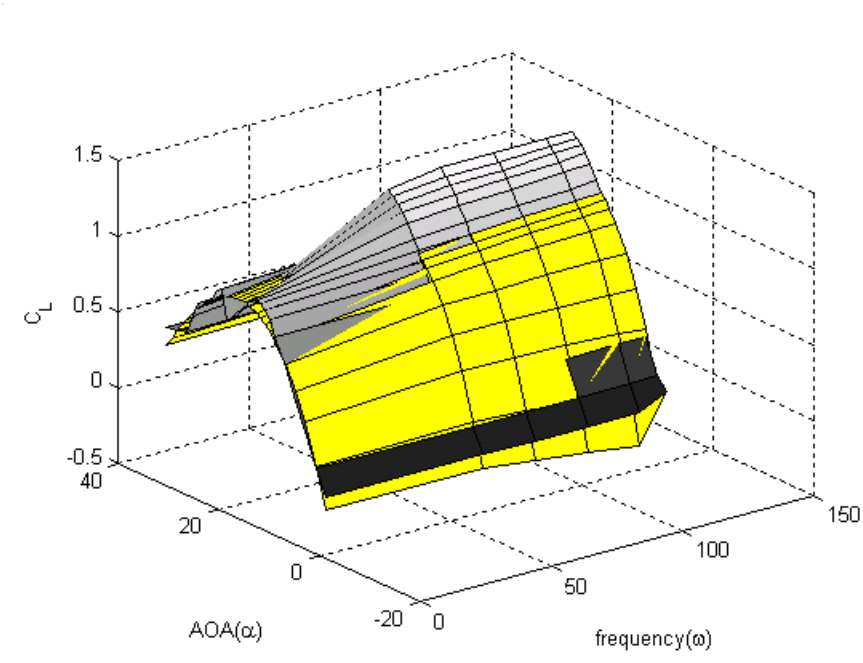

Figure 3.7 Lift coefficient fitting of Model 3

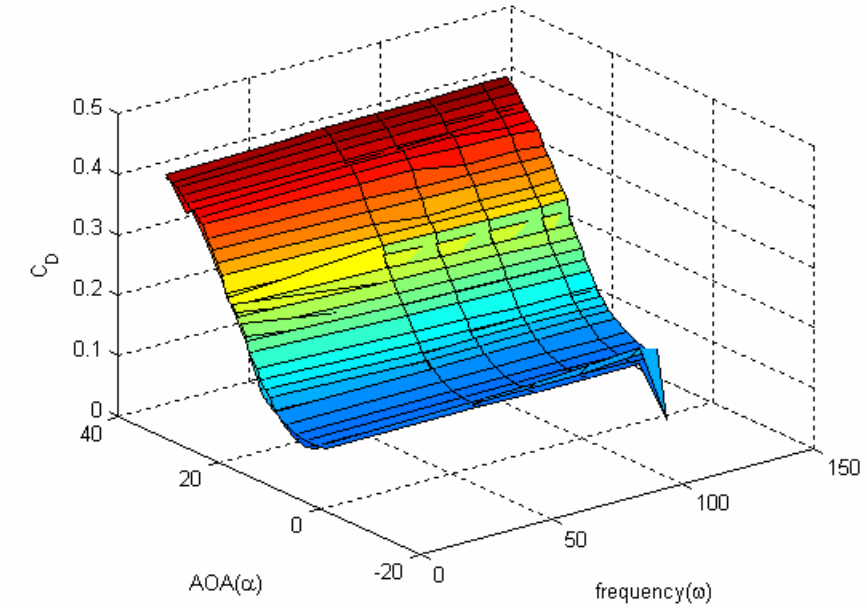

Figure 3.8 Drag coefficient fitting of Model 3.

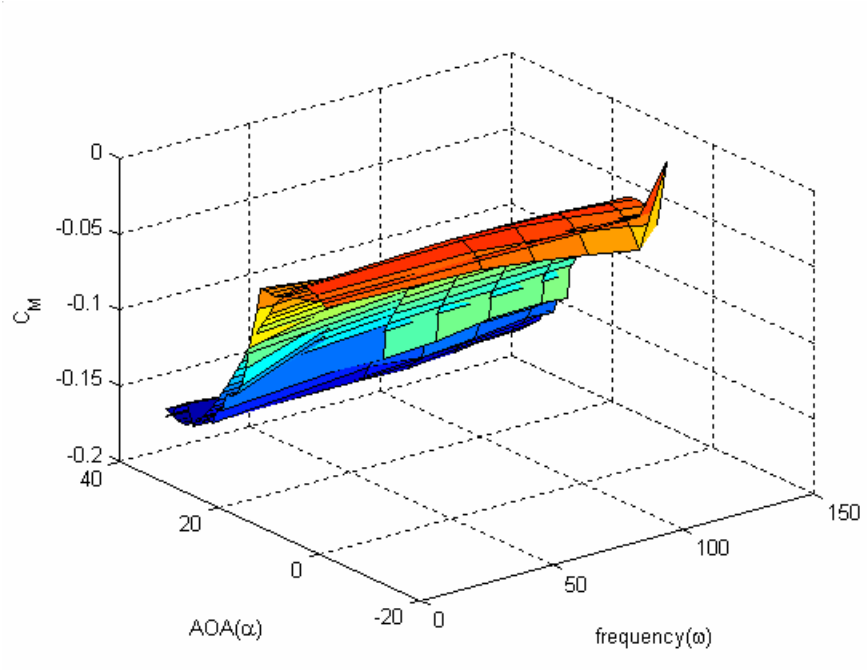

Figure 3.9 Moment coefficient fitting of Model 3.

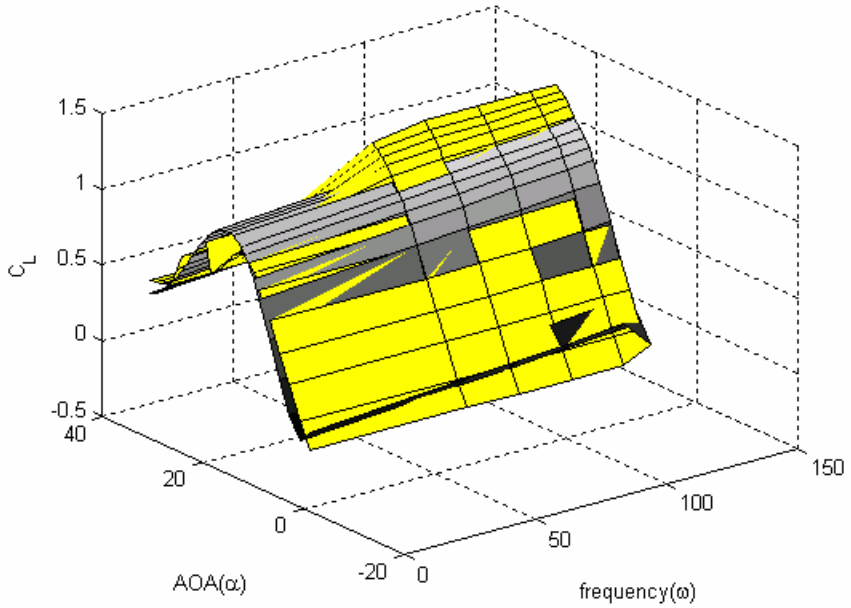

Figure 3.10 Lift coefficient fitting of Model 4.

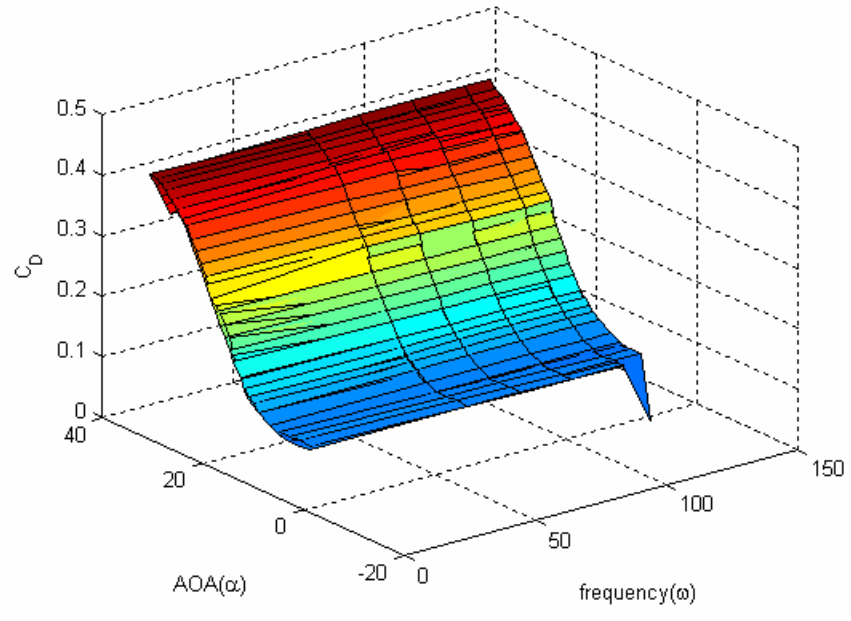

Figure 3.11 Drag coefficient fitting of Model 4.

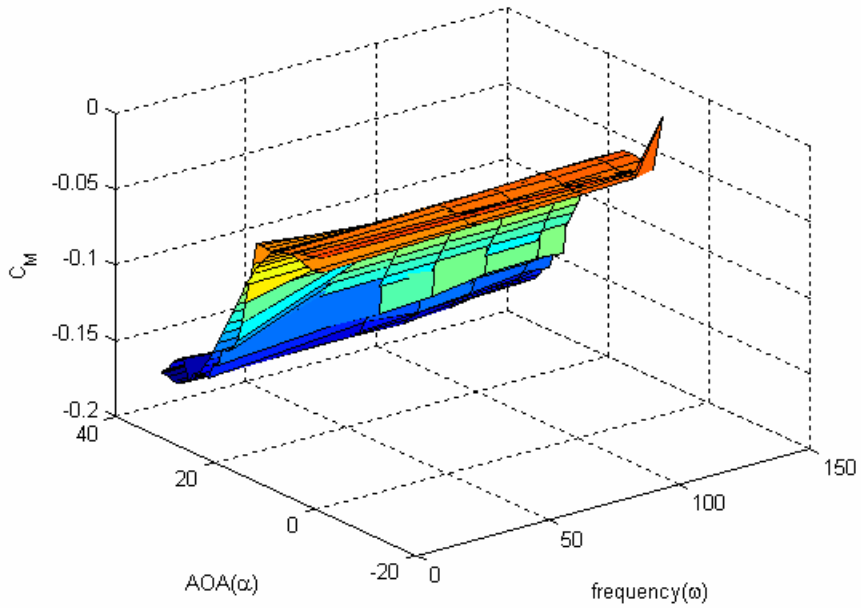

Figure 3.12 Moment coefficient fitting of Model 4

Table 3.6 Errors of linear least squares in Model 3

Force	Lift force	Drag force	Moment
Force Sum	139.09	42.8070	16.1550
Error	1.8521(1.3%)	0.0244(0.057%)	0.0144(0.08%)

Table 3.7 Errors of linear least squares in Model 4

Force	Lift force	Drag force	Moment
Force Sum	139.09	42.8070	16.1550
Error Sum	1.1970(0.86%)	0.1297 (0.301%)	0.1099(0.67%)

Table 3.8 The coefficients of aerodynamic coefficients in Model 3

	Lift($a_i', i = 0...9$)	Drag($b_i', i = 0...9$)	Moment($c_i', i = 0...9$)
ω^3	5.9386×10^{-8}	7.4906×10^{-10}	5.7227×10^{-10}
α^3	4.2373×10^{-5}	-2.9376×10^{-5}	1.5307×10^{-5}
$\alpha\omega^2$	3.9205×10^{-7}	-6.8579×10^{-8}	7.0259×10^{-8}
$\alpha^2\omega$	-9.8647×10^{-6}	1.1033×10^{-6}	-8.3860×10^{-7}
ω^2	-2.6234×10^{-5}	1.1097×10^{-6}	-1.7507×10^{-6}
$\alpha\omega$	2.8601×10^{-4}	-2.5128×10^{-5}	1.9104×10^{-5}
α^2	0.0051	0.0015	-6.8867×10^{-4}
ω	0.0015	-3.9664×10^{-5}	8.5607×10^{-5}
α	0.1315	-0.0078	0.0028
C_{L_0} , C_{D_0} C_{M_0}	-0.0106	0.1012	0.0311

Table 3.9 λ values (nonlinear terms) in Model 4

Nonlinear term	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
value	-0.025	0.0039	0.0015	0.0017	-0.0014	0.00069

To get the λ values, as explained in Model 2, we used the MATLAB function of FMINS again.

Table 3.10 The coefficients of aerodynamic coefficients in Model 4

	Lift Coefficient	Drag Coefficient	Moment Coefficient
$\lambda_1(\alpha+\omega)$	-1.2586×10^{4}	5.1885×10^{2}	-1.4139×10^{3}
$-\lambda_2(\alpha^2+\omega^2+\alpha\omega)$	8.8013×10^{7}	-4.195×10^{6}	1.0175×10^{7}
$-\lambda_3\alpha^3$	6.2835×10^{9}	-2.965×10^{8}	7.2501×10^{8}
$e^{-\lambda_4(\alpha+\omega)}$	-5.1109×10^{9}	2.414×10^{8}	-5.8982×10^{8}
$e^{-\lambda_5(\alpha+\omega)}$	8.2208×10^{7}	-3.830×10^{6}	9.4610×10^{6}
$\cos(\lambda_6(\alpha+\omega))$	-1.3428×10^{9}	6.3145×10^{7}	-1.5483×10^{8}

The errors and numerical properties of the simulation result for Model 3 and 4 are shown in table 3.6-3.10. They are calculated in the same way with model 1 and 2. Upon consideration of SJA effects, the error between the measured and assumed model is found to be much smaller in the case of Model 3 except for the lift coefficient case which means highly nonlinear characteristic of SJA has affected to the lift force more than other forces in high Angle of Attack. Qualitatively, the particular nonlinear terms related in Model 4 capture well in these experiments (less than 1% error).

As is evident, these models fit the data reasonably well however, the structure of these models is as ad-hoc and relies upon a subjective, experimental based derivation. A more general–purpose, adaptive modeling is needed, especially for highly nonlinear and higher dimensioned input/output modeling problems.

The pessimistic conclusion one might draw from the above is that there are

infinity of comparably good curve fits that can model a given set of measurements so, efficiency of computation and similar issues should play an important role in deciding which is best. More importantly, one might infer that “it would be nice”, if the approximation approach was inherently adaptive in the sense that the mathematical structure of the approximation method was learned from the data, rather than merely estimating values in an apriori assumed curve-fitting model. These observations provided much of the motivation of the work in this thesis.

CHAPTER IV

FOUNDATIONS OF RADIAL BASIS NEURAL NETWORK

What is Neural Network?

A desire to have system of mathematical modeling and pattern inference like the human brain motivated the invention of the artificial neural network. Somewhat analogous to the human brain, artificial neural networks are composed of several processing element neurons that are highly interconnected. Each neuron transforms a set of inputs to a set of desired outputs. Artificial neural networks are typically composed of interconnected units, the most basis unit is a single neuron which serves as a model neurons. Each unit converts the pattern of incoming activities that it receives into a one or more outgoing activities that it broadcasts to other units the most basis unit is a single (Rick Robinson, 2000). Typically, a neuron multiplies each incoming signal by the weight on the connection and adds together all these weighted inputs to get a quantity called the total input. And for the next step, a unit neuron uses a prescribed input-output function that transforms the total input into the outgoing signal.

The behavior of an Artificial Neural Network (ANN) depends on both the weights and the input-output function (which is also called transfer function) that is specified for the units.[15,16] To make a neural network that performs some specific task, we must choose how the units are connected to one another, and we must set the weights on the connections appropriately. Often an algorithm is selected to tune (train) the weights of the network so that some given training input/output behavior is mimiced

adequately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence. The architecture of the neural network is shown in figure 4.1.

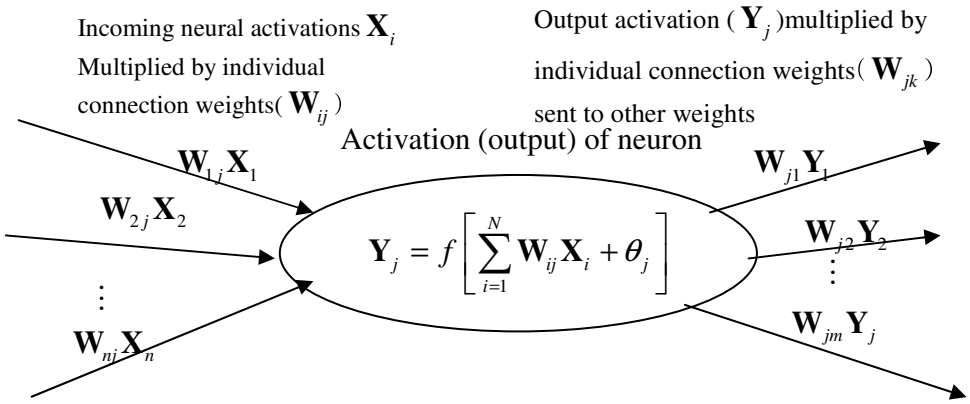

Figure 4.1 Anatomy of neural network

The most common type of artificial neural network consists of three groups, or layers, of units. The input layer is connected to a hidden layer, which is connected to output layer. The behavior of the input units represents the unrefined information that is conveyed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

To train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. During this training process, the neural network should calculate how the error changes as each weight is increased or decreased slightly according to some algorithm.

The back propagation algorithm using the gradient descent least square minimization method is the most widely used algorithm for determining weight updates. The back-propagation algorithm can compute these error corrections, although the rate of learning may be too slow in some applications. The algorithm computes each weight correction by first computing the rate at which the error changes as the activity level of a unit is changed. For output units, this rate is simply the difference between the actual and the desired output. To compute the rate for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the rates of those output units and add the products. This sum equals the rate for the chosen hidden unit. After calculating all the rates in the hidden layer just before the output layer, we can compute in like fashion the rates for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the rate at which the error changes as the activity level of a unit is changed has been computed for a unit, it is straight forward to compute the error weight for each incoming connection of the unit. The error weight is the product of the error rate and the activity through the incoming connection. For non-linear units, the back-propagation algorithm includes an extra step. Before back propagating, this error rate must be converted into the rate at which the error changes as the total input received by a unit is changed. It is not difficult to prove that the back propagation algorithm is simply an implementation of the method of gradients for nonlinear optimization. The speed of convergence is one issue, and the accuracy of convergence is a second

important issue affecting the suitability of a given ANN to a particular problem. No global claims can be model and these issues unfortunately must be addressed anew in each application.

Artificial neural network are used for such applications as pattern recognition and process control. ANN can in principle perform a host of adjustment functions in aircraft including many in the field of adaptive control. Neural networks may be able to deal with some system failures by learning a new input/output behavior. In most of these applications feedforward network with backpropagation algorithm has been used even though their defect like slow convergence rate.[4,15.] Difficult issues associated with controllability and rate of learning may exist in there problems, and every significant application requires a systematic effort to validate the feasibility of the ANN implementation. Also of significance, if the architecture is fixed apriori, then a given ANN may be "destined to fail" even with optimal training, but this can only be inferred after an unsuccessful attempt to train the ANN in a given application. The severity of these problems motivates further research, toward ANNs for which the architecture itself is adaptive. The ultimate purpose of this research is to develop a new adaptive control approach for aircraft with nonlinear actuations such as SJA's based on the use of artificial neural network. This controller is based on the inverse dynamics determination of the mathematical models taking advantage of the neural networks on-line learning capability.[25,26]

Introduction to Radial Basis Neural Network

Owing to their good globalization properties, Radial Basis Function Networks

have been broadly used for function approximation and for applying controls. Radial Basis Function Networks (figure 4.2) consist of an input layer, one hidden layer, and an output layer. Gaussian function RBF networks have a hidden layer of basis functions each of which has a response that is radially symmetric and it performs a nonlinear transform on inputs. The output layer completes the model by linearly combining the locally dominant function to get the global input/output representation of the measurements. The great advantage of RBF networks is that the learning algorithm includes the solution of a linear problem, and is therefore fast (Chris, 1991). Additional features needed to compute the algorithm including the particular learning rule used to adjust the parameter. Gaussian radial basis functions are

$$f(\vec{x}) = \sum_{\alpha=1}^{M} A^{\alpha} R^{\alpha}(\vec{x}) \tag{2}$$

$$R^{\alpha}(\vec{x}) = R(\left\| \vec{x} - \vec{x}^{\alpha} \right\| / \sigma^{\alpha}) \tag{3}$$

$$R^{\alpha}(\vec{x}) = e^{-\left\| \vec{x} - \vec{x}^{\alpha} \right\|^2 / (\sigma^{\alpha})^2} \tag{4}$$

For the one dimensional case,

$$R(x) = \exp(-\frac{x^2}{2\sigma^2}) \tag{5}$$

For multidimensional case, we propose

$$R(x) = \exp(\frac{-\mathbf{x}^{\mathbf{T}} \sum^{-1} \mathbf{x}}{2\sigma^2}) \tag{6}$$

where $\vec{x}^{\alpha}$ are centers of Gaussian function, we take the σ^{α} as the root mean square distance between a hidden neuron center and the center of its N nearest neighbors and

A^{α} are weights of the parameters. Location of the centers of the receptive fields is the crucial issue in RBFN performance. Note Σ^{-1} in eqn(6) is a positive definite matrix controlling the size, shape, and orientation of the radial basis function. In the simplest case, $\Sigma^{-1} = dig(\sigma_1^{-2}, \cdots, \sigma_n^{-2}) = \frac{1}{\sigma^2} I$. Therefore type of $R^{\alpha}(\vec{x})$, σ^{α} and center location $\vec{x}^{\alpha}$ must be carefully chosen.

The RBFN provide a highly promising interpolation approach to deal with irregularly positioned data points. Compared to the traditional Multilayer Feed-forward Network (MFN), RBFN has the following advantages; Good generalization ability, simple topological structure (RBFNs have the ability to reveal how learning proceeds in an explicit manner), fast convergence rate easily augmented by additional basis functions and insensitivity to the local minimum. The fact that these functions' arguments are the input variable, and that dominates near its center, the parameters of this network allow important heuristic or physical interpretations.

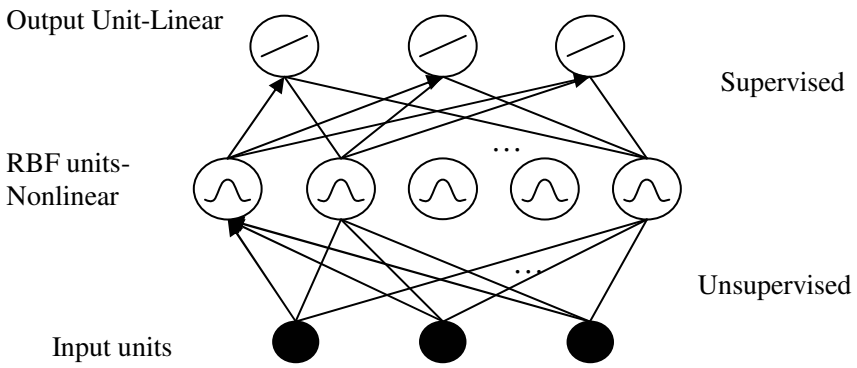

Figure 4.2 Radial basis neural network architecture

The spatially localized network controller (RBF method) can provide a better tracking performance than tradition ANNs and exhibit the potential for on-line application of learning control systems. Localization permits local learning methods with redeviced dimensionality.

A stable weight adjustment can be derived and utilized by the neural network controller. However, only the weight adjustment results in inaccurate centers and widths which make deterioration of the performance especially in the time-varying system. Recently, fully tuned RBFN begins to exhibit their great potential for accurate approximation and identification. In fully tuned RBFN, not only the weights of the output layer, but also the other parameters of the network are updated so that the local nonlinearities of the dynamic system can be captured as quickly as possible with a small number of basis functions. In contrast to traditional ANN's the architecture of the network, the number of nodes, the location of the nodes, and especially the shape of the basis functions and adapted to best represent the given system's input/output behavior. This is in contrast to merely adaptively weights in a fixed architecture with basis function whose shape and location may have little bearing upon the local behavior of the procedure at hand. In determining the proper number of hidden neurons for a given problem, the approach of RAN (Resource Allocating Network), RANEKF (Resource Allocating Network using Extended Kalman Filter to calculate parameters), MRAN (Minimal RAN) was introduced.[14,27]

A most important issue in real time application is that whatever adaptation algorithm is ultimately employed to adjust the parameters in a neural network, it must

ensure stability of the controlled process. Therefore the control structure designed and the parameter tuning rules adopted must meet the requirement of the stability and convergence for the overall system.[14,28]

CHAPTER V

DEVELOPMENTS IN NEURAL NETWORK LEARNING THEORY AND IMPLEMENTATION OF THE ALGORITHMS

Gradient-descent Based Backpropagation Solution

In this section we introduce the MNN (Multilayer Feedforward Networks) prior to discussing our proposed the RBFN. One of the popular algorithms of MNN is standard back propagation approach using the gradient descent idea and generalization of the Least Mean Squares (LMS) algorithm. The backpropagation algorithm is a supervised learning method for MNN networks typically employed with the sigmoidal activation transfer functions. The goal is to find a good input-output mapping by traning weights of the hidden units. When a new data is supplied to a network, the network provides a mapping to the output layer by using the actual input/output structure of the training set. There are many variations of the back propagation algorithm. The distinguishing feature of these learning algorithms is that all are based on recursive minimization of the error between the network output and the training data. The main idea is to establish a mapping between input vectors and the corresponding target output vectors to train a network until it can adequately approximate the nonlinear mapping function. There are generally four steps in the training process, as discussed below.

The main difference between MNNs with error back propagation and our radial basis function network is described in Table 5.1. However a goal we pursue is to determine which neural network algorithm can give the minimum least squared error so

that we can anticipate valid model-based controls. Figure 5.1 show the simple flow chart of MNN algorithm

For SJA wings, depending upon the configuration, there are many possibilities that can be selected control variables or state variable, frequency, Angle of Attack(α), derivative of AOA($\dot{\alpha}$), Slot width, Mach Number etc. To get a precise pressure distribution, we need the force balance with variation of possible all kinds of control and state parameters. As of now, the data available are the aerodynamic force coefficients with AOA(α), SJA frequency(ω) and mach number(M). The networks must capture the output aerodynamics forces on all state and control variables. Thus we only simulate with how much neural network can decrease the error as these parameters are varied. There follows the estimated lift coefficient estimate using back propagation algorithm essentially, the method of gradients.

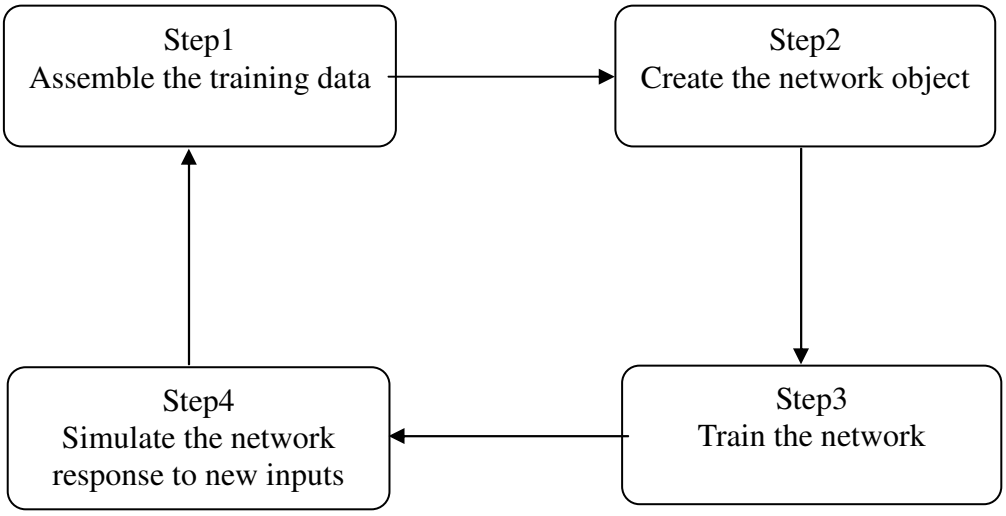

Figure 5.1 Training steps in neural network

Table 5.1 Comparison between RBFN and MNN

RBFN←→MNN	
RBFN	MNN
Single hidden layer	Single or multiple hidden layer
Non-linear hidden layer, linear output layer	Non-linear hidden layer, Nonlinear linear output layer
Argument of hidden units: Euclidean norm	Argument of hidden units: Scalar product
Universal approximation property	Universal approximation property
local approximator	Global approximator
localized learning	Global learning
adaptive basis function	Fixed basis function

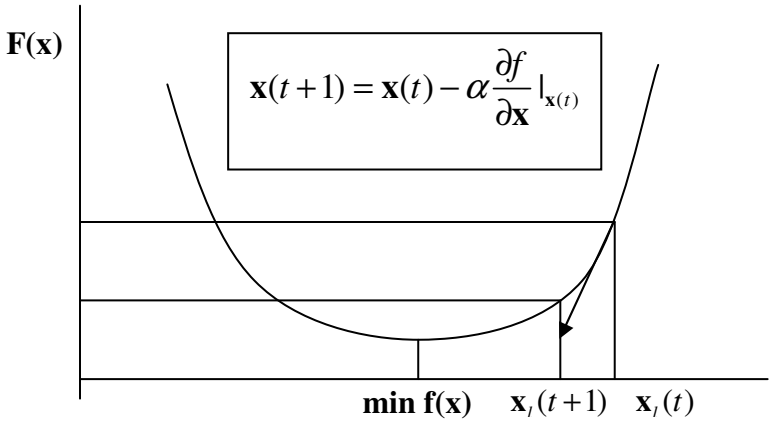

Figure 5.2 Description of gradient descent algorithm

Let us first look at the short description of gradient descent which is the main algorithm of backpropagation first. Figure 5.2 and 5.3 show simple geometrical descriptions of the whole backpropagation algorithm.

We desired to train a MNN network by gradient descent to approximate an unknown function, based on some training data consisting of pairs (**x**,**t**). The vector **x** represents inputs to the network, and the vector **t** represents the corresponding desired output target. We will describe how to compute the gradient for just a single training pattern.

We need to minimize the error function E using parameter **w** which is weight vector in neural network.

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \frac{\partial E}{\partial \mathbf{w}} |_{\mathbf{w}(t)} \tag{7}$$

$$= \mathbf{w}(t) + 2\alpha \mathbf{x}(t - \mathbf{x}^{\mathbf{T}}\mathbf{w})$$

The Learning algorithm is composed of three steps. Firstly we need to define the error criterion of the function E followed by the evaluation of E with respect to parameters **w**. Finally parameters **w** are adjusted according to derivatives.

$$E = \sum_{p=1}^{P} E_p(y_1, ..., y_c) \tag{8}$$

In the batch algorithm p will be omitted.

Assuming that we are using sum squared error for the output unit error E defined by

$$E = \frac{1}{2} \sum_{0} (t_0 - z_0)^2 \tag{9}$$

For the online algorithm (figure 5.4) the **a** is defined as

$$\mathbf{a}_i^{(l)} = \sum_j \mathbf{w}_{ij}^{(l)} \mathbf{z}_j^{(l-1)} \tag{10}$$

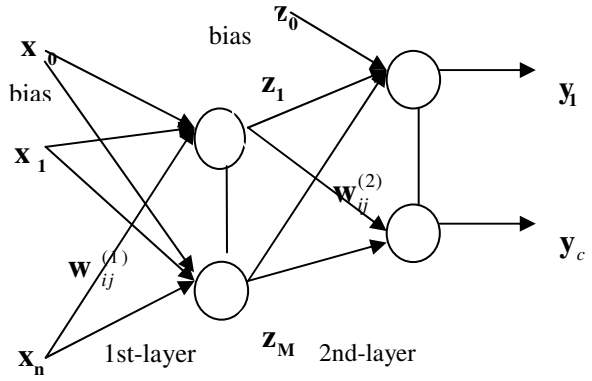

Figure 5.3 Architecture of online neural network in error backpropagation

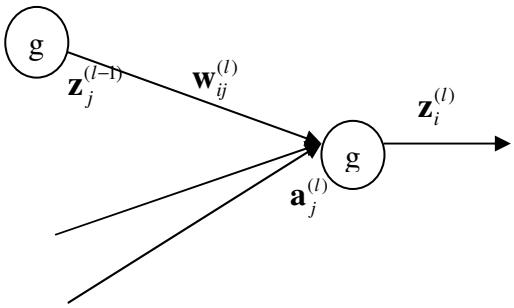

Figure 5.4 Output units for the neural network

To apply the Gradient descent algorithm let us evaluate the value of the

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)} \tag{11}$$

where the error signal $\delta_i^{(l)} = \dfrac{\partial E}{\partial a_i^{(l)}}$ and

$$\delta_i^{(l)} = \frac{\partial E}{\partial a_i^{(l)}} = \frac{\partial E}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial a_i^{(l)}} = \frac{\partial E}{\partial y_i^{(l)}} g'(a_i^{(l)}) \tag{12}$$

$\frac{\partial E}{\partial y_i^{(l)}}$ is derivative of known error criterion and $g'(a_i^{(l)})$ is known value.

For hidden units, as we see from the figure 5.5

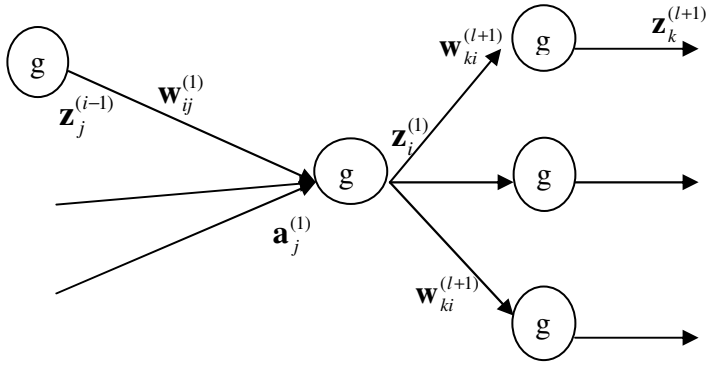

Figure 5.5 Hidden units of the neural network

The error term (δ) is expressed as a combination of errors in the next layers.

$$\delta_i^{(l)} = \frac{\partial E}{\partial \mathbf{a}_i^{(l)}} = \sum_k \frac{\partial E}{\partial \mathbf{a}_k^{(l+1)}} \frac{\partial \mathbf{a}_k^{(l+1)}}{\partial \mathbf{a}_i^{(l)}} = \sum_k \{\delta_k^{(l+1)} \frac{\partial \left(\sum_{j=(l)} \mathbf{w}_{kj}^{(l+1)} \mathbf{z}_j^{(l)} \right)}{\partial \mathbf{a}_i^{(l)}}\} \tag{13}$$

$$= \sum_k (\delta_k^{(l+1)} \mathbf{w}_{ki}^{(l+1)} g'(\mathbf{a}_i^{(l)})) = g'(\mathbf{a}_i^{(l)}) \sum_k (\delta_k^{(l+1)} \mathbf{w}_{ki}^{(l+1)})$$

Operation an input vector $\mathbf{x}^{\mathbf{k}}$ and propagate it through the network to evaluate all activations $\mathbf{a}_i^{(l)}$ and neuron outputs $\mathbf{z}_i^{(l)}$. After evaluating the error terms $\delta_i^{(o)}$ in the output layer $\delta_i^{(l)}$, back propagate error terms $\delta_i^{(l)}$ to find error terms $\delta_i^{(l-1)}$. Then, evaluate all the derivatives of $\frac{\partial E}{\partial \mathbf{w}_{ij}^{(l)}} = \delta_i^{(l)} \mathbf{z}_j^{(l-1)}$. Propotioned to derivatives and using a gradient descent scheme, weights are adjusted.

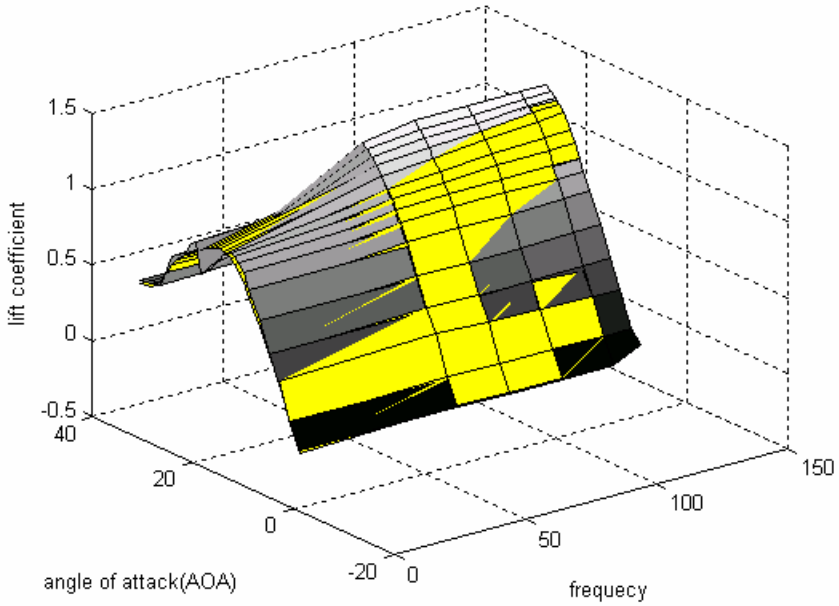

Figure 5.6 Lift coefficient approximation with back propagation algorithm

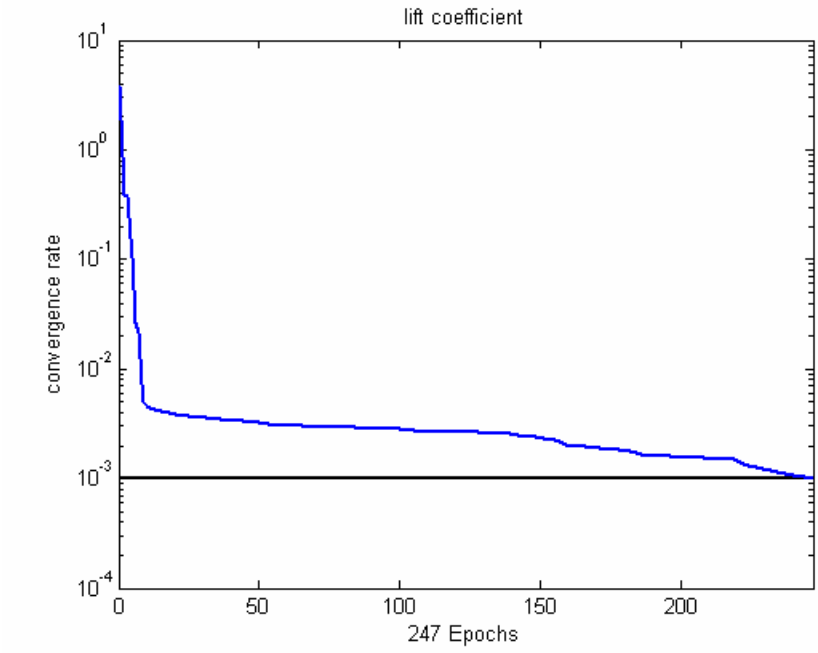

Figure 5.7 Error convergence rate of lift coefficient

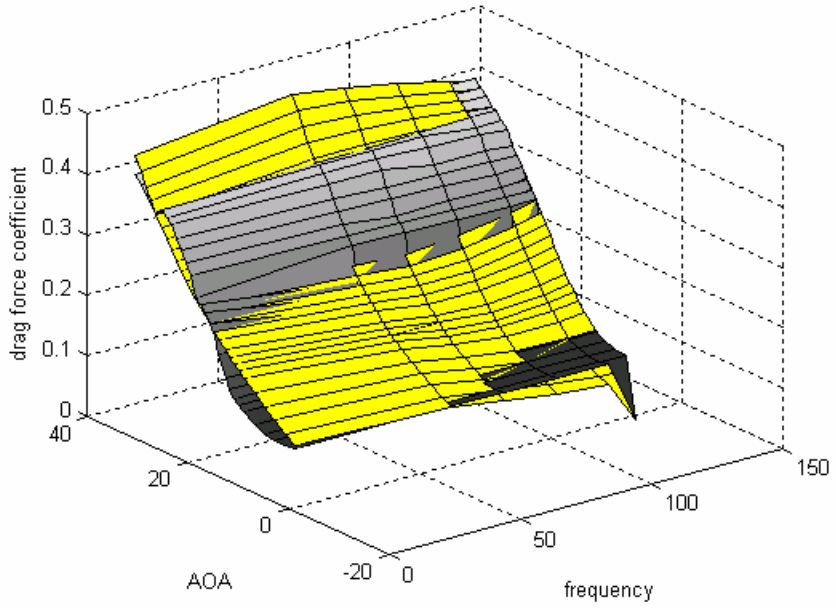

Figure 5.8 Drag coefficient approximation with back propagation algorithm

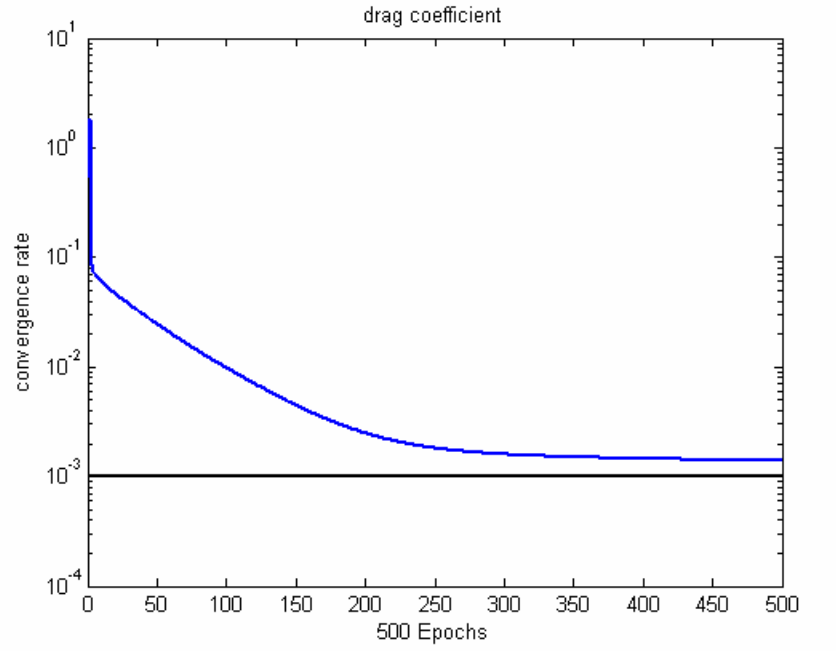

Figure 5.9 Error convergence rate of drag coefficient

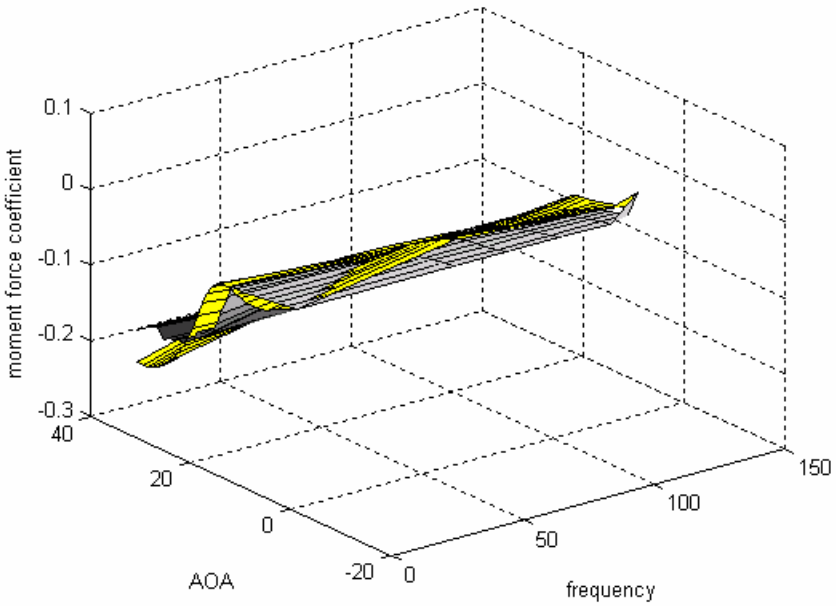

Figure 5.10 Moment coefficient approximation with backpropagation algorithm

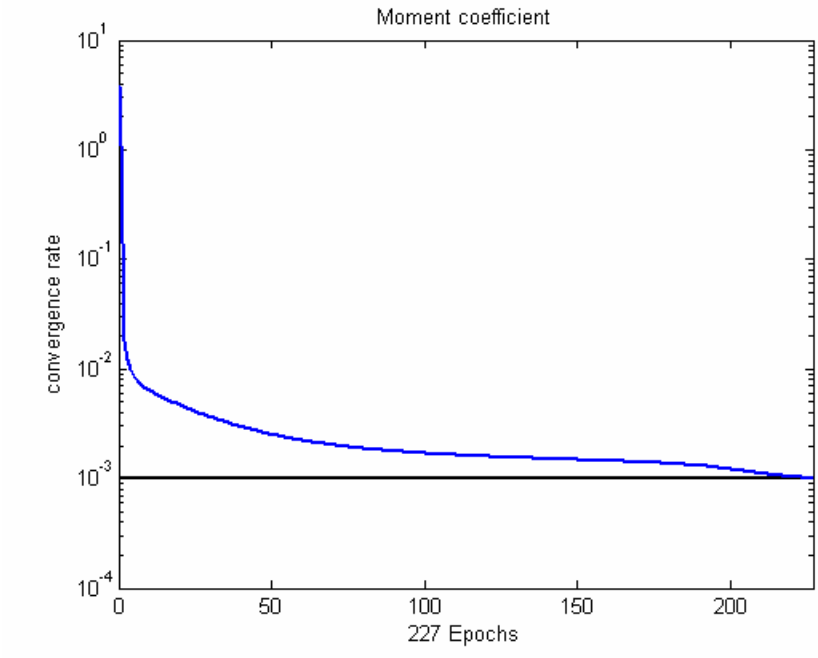

Figure 5.11 Error convergence rate of moment coefficient

Table 5.2 Properties of aerodynamic coefficient approximation with backpropagation algorithm

Force	C_L	C_D	C_M
Force Sum	139.09	42.8070	16.1550
Errors Sum	0.00868(0.0062%)	0.0294(0.069%)	0.3316(2.01%)
Mean Errors	0.0433	0.0190	0.0221
# of epoch	247	500	227
Learning rate	0.01	0.03	0.05

Figure 5.6 through 5.11 display the simulation results of lift, drag, and moment coefficient with the error backpropagation algorithm using the SJA data. From the table 5.2, we can recognize that in all cases it takes several hundred of iterations to converge to the error goal which shows the typical slow convergence rate of backpropagation for SJA. Especially, for the drag coefficient convergence is even slower rate than other coefficients. The error goal is set to 0.001 for all cases.

Forward Selection with Regularized Orthogonal Least Square (FS-ROLS) Method for Implementing RBFN (Mark 1996)

The neural network algorithm with the radial basis function is a candidate for SJA wing data fitting. In this section we evaluate a modification of this concept by adding new subset (which is Gaussian) to the design matrix where the error is biggest

until finding the best center. These modifications are motivated by problem noted in the literature (Mark 1996). In more detail, these four algorithms are combined; forward selection, orthogonal least squares, regularization (using a regularization parameter λ), these last two methods are to avoid the over-fitting and poor generalization. Finally generalized cross validation is used for choosing the best variance. Over-fitting occurs when the error variance becomes too large, and the model fits the noise in the training data and hence captures the underlying function poorly (Irwin, 1995). The motivation of selecting regularization is from the fact that it is connected to the minimal variance estimation with apriori estimates. If λ (regularization parameter) is not zero then it tends to "damp" oscillations in convergence by giving some weight to the previous estimate as a "measurement".

Regularization involves redefining the model cost function by adding a constraint to the MSE to produced a new criteria the regularization parameter, λ, and this has to be chosen a priori or estimated from the data. Another way of avoiding over-fitting is using a smaller number of centers than the data point by selecting the center column from the full design matrix. Orthogonalization algorithm is adopted because it can speed up the computations. Orthogonalization Least Squares (OLS) method has superior numerical characteristics compared with the regular Least Squares. And these properties can be used to the Forward Selection for choosing the regressor (**H** matrix). The efficiency of the orthogonalization scheme in this manner can be observed in relative ease of computing of the eqn(37) over eqn(30).

This FS-ROLS method has several advantages over other RBFN algorithms.

Like Resource Allocating Network (RAN) algorithm, it uses output value as well as input vector and attempt to find the center location which is the most suitable for the system. It also can search a discrete space as well as continuous regime. In FS-ROLS centers are fixed but there is a procedure of selection which centers should be included in the RBFN while centers are adapted in the RAN. FS-ROLS has only one preset parameter of RBF width, and can be adapted in the on line as well as in the batch model where as while RAN can be used only in on-line process. We also model the same data using RAN with the Extended Kalman Filter algorithm (EKF) for evaluate the relative merits.

Initially, there are no additional radial basis functions permitted in the design matrix F (F is initial full design matrix with all data points); we find the maximum error which can be attached to empty subset. Now we introduce new radial basis function one by one choosing their location based upon the error matrix. Suppose that we want to estimate the function y with **H** matrix from the Radial Basis Function design.

$$y = f(\mathbf{x}) = \sum_{j=1}^{m} w_j h_j(\mathbf{x}) \tag{14}$$

We can rewrite this with vector form of

$$\mathbf{y} = \mathbf{H}\mathbf{w} + \mathbf{e} \tag{15}$$

where the regressors $\{h_j(.)\}_1^m$, coefficients $\{w_j\}_1^m$ and **H** is general design matrix (this is full design matrix and denoted by **F**($p \times m$)), **y**= $p \times 1$ matrix.

$$E = \mathbf{e}^{\mathbf{T}}\mathbf{e} + \lambda \mathbf{w}^{\mathbf{T}}\mathbf{w} \tag{16}$$

vector **e** is p unknown error and E is the error matrix. Choosing the first subset column

f_J where the sum squared error is biggest from the maximum error column J of initial design matrix **F** (full design matrix).

$$f(x_i) = \sum_{j=1}^{m} w_j h_j(x_i) = h_i^T \hat{\mathbf{w}} \tag{17}$$

where $h_j(\mathbf{x}) = \phi(\frac{\| \mathbf{x} - \mathbf{c}_j \|}{r_j})$.

$$\mathbf{f} = [h_1^T \hat{w}, h_2^T \hat{w}, ..., h_p^T \hat{w}]^T = \mathbf{H}\hat{\mathbf{w}} = \mathbf{H}\mathbf{A}^{-1}\mathbf{H}^T\hat{\mathbf{y}} \tag{18}$$

where $\mathbf{A}^{-1} = (\mathbf{H}^T\mathbf{H})^{-1}$ and we call this to variance matrix in the case of without regularization, in that $\mathbf{E} = \mathbf{e}^T\mathbf{e}$.

$$\hat{\mathbf{y}} - \mathbf{f} = \hat{\mathbf{y}} - \mathbf{H}\mathbf{A}^{-1}\mathbf{H}^T\hat{\mathbf{y}} = (\mathbf{I}_p - \mathbf{H}\mathbf{A}^{-1}\mathbf{H}^T)\hat{\mathbf{y}} = \mathbf{P}\hat{\mathbf{y}} \tag{19}$$

where $\mathbf{I}_p - \mathbf{H}\mathbf{A}^{-1}\mathbf{H}^T = \mathbf{P}$. We call this **P** matrix as a projection matrix.

$$\hat{S} = (\hat{\mathbf{y}} - \mathbf{f})(\hat{\mathbf{y}} - \mathbf{f}) = \hat{\mathbf{y}}^T \mathbf{P}^T\mathbf{P}\hat{\mathbf{y}} = \hat{\mathbf{y}}^T \mathbf{P}^2 \hat{\mathbf{y}} \tag{20}$$

Adding a new basis function, the $(m+1)^{th}$, to a model which already has *m* basis function

$$\mathbf{H}_{m+1} = [\mathbf{H}_m \quad \mathbf{h}_{m+1}], \tag{21}$$

where $\mathbf{h}_{m+1} = [h_{m+1}(\mathbf{x}_1), h_{m+1}(\mathbf{x}_2), ..., h_{m+1}(\mathbf{x}_p)]^T$.

The new variance matrix is $\mathbf{A}_{\mathbf{m+1}}^{-1}$ with regularization.

$$\begin{aligned}\mathbf{A}_{m+1} &= \mathbf{H}_{m+1}{}^{\mathbf{T}}\mathbf{H}_{m+1} + \mathbf{\Lambda}_{m+1} \\ &= \begin{pmatrix} \mathbf{H}_m{}^{\mathbf{T}} \\ \mathbf{h}_{m+1}{}^{\mathbf{T}} \end{pmatrix}\begin{pmatrix} \mathbf{H}_m & \mathbf{h}_{m+1} \end{pmatrix} + \begin{pmatrix} \mathbf{\Lambda}_{m+1} & \mathbf{0} \\ \mathbf{0} & \lambda_{m+1} \end{pmatrix}\end{aligned}$$

$$= \begin{pmatrix} \mathbf{A}_m & \mathbf{H}_m{}^{\mathrm{T}}\mathbf{h}_{m+1} \\ \mathbf{h}_{m+1}{}^{\mathrm{T}}\mathbf{H}_m & \lambda_{m+1} + \mathbf{h}_{m+1}{}^{\mathrm{T}}\mathbf{h}_{m+1} \end{pmatrix} \tag{22}$$

where $\mathbf{A}_m = \mathbf{H}_m{}^{\mathrm{T}}\mathbf{H}_m + \mathbf{\Lambda}_m,\ \mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_m \end{pmatrix}.$

Let us utilize the below inverse of partitioned matrix law.

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{A}_{11} + \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{\Delta}^{-1}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & -\mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{\Delta}^{-1} \\ -\mathbf{\Delta}^{-1}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{\Delta}^{-1} \end{pmatrix} \tag{23}$$

where $\mathbf{\Delta} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$.

Applying the formula for the inverse of a partitioned matrix yields

$$\mathbf{A}_{m+1}^{-1} = \begin{pmatrix} \mathbf{A}_m^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} + \frac{\mathbf{1}}{\lambda_{m+1} + \mathbf{h}_{m+1}{}^{\mathrm{T}}\mathbf{P}_m\mathbf{h}_{m+1}} \begin{pmatrix} \mathbf{A}_m^{-1}\mathbf{H}_m{}^{\mathrm{T}}\mathbf{h}_{m-1} \\ -\mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{A}_m^{-1}\mathbf{H}_m{}^{\mathrm{T}}\mathbf{h}_{m-1} \\ -\mathbf{1} \end{pmatrix}^T \tag{24}$$

where $\mathbf{P}$ is the projection matrix and $\mathbf{P}_m = \mathbf{I}_p - \mathbf{H}_m\mathbf{A}_m{}^{-1}\mathbf{H}_m{}^{\mathrm{T}}$

$$\begin{aligned} \mathbf{P}_{m+1} &= \mathbf{I}_p - \mathbf{H}_{m+1}\mathbf{A}_{m+1}{}^{-1}\mathbf{H}_{m+1}{}^{\mathrm{T}} \\ &= \mathbf{P}_m - \frac{\mathbf{P}_m\mathbf{h}_{m+1}{}^{\mathrm{T}}\mathbf{h}_{m+1}\mathbf{P}_m}{\lambda_{m+1} + \mathbf{h}_{m+1}{}^{\mathrm{T}}\mathbf{P}_m\mathbf{h}_{m+1}} \end{aligned} \tag{25}$$

Using this relationships, after attaching f_J th column (h_{m+1}) design matrix is changed to $\mathbf{H}_{m+1} = [\mathbf{H}_m \ \ \mathbf{f}_\mathbf{J}]$. Projection matrix is changed for appended subset matrix f_J

$$\mathbf{P}_{m+1} = \mathbf{P}_m - \frac{\mathbf{P}_m\mathbf{f}_\mathbf{J}\mathbf{f}_\mathbf{J}{}^{\mathrm{T}}\mathbf{P}_m}{\lambda + \mathbf{f}_\mathbf{J}{}^{\mathrm{T}}\mathbf{P}_m\mathbf{f}_\mathbf{J}} \tag{26}$$

where $\mathbf{P}_m$ is a projection matrix and $\{f_J\}_{J=1}^{M}$ is the columns of the design matrix. From the result of the least squares, an optimal weight vector can be determined.

$$\mathbf{H^T}\,\hat{\mathbf{y}} = \mathbf{H^T f} + \mathbf{\Lambda}\hat{\mathbf{w}} = (\mathbf{H^T H} + \mathbf{\Lambda})\hat{\mathbf{w}} \tag{27}$$

where $\hat{\mathbf{w}}_\mathbf{m} = (\mathbf{H}_m^T\mathbf{H}_m + \mathbf{\Lambda})^{-1}\mathbf{H}_m^T\,\hat{\mathbf{y}}$. And the minimized energy is

$$E_m^{(i)} = \mathbf{e}_m^T\mathbf{e} + \lambda\mathbf{w}_m^T\mathbf{w} = \mathbf{y}^T\mathbf{P}_m\mathbf{y} \tag{28}$$

where $\mathbf{P}_m = \mathbf{I}_p - \mathbf{H}_m(\mathbf{H}_m^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}_m^T$.

To select the best column from F, we select the criteria of $E_m^{(i)} \le E_m^{(j)}, \quad 1 \le j \le p$

which is equivalent to selecting $\mathbf{f}_i$ to maximize

$$E_{m-1} - E_m^{(i)} = \frac{(\mathbf{y}^T\mathbf{P}_{m-1}\mathbf{f}_i)^2}{\lambda + \mathbf{f}_i^T\mathbf{P}_{m-1}\mathbf{f}_i}. \tag{29}$$

Once the best column is chosen from amongst the $\{\mathbf{f}_i\}_1^p$ it is appended to the previously chosen columns to become $\mathbf{h}_m$, the last column of $\mathbf{H}_m$.

When we consider an orthogonalization, H (design matrix) can be factored into

$$\mathbf{H}_m = \tilde{\mathbf{H}}_m\mathbf{U}_m \tag{30}$$

where $\mathbf{H}_m = [\tilde{h}_1, \tilde{h}_2, ... \tilde{h}_m]$, $\mathbf{U} =$ *upper triangular matrix* . The regression problem is formed as

$$\mathbf{y} = \tilde{\mathbf{H}}_m\tilde{\mathbf{w}}_m + \mathbf{e}_m \tag{31}$$

where $\tilde{\mathbf{w}}_m = \mathbf{U}_m\mathbf{w}_m$. At the m-th step is augmented by a new column of

$$\tilde{\mathbf{H}}_{m+1} = [\tilde{\mathbf{H}}_m \quad \tilde{\mathbf{h}}_{m+1}], \tag{32}$$

Minimized energy is expressed as

$$E_m^{(i)} = \mathbf{e}_m^T\mathbf{e} + \lambda\tilde{\mathbf{w}}_m^T\tilde{\mathbf{w}} = \mathbf{y}^T\tilde{\mathbf{P}}_m\mathbf{y} \tag{33}$$

As we derived the weight vector, orthogonalized variance matrix **A** with regularization parameter λ becomes

$$\begin{aligned}\mathbf{A}^{-1} &= (\mathbf{H}_m^{\mathbf{T}}\mathbf{H}_m + \lambda\mathbf{U}_m^{\mathbf{T}}\mathbf{U}_m)^{-1} \\ &= \mathbf{U}_m^{-1}(\tilde{\mathbf{H}}_m^{-1}\tilde{\mathbf{H}}_m + \lambda\mathbf{I}_m)^{-1}(\mathbf{U}_m^{\mathbf{T}})^{-1} \\ &= \mathbf{U}_m^{-1}\tilde{\mathbf{A}}^{-1}(\mathbf{U}_m^{\mathbf{T}})^{-1}\end{aligned} \tag{34}$$

The projection matrix after the regularization changes to

$$\mathbf{P}_m = \mathbf{I}_P - \tilde{\mathbf{H}}_m(\tilde{\mathbf{H}}_m^{\mathbf{T}}\tilde{\mathbf{H}}_m + \lambda\mathbf{I}_m)^{-1}\tilde{\mathbf{H}}_m^T = \mathbf{I}_P - \tilde{\mathbf{H}}_m \begin{pmatrix} \frac{\mathbf{1}}{\lambda + \tilde{\mathbf{h}}_1^{\mathbf{T}}\tilde{\mathbf{h}}_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\mathbf{1}}{\lambda + \tilde{\mathbf{h}}_m^{\mathbf{T}}\tilde{\mathbf{h}}_m} \end{pmatrix} \tilde{\mathbf{H}}_m^T$$

$$= \mathbf{I}_{\mathbf{p}} - \sum_{j=1}^{m} \frac{\tilde{\mathbf{h}}_j\tilde{\mathbf{h}}_j^T}{\lambda + \tilde{\mathbf{h}}_j^T\tilde{\mathbf{h}}_j} = \mathbf{I}_{\mathbf{p}} - \sum_{j=1}^{m-1} \frac{\tilde{\mathbf{h}}_j\tilde{\mathbf{h}}_j^{\mathbf{T}}}{\lambda + \tilde{\mathbf{h}}_j^{\mathbf{T}}\tilde{\mathbf{h}}_j} - \frac{\tilde{\mathbf{f}}_i\tilde{\mathbf{f}}_i^{\mathbf{T}}}{\lambda + \tilde{\mathbf{f}}_i^T\tilde{\mathbf{f}}_i} = \tilde{\mathbf{P}}_{m-1} - \frac{\tilde{\mathbf{f}}_i\tilde{\mathbf{f}}_i^{\mathbf{T}}}{\lambda + \tilde{\mathbf{f}}_i^T\tilde{\mathbf{f}}_i} \tag{35}$$

We can find the orthogonalized sum squared error from the new design matrix attached J-th column as

$$\hat{S}_m - \hat{S}_{m+1} = \hat{\mathbf{y}}^{\mathbf{T}}(\mathbf{P}_m^{\mathbf{2}} - \mathbf{P}_{m+1}^{\mathbf{2}})\hat{\mathbf{y}} = \frac{(\hat{\mathbf{y}}^{\mathbf{T}}\tilde{\mathbf{f}}_{\mathbf{J}})^{\mathbf{2}}}{\lambda + \tilde{\mathbf{f}}_{\mathbf{J}}^{\mathbf{T}}\tilde{\mathbf{f}}_{\mathbf{J}}} \frac{\mathbf{2}\lambda + \tilde{\mathbf{f}}_{\mathbf{J}}^{\mathbf{T}}\tilde{\mathbf{f}}_{\mathbf{J}}}{\lambda + \tilde{\mathbf{f}}_{\mathbf{J}}^{\mathbf{T}}\tilde{\mathbf{f}}_{\mathbf{J}}} \tag{36}$$

Next we get the maximum error column to append the design matrix continuously until the certain error goal met, as we mentioned above, $\tilde{E}_m^{(i)} \le \tilde{E}_m^{(J)},\ 1 \le J \le p$.

The optimum $\tilde{\mathbf{f}}_i$ is the one which maximizes

$$\tilde{E}_{m-1} - \tilde{E}_m^{(i)} = \frac{(\mathbf{y}^T\tilde{\mathbf{f}}_i)^2}{\lambda + \tilde{\mathbf{f}}_i^T\tilde{\mathbf{f}}_i}. \tag{37}$$

So far we discuss about how to select the suitable centers for RBFN. To halt the

selection we need some criteria like a threshold. As a candidate criterion we introduce the Generalized Cross Validation. We define the criteria of selection variance by

$$\sigma^2 = \frac{1}{p}\sum_{i=1}^{p}(f_m^{(i)}(x_i) - y_i)^2 \tag{38}$$

Good generalization performance is determined by the point at which this measure reaches a minimum.[29,30] It can be derived analytically (Golub *et al* 1979) as

$$\sigma^2 = \frac{1}{p} \| (diag(\tilde{\mathbf{P}}_m))^{-1}\tilde{\mathbf{P}}_m\mathbf{y} \|^2 \tag{39}$$

In FS-ROLS product of $diag(\tilde{\mathbf{P}}_m)$ and $\tilde{\mathbf{P}}_m\mathbf{y}$ is equivalent to a mere element by element division of two p-dimensional vectors. Therefore Generalized Cross Validation is given

$$\sigma^2_{GCV_m} = \frac{1}{p}\frac{\| \tilde{\mathbf{P}}_m\mathbf{y} \|^2}{((1/p)trace(\tilde{\mathbf{P}}_m))^2} \tag{40}$$

GCV is certainly good criteria for avoiding overfit but using regularization as well can more decrease the likelihood of overfit. To get the good λ value, we can automatically calculate from the above GCV criteria (see Appendix A).

This learning algorithm is applied and simulated to the SJA wing data and it results in a significant improvement in the error convergence rate when we compare to the backpropagation algorithm and least linear squares. After few hidden units are added all the coefficients are converged within the error criteria. Mean Error of the all observation points are reasonably small which is desirable value. In figure 5.12 through 5.23, the lift, drag and moment coefficient are estimated with FS-ROLS algorithm and table 5.3 introduce errors and number of hidden units for all cases. 167 training set with

random noise and 167 test pattern used for learning.

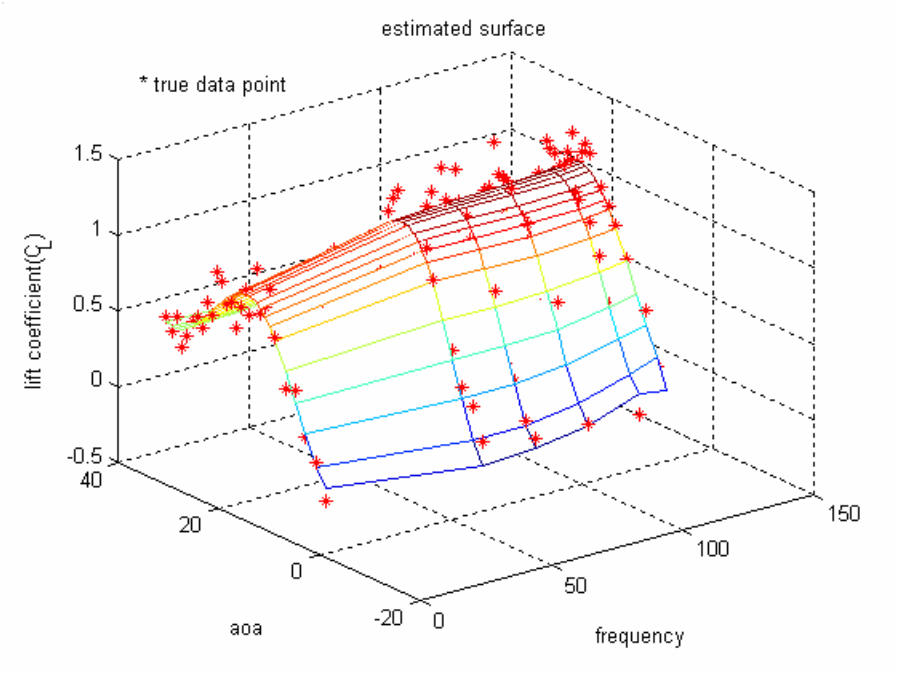

Figure 5.12 Lift coefficient approximation with FS-ROLS and true data points

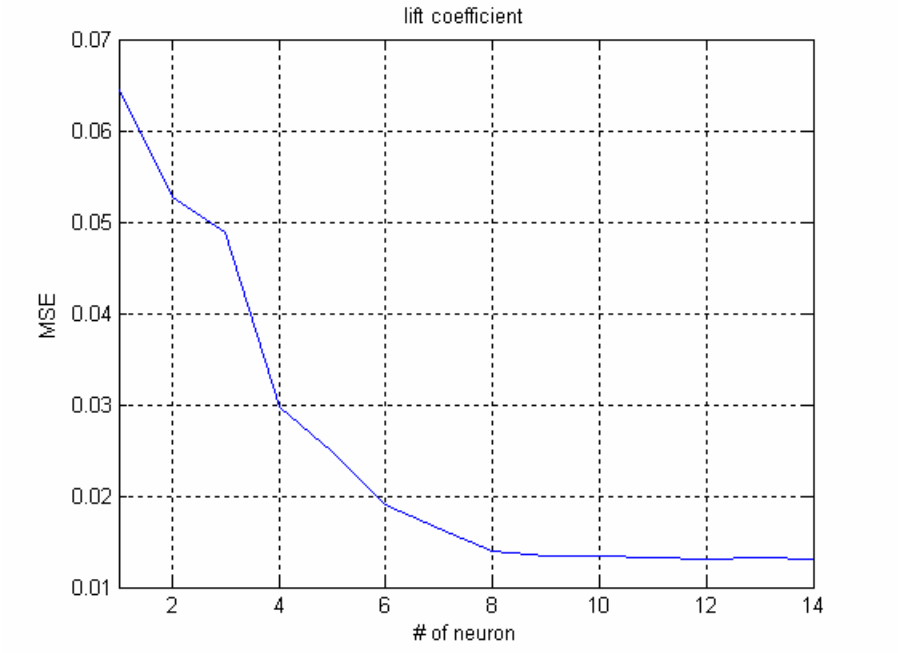

Figure 5.13 Error convergence of lift coefficient in FS-ROLS

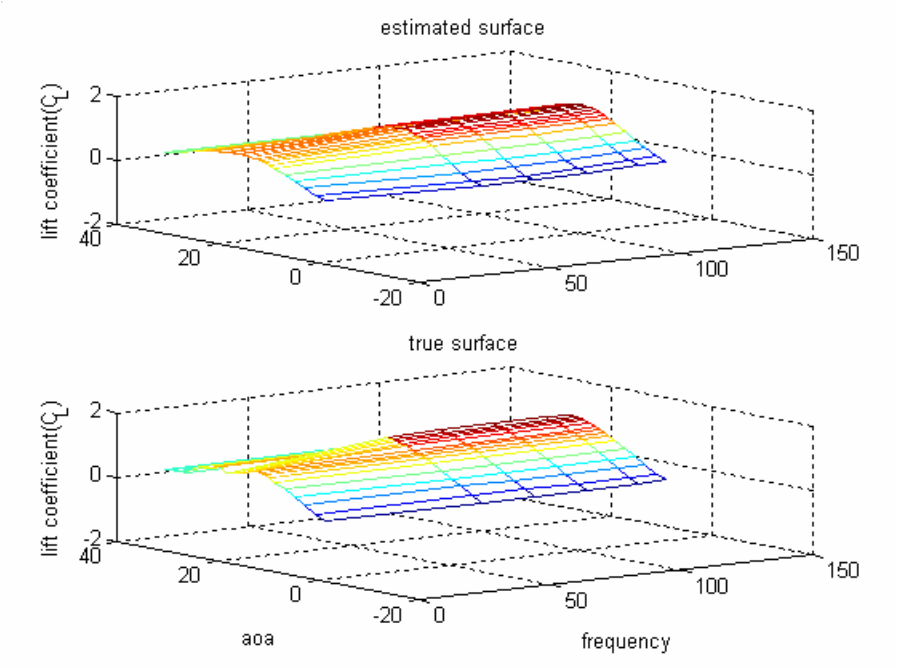

Figure 5.14 Estimated and true surface of lift coefficient

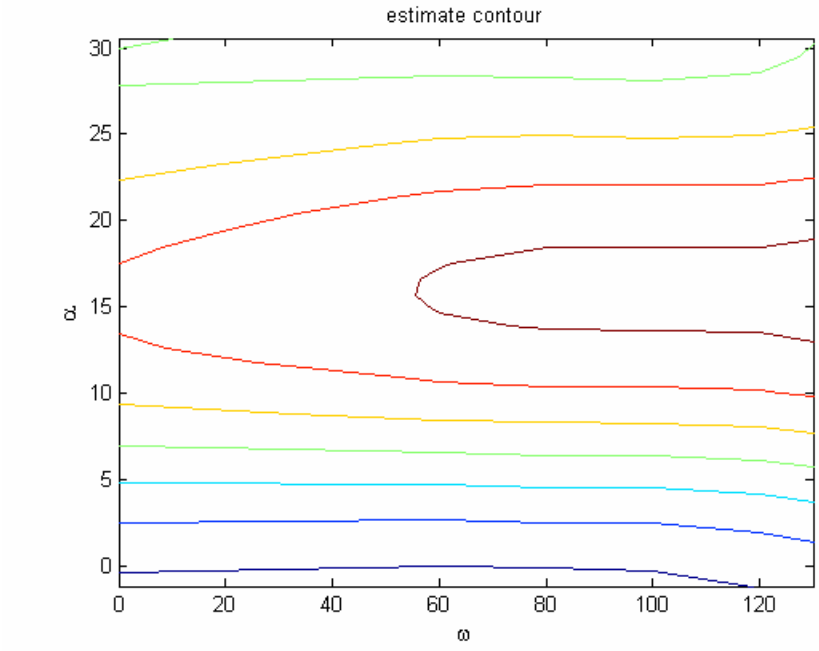

Figure 5.15 Estimated contour of lift coefficient

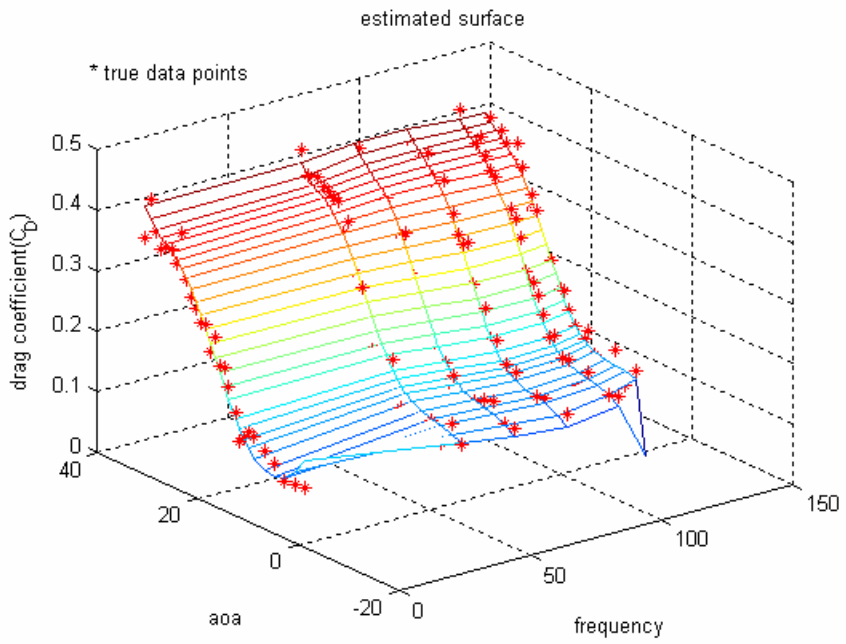

Figure 5.16 Drag coefficient approximation with FS-ROLS and true data points

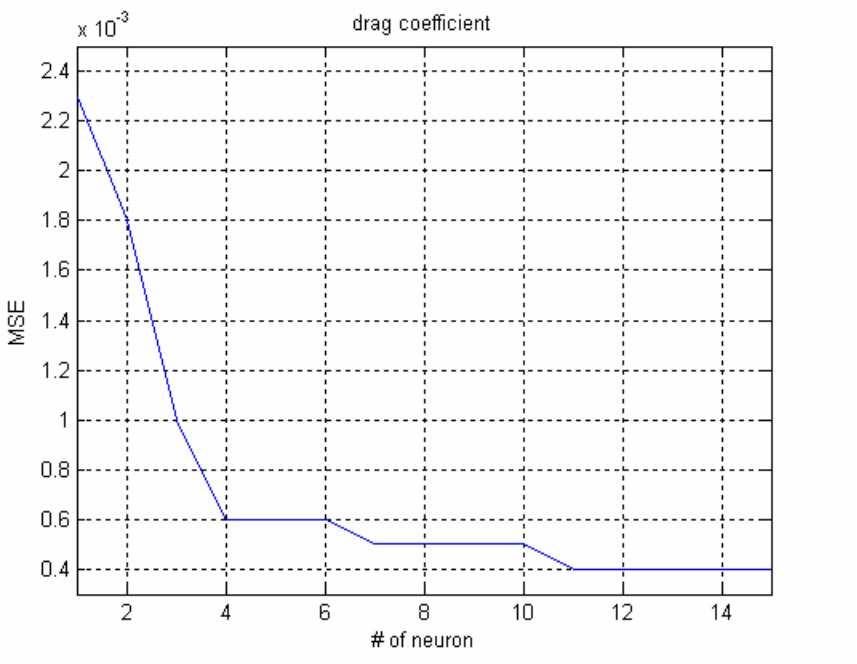

Figure 5.17 Error convergence of drag coefficient in FS-ROLS

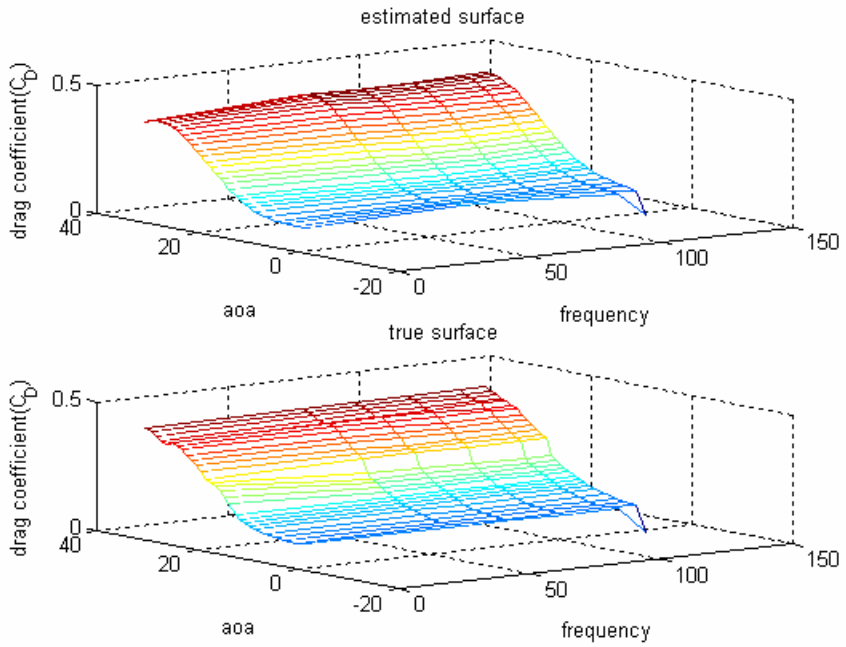

Figure 5.18 Estimated and true surface of drag coefficient

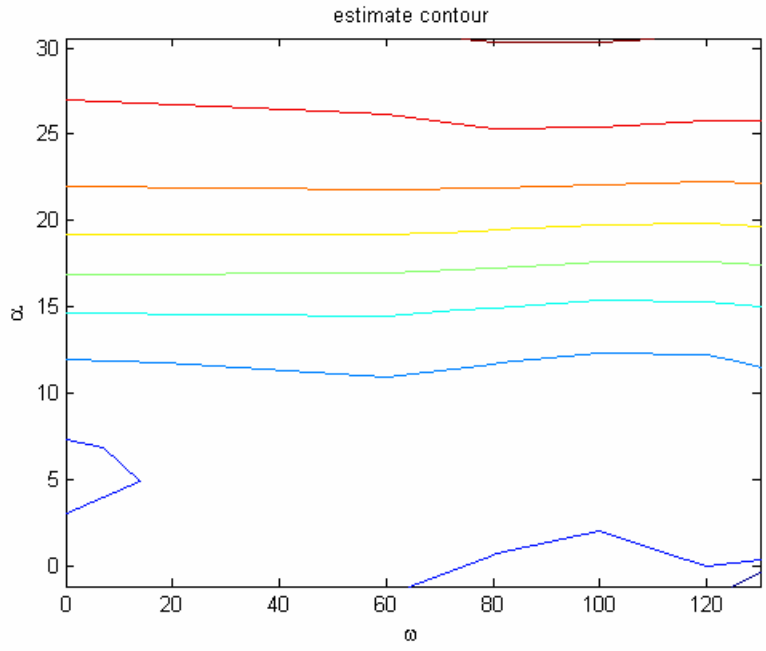

Figure 5.19 Estimated contour of drag coefficient

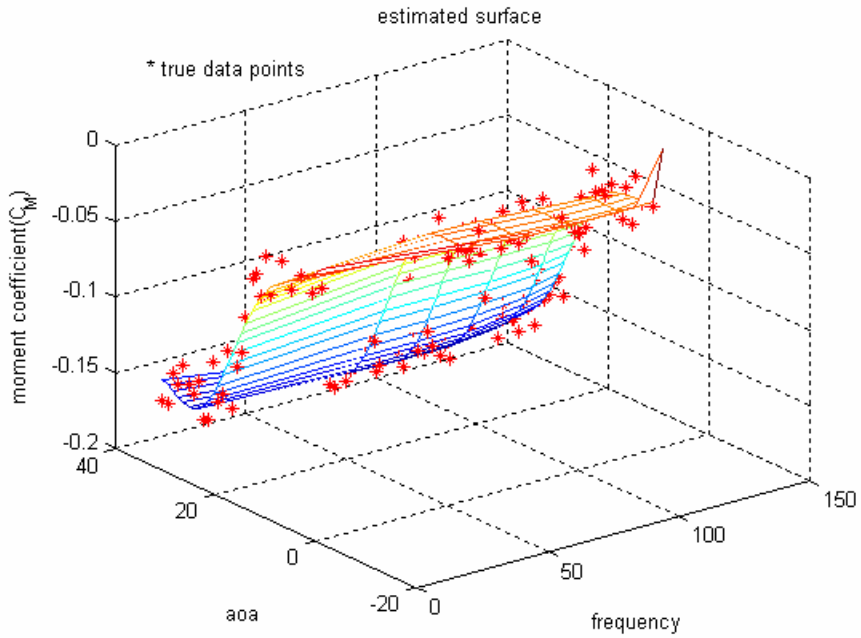

Figure 5.20 Moment coefficient approximation with FS-ROLS and true data points

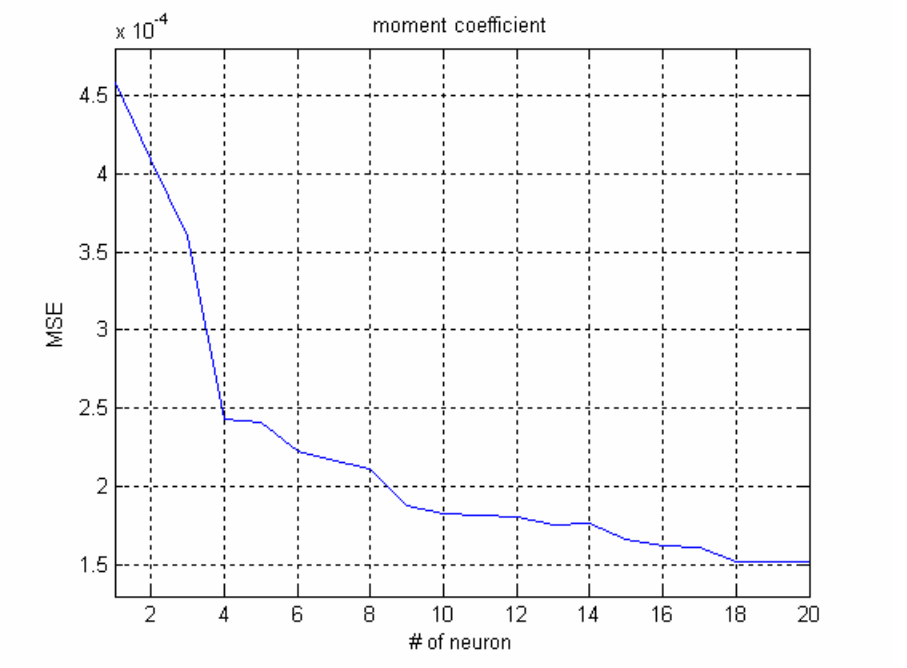

Figure 5.21 Error convergence of moment coefficient in FS-ROLS

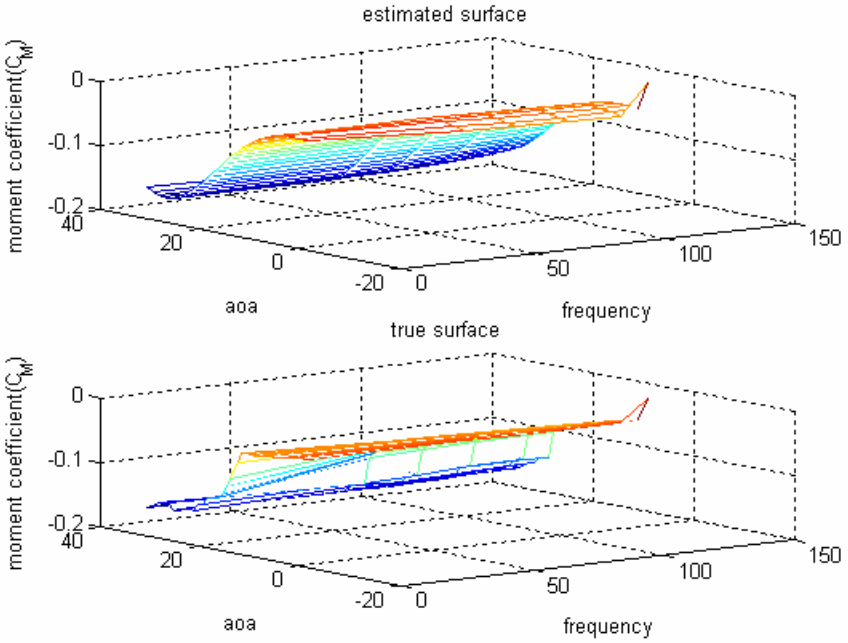

Figure 5.22 Estimated and true surface of moment coefficient

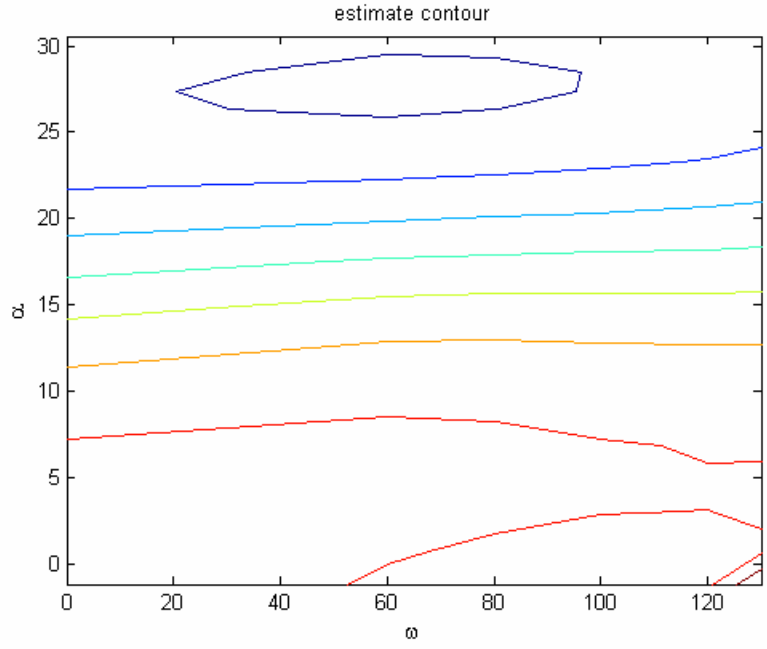

Figure 5.23 Estimated contour of moment coefficient

Table 5.3 Errors and number of hidden units in FS-ROLS

Force	C_L	C_D	C_M
Force Sum	139.09	42.8070	16.1550
Errors Sum	0.0135(0.0097%)	0.0114(0.026%)	0.0091(0.056%)
Mean Errors	0.0084	0.0047	-3.1425×10^{-4}
# of hidden Unit	14	15	18

With updated pitch moment data which means more pressure taps in tail part of the wing (originally 32 pressure taps in SJA) in the range of angle of attack from 17 to 27 degree, we can get faster convergence rate than previous data as we observe from the figure 5.24-5.28. This is pitch moment only.

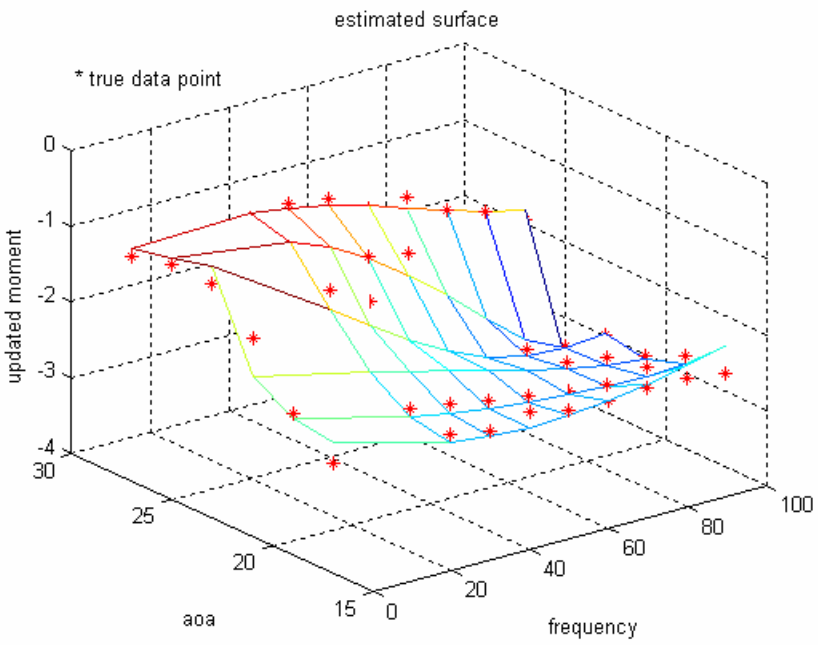

Figure 5.24 Updated pitch moment approximation with FS-ROLS

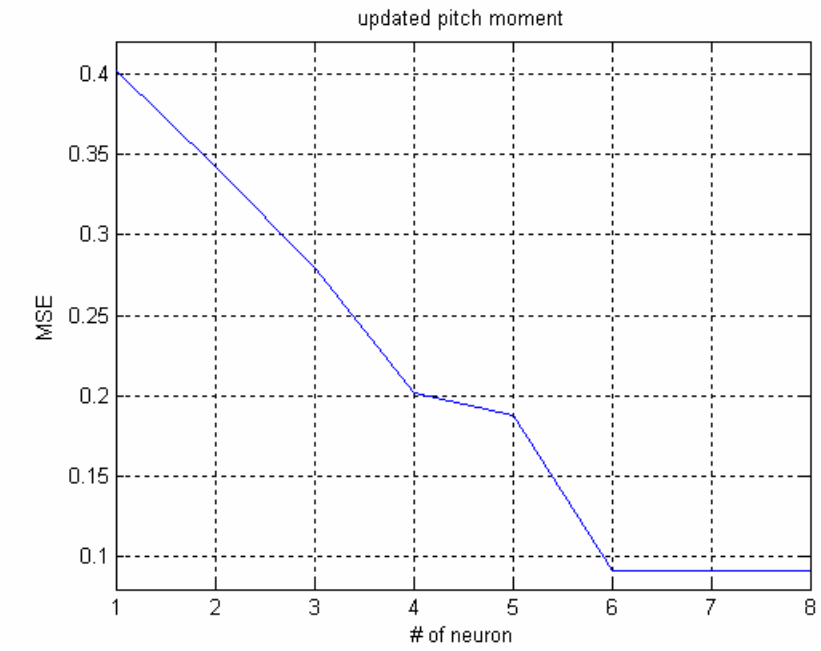

Figure 5.25 Error convergence of updated pitch moment in FS-ROLS

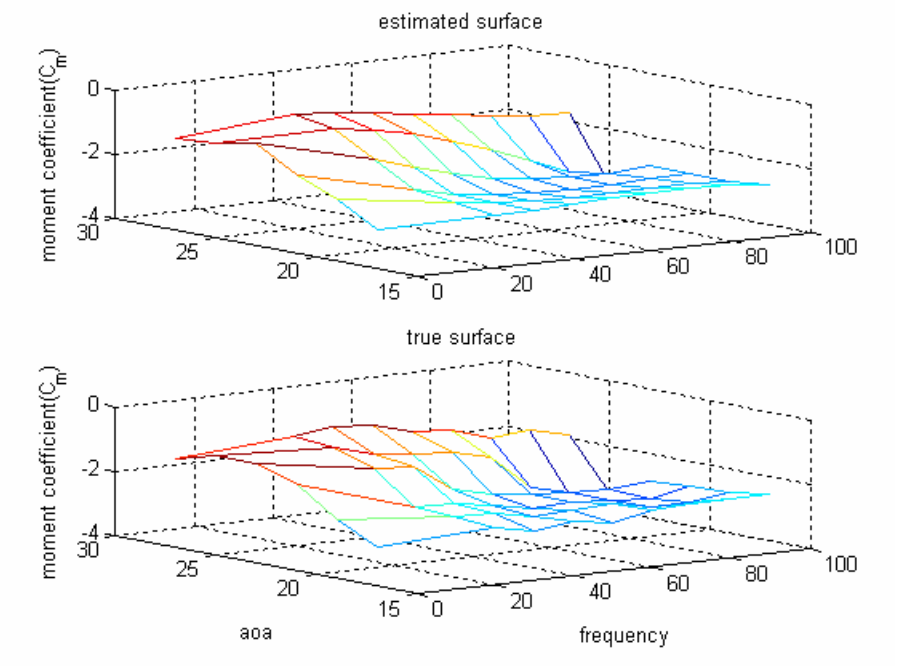

Figure 5.26 Estimated and true surface of updated pitch moment

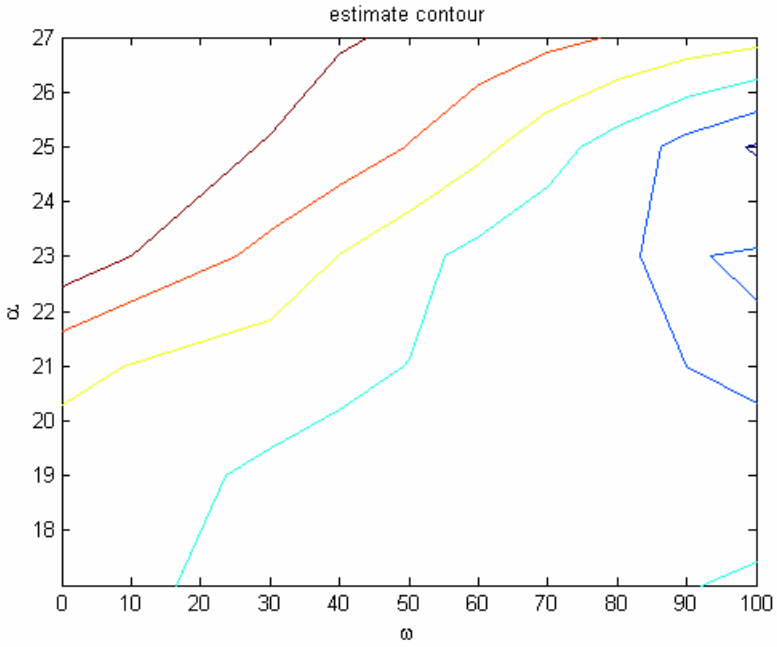

Figure 5.27 Estimated contour of updated pitch moment

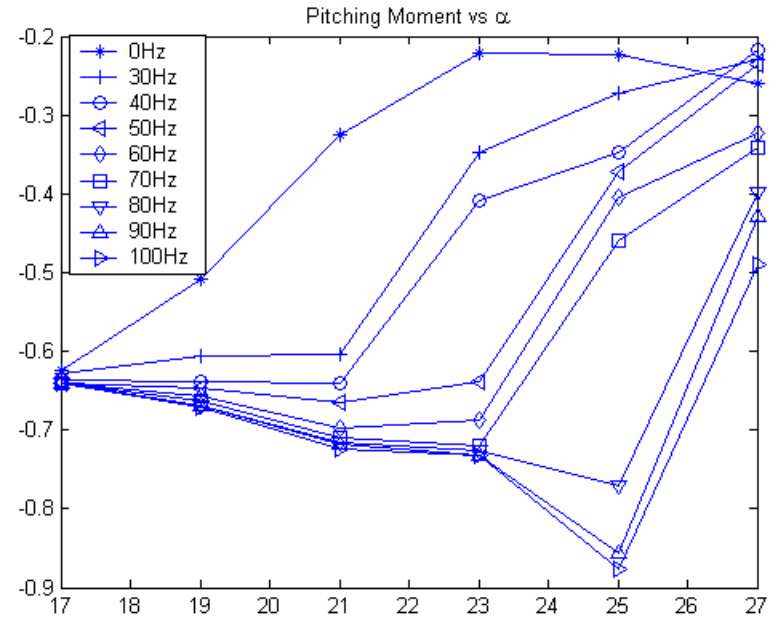

Figure 5.28 Updated pitch moment versus α

Resource Allocating Neural Network with Extended Kalman Filters (RAN-EKF)

The sequential function estimation problem is how we combine given prior estimate $f^{(n-1)}$ and new observation $I^{(n)}$ in obtaining the posterior estimate $f^{(n)}$. As one of the sequential learning method, RAN was developed which is by allocating new resources learning could be achieved in polynomial time.

The RAN is a single hidden layer network whose output response to an input pattern is a linear combination of the hidden unit responses.

$$f(\mathbf{x_n}) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(\mathbf{x_n}) \tag{41}$$

where $\phi_k(\mathbf{x_n})$ are the responses of the hidden units to an input **x,** K is number of hidden neurons and $\alpha_0 = y_0$ is an initial condition. The RAN hidden unit responses are given by

$$\phi_k(\mathbf{x_n}) = \exp(-\frac{1}{\sigma_k^2} \| \mathbf{x_n} - \mu_\mathbf{k} \|^2). \tag{42}$$

For each input $\mathbf{x_n}$, we compute $\phi_k(\mathbf{x_n})$ from the eqn(42), then

$$\varepsilon_n = \max\{\varepsilon_{\max}\gamma^n, \varepsilon_{\min}\}, \quad (0 < \gamma < 1).$$
$$e_n = y_n - f(\mathbf{x_n}), \tag{43}$$
$$e_{rmsn} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n} |y_i - f(\mathbf{x_i})|^2}{M}}$$

If $e_n > e_{\min}$, *and* $\|x_n - \mu_{nr}\| > \varepsilon_n$, *and* $e_{rmsn} > e'_{\min}$ then, allocate a new hidden unit with

$$\alpha_{k+1} = e_n, \mu_{\mathbf{k+1}} = \mathbf{x_n}, \text{and } \sigma_{k+1} = \kappa\|\mathbf{x_n} - \mu_{\mathbf{nr}}\| \tag{44}$$

When the observation $(\mathbf{x}_n, y_n)$ does not satisfy the criteria, the LMS algorithm is used to adapt the network parameters. However here we will use the EKF instead of LMS for updating the parameters. Given parameter vector **w**, the EKF algorithm provides the posterior estimate $\mathbf{w}^{(n)}$ from its prior error covariance estimate $\mathbf{w}^{(n\text{-}1)}$ and its prior error covariance estimate $\mathbf{P}_{n\text{-}1}$.

$$\begin{aligned} \mathbf{w_n} &= \mathbf{w_{(n\text{-}1)}} + \mathbf{k_n} e_n, \\ \mathbf{k_n} &= [R_n + \mathbf{a_n^T} P_{n-1} \mathbf{a_n}]^{-1} P_{n-1} \mathbf{a_n}, \\ P_n &= [I - \mathbf{k_n} \mathbf{a_n^T}] P_{n-1} + QI. \end{aligned} \tag{45}$$

where $\mathbf{k_n}$ is the Kalman gain matrix, $\mathbf{a_n}$ is the gradient vector and R_n is the variance of the measurement noise.

We compute the outputs of all hidden units $\sigma_k^n, (k = 1, ..., K)$ and find the largest absolute hidden unit output value $\left\| \sigma_{\max}^n \right\|$. Next, calculate the normalized value for each hidden unit $r_k^n = \left\| \dfrac{\sigma_k^n}{\sigma_{\max}^n} \right\|$, $(k = 1, ...K)$. If $r_k^n < \delta$ for M consecutive observations, then prune the k^{th} hidden neurons reduce the dimensionality of P_n to fit for the requirement of EKF.

$$\begin{aligned} \mathbf{a_n} &= \nabla_{\mathbf{w}} f(\mathbf{x_n}) \\ &= [1, \phi_1(\mathbf{x_n}), \phi_1(\mathbf{x_n}) \frac{2\alpha_1}{\sigma_1^2} (\mathbf{x_n} - \mu_1)^T, \phi_1(\mathbf{x_n}) \frac{2\alpha_1}{\sigma_1^3} \left\| \mathbf{x_n} - \mu_1 \right\|^2, \cdots, \\ &\quad \phi_{\mathbf{K}}(\mathbf{x_n}), \phi_{\mathbf{K}}(\mathbf{x_n}) \frac{2\alpha_{\mathbf{K}}}{\sigma_{\mathbf{K}}^2} (\mathbf{x_n} - \mu_{\mathbf{K}})^T, \phi_K(\mathbf{x_n}) \frac{2\alpha_{\mathbf{K}}}{\sigma_{\mathbf{K}}^3} \left\| \mathbf{x_n} - \mu_{\mathbf{K}} \right\|^2]^T \end{aligned} \tag{46}$$

The output of hidden units is

$$\sigma_k = \alpha_k \exp(-\frac{\left\|\mathbf{x_n} - \mu_\mathbf{i}\right\|}{\sigma_i})^2 . \tag{47}$$

The network parameters to be adjusted are included in $\mathbf{w} = [\alpha_0, \alpha_1, \mu_1^T, \sigma_1, \cdots, \alpha_k, \mu_k^T, \sigma_k]^T$.

$\mathbf{P}_n$ is updated by eqn(45). When a new hidden unit is allocated, the dimensionality of $\mathbf{P}_n$ increased by

$$\mathbf{P}_n = \begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{O} \\ \mathbf{O} & p_0\mathbf{I} \end{bmatrix} \tag{48}$$

and p_0 is an estimate of the uncertainty in the initial values as signed to the parameters and initializes the new rows and columns. The dimension of the matrix **I** is equal to the number of new parameters.

This RAN-EKF learning algorithm is implemented and simulated for the SJA wing data. From the results, we can observe the decrease of errors and faster convergence rate than backpropagation (MNN) but, it is obviously slower than FS-ROLS algorithm. In figure 5.29-37, the lift, drag and moment coefficient are displayed with RAN algorithm. 167 training set with random noise and test pattern used for learning. The parameter values selected in this experiment is shown in table 5.4. Slight changes of these parameters affect much in the each convergence rate. Therefore we have to choose these parameters carefully. As we observe from the table 5.5, the number of hidden units in RAN-EKF is over 100 in for all force coefficients which is much more than FS-ROLS. In figure 5.38-40, updated pitch moment data with RAN algorithm is performed in a range of angle of attack from 17 to 27 degree. Same as FS-ROLS algorithm simulation in section 5.3 we also get a better approximation result than

previous pitch moment data.

Table 5.4 Tuning Parameters of RAN-EKF

	$\varepsilon_{\max}$	$\varepsilon_{\min}$	$e_{\min}$	e_{rmse}	κ	δ
C_L	1	0.002	0.02	0.15	0.5	0.05
C_D	1	0.02	0.02	0.01	0.3	0.05
C_M	0.5	0.002	0.01	0.015	0.03	0.005

Table 5.5 Errors and number of hidden units in RAN-EKF algorithm

Force	C_L	C_D	C_M
Force Sum	139.09	42.8070	16.1550
Errors Sum	0.057(0.041%)	0.0133(0.031%)	0.0646(0.39%)
Mean error	0.0835	0.0064	-6.15×10^{-4}
# of hidden Unit	111	120	137

Table 5.6 Comparison between FS-ROLS and RAN-EKF algorithm in the updated pitch moment simulation

C_M	FS-ROLS	RAN-EKF
Errors Sum	0.0012%	0.036%
Mean error	0.0019	0.0058
# of hidden Unit	6	50

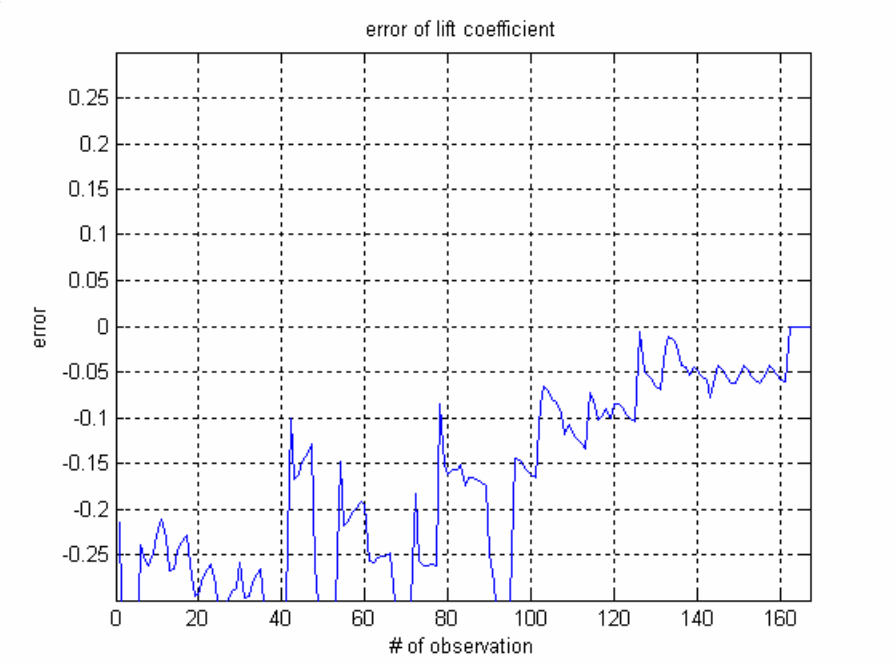

Figure 5.29 Error convergence of lift coefficient in RAN-EKF

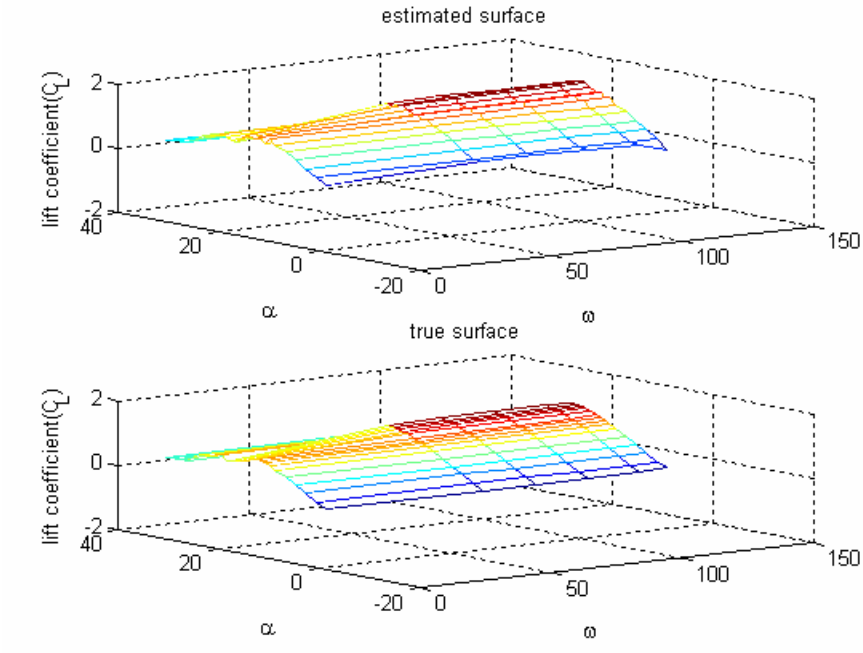

Figure 5.30 Estimated and true surface of lift coefficient

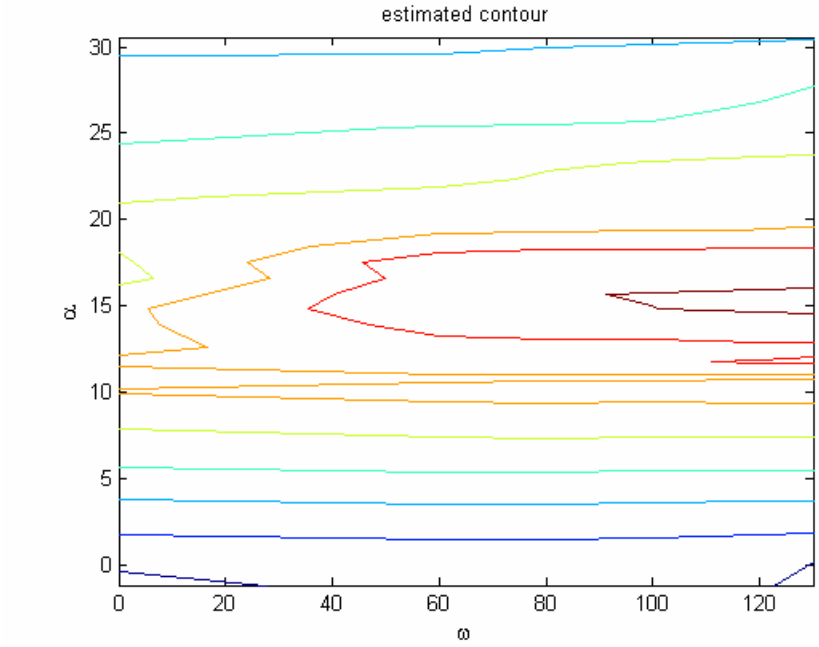

Figure 5.31 Estimated contour of lift coefficient

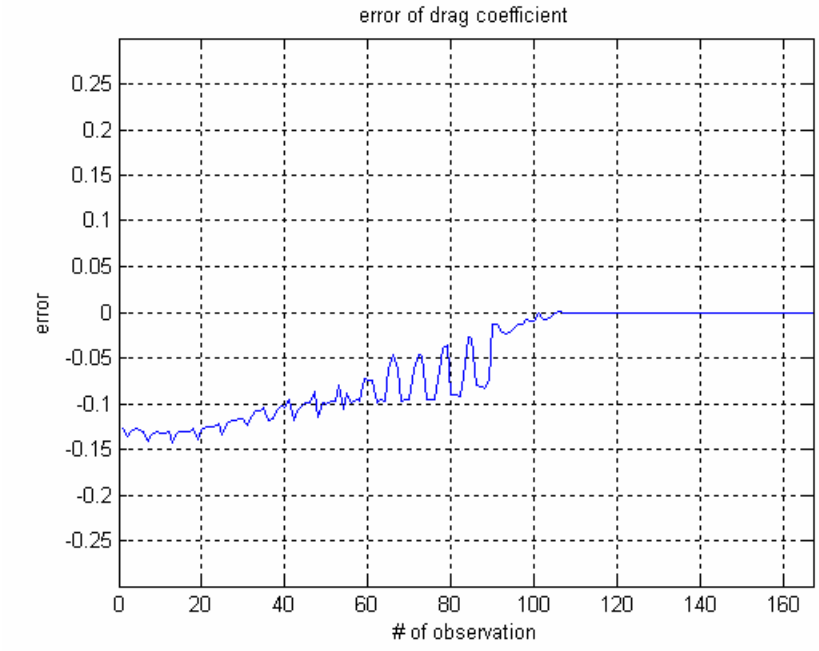

Figure 5.32 Error convergence of drag coefficient in RAN-EKF

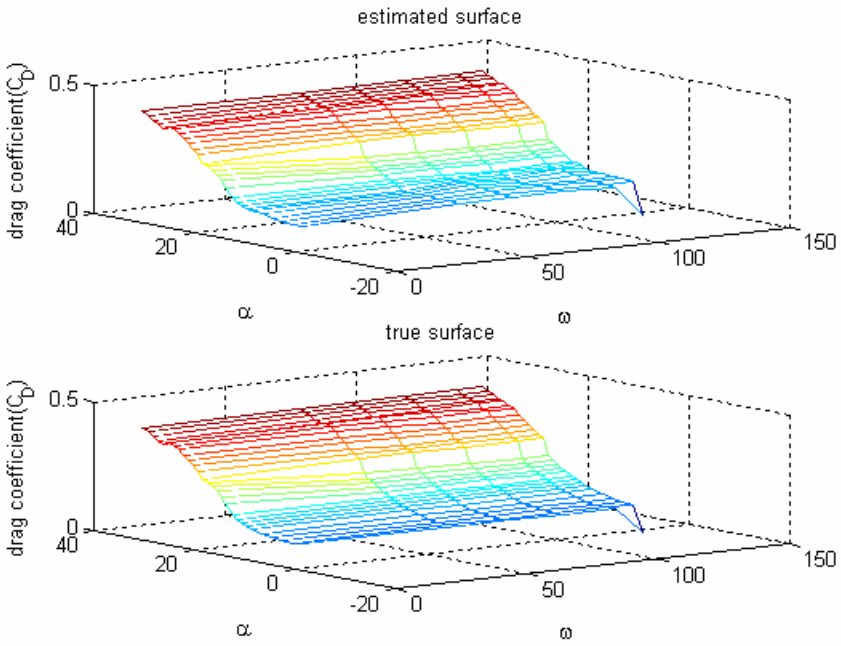

Figure 5.33 Estimated and true surface of drag coefficient

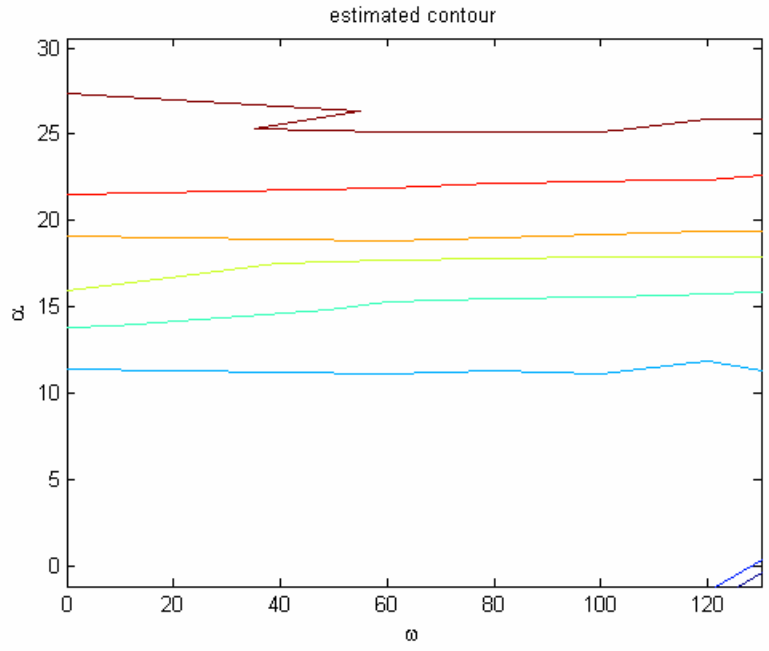

Figure 5.34 Estimated contour of drag coefficient

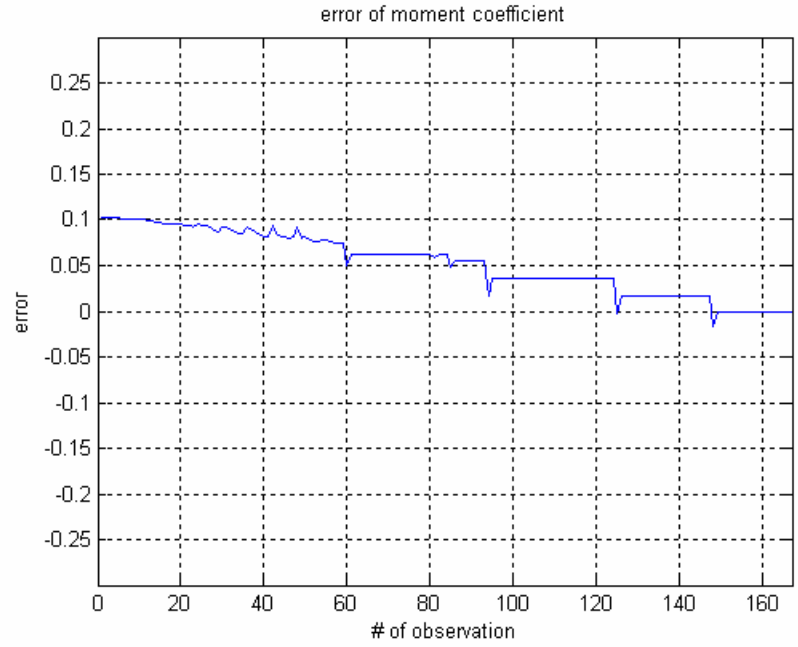

Figure 5.35 Error convergence of moment coefficient in RAN-EKF

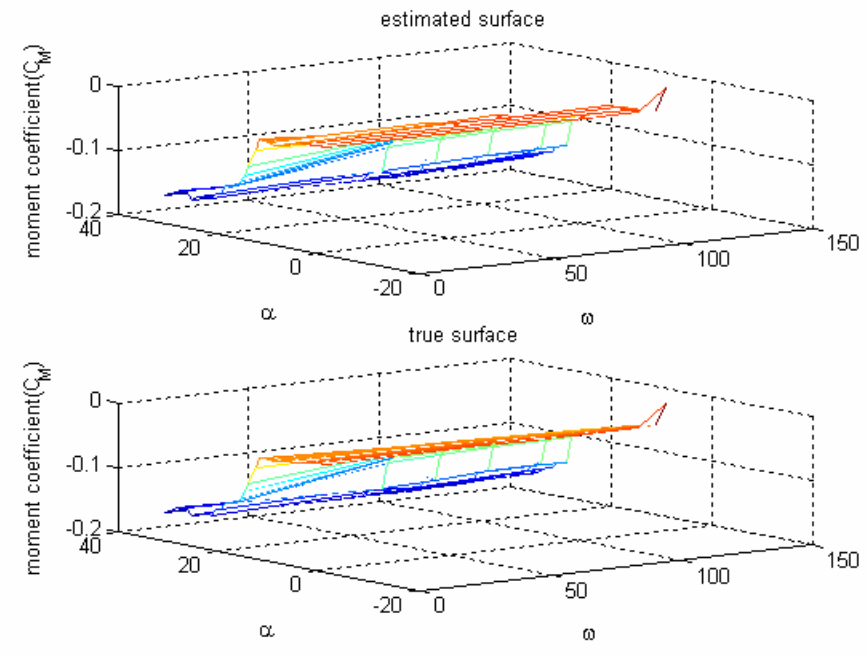

Figure 5.36 Estimated and true surface of moment coefficient

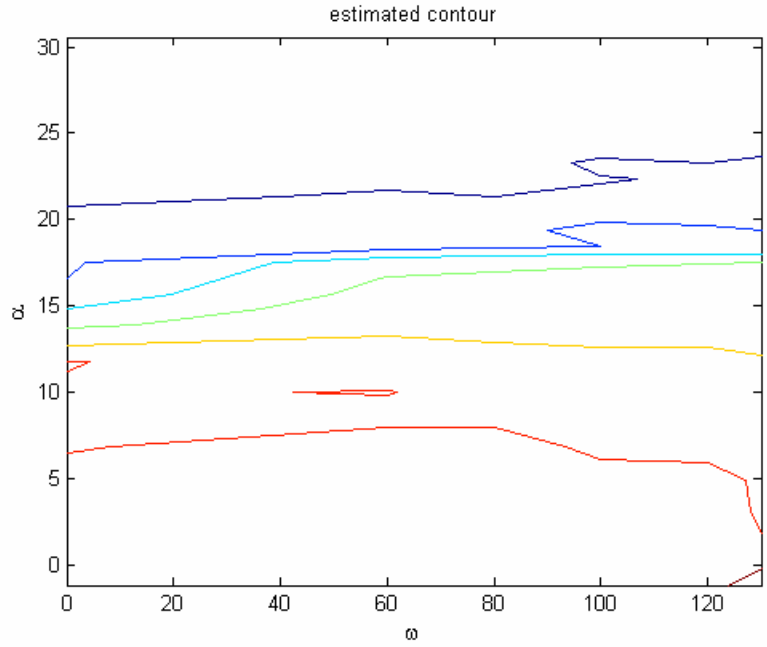

Figure 5.37 Estimated contour of moment coefficient

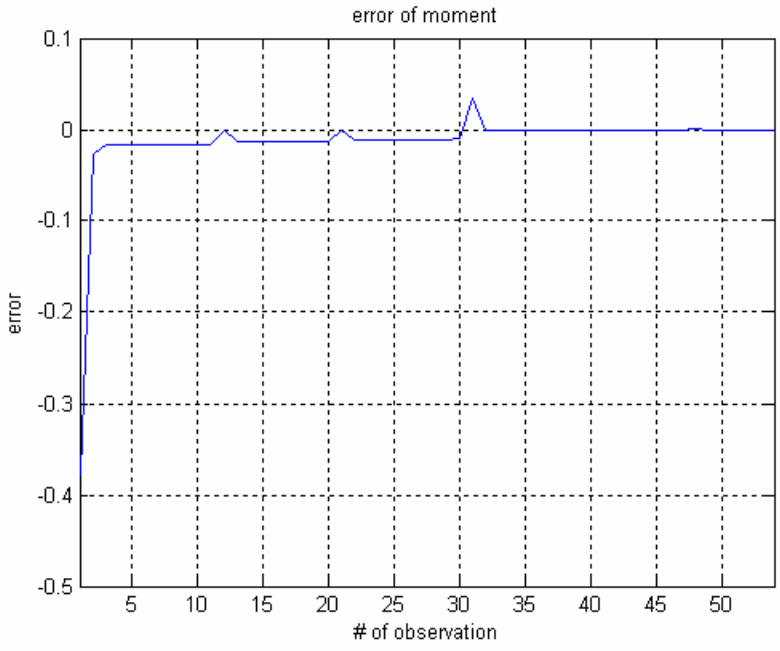

Figure 5.38 Error convergence of updated pitch moment in RAN-EKF

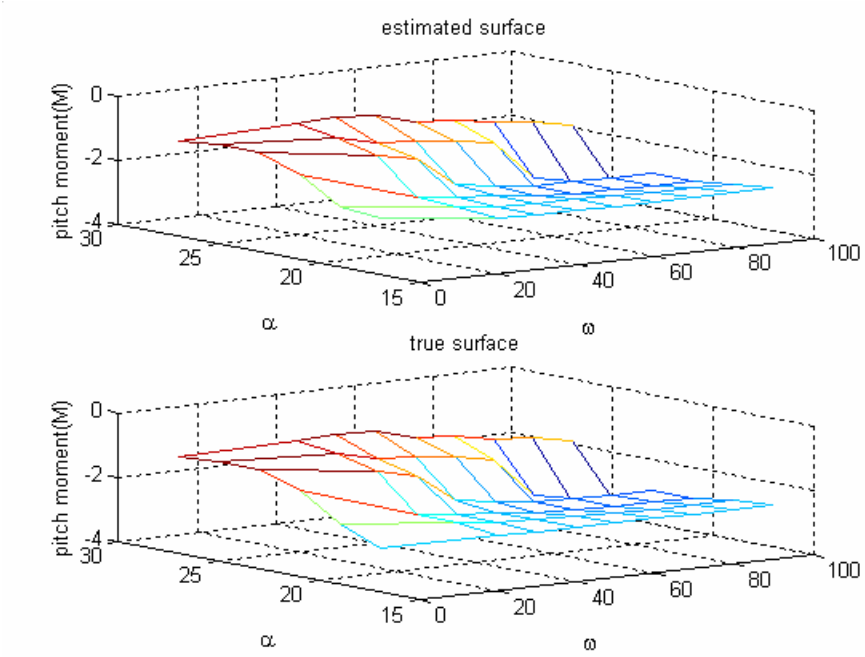

Figure 5.39 Estimated and true surface of updated pitch moment

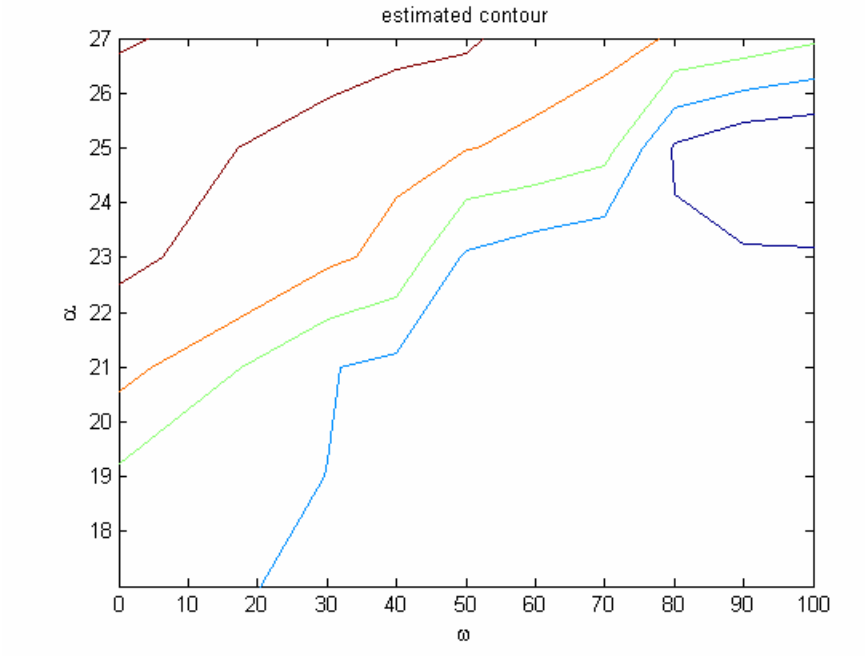

Figure 5.40 Estimated contour of updated pitch moment

From the above results we can easily recognize that all the errors of Resource Allocating Neural Network are finally converged after some observations and convergence rate is good but not as good as that of the FS-ROLS.

Let us compare two of RBFN Approximation Algorithms of FS-ROLS and RAN. The main similarities and differences between RAN and FS-ROLS are as follows. FS-ROLS and RAN both use the output values as well as the input values of the training set to determine the center placement. While RAN involve adaptive center and consequently some kind of learning procedure and multiple passes through the data, FS-ROLS has the process of heuristic center selection to determine which ones are included in the network. RAN has several preset parameters and thresholds which must be tuned to each new problem. Last, RAN searches a continuous space which grows in dimension as centers are added which is much better for sequential process while FS-ROLS heuristically searches a discrete space of different combinations of fixed centers.

The FS-ROLS fitting error is consistently smaller than the result from the RAN-EKF algorithm on the SJA wing test. Besides the number of hidden units in FS-ROLS algorithm is much smaller than for the RAN-EKF algorithm. We can also recognize FS-ROLS leads to much less error in table 5.6 which compares these two algorithms in the updated moment data. These results tell us that FS-ROLS algorithm performs better than RAN-EKF in the applications to the model of the SJA wing. However more elaborate applications the network grows complex and dimensionality may decrease the accuracy with which estimation algorithm converge so it is difficult to generalize from these results.

CHAPTER VI

THE NEURAL NETWORK ADAPTIVE CONTROL APPROACH

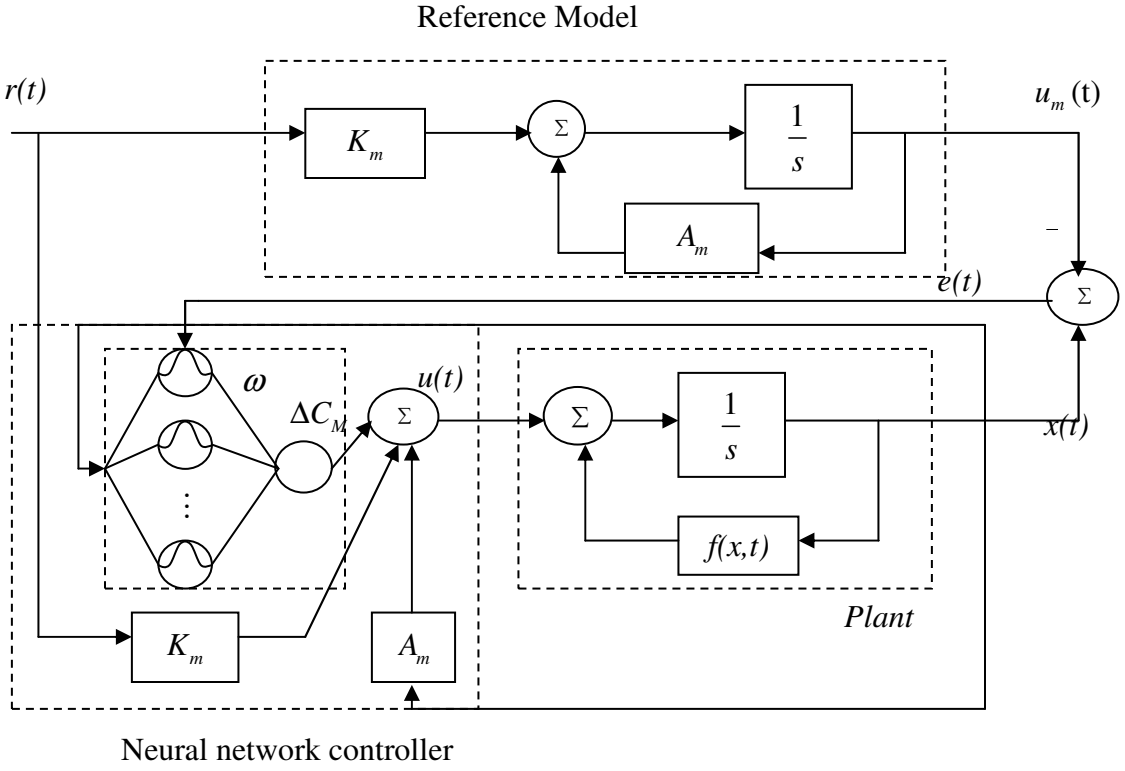

Figure 6.1 Model reference adaptive control system structure with neural network

Overview of Neural Network Adaptive control

Neural Network Adaptive control is powerful especially for controlling highly uncertain nonlinear and complex systems. In the model reference adaptive inverse control, the adaptive algorithm receives the tracking error between the plant output and the reference model output and the controller parameters updated to minimize the tracking error (Hagan, 1999) and this approach may be affected by sensor noise and

plant disturbance.[31-34] Using a neural network, an on-line model will be trained to receive the same inputs as the plant and to produce the same output. This controller scheme uses the two neural networks, one is a controller network and the other is model network which is trained off-line using plant measurements.[26,27,35] The controller is trained adaptively so that the plant output can track a reference model output. One of the problems implementing the neural network to control scheme is that computational work of performing real-time control on any system with more than a few degrees of freedom becomes excessively high and may exceed available computational resources (Jacob 1997). Figure 6.1 represent the block diagram of Model reference adaptive control system structure with neural network.

There are two kinds of adaptive neural control designs used in recent literature. One is direct adaptive control approach and the other is the indirect adaptive control approach. In the direct adaptive control approach, Lyapunov stability theory is used for designing the network tuning rule. On the contrary, in the indirect adaptive control, two neural networks are used; one is for identifying the forward/inverse dynamics of the system and the other is connected in cascade with the system to be controlled and its parameters are updated on-line to implement a suitable control law (Sundararajan, Saratchandran and Li, 2002).

The main advantage of direct adaptive control over the indirect adaptive control is that in the latter, there are no strict mathematical proof to guarantee the stability of the tracking error and the convergence of the network parameters. Also, for the direct approach, we need a smaller amount of information about the plant and a simpler design.

Specifically, bounds of the inputs are only assumed to exist, but neither to be known or to be estimated (Raul, 2001). This architecture employees a Gaussian RBF network to adaptively compensate for the system nonlinearities. To implement the RBFN, a stable weight adjustment mechanism is derived using Lyapunov theory. With this tuning rule the weights of the RBFN converge to the optimal weights gradually. Using Lyapunov stability theory, the derived tuning rule can guarantee the convergence of the tracking error and the stability of the overall system. In the feedback-error-learning strategy, the total control effort, **u**, is composed of the output of the neural controller and the output of the conventional feedback controller. The output of the conventional controller is utilized as the feedback error signal to tune the parameters of the RBFN, so it is expected that the output of the conventional feedback controller will tend to zero as the neural controller learns the appropriate control law.[36] The main advantage of direct adaptive control scheme is that the stability of the overall system can be guaranteed provided the adaptive tuning law for the nonlinear system is derived based on a Lyapunov synthesis approach. Some useful dynamic theory to formulate adaptive control law introduced in the Appendix B.

Nonlinear System Identification Using Lyapunov-based Fully Tuned RBFN

Identification Strategy and System Error Dynamics

Nonlinear dynamic system is given as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \quad \mathbf{x}(0) = 0 \tag{49}$$

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \tag{50}$$

$$\mathbf{g}(\mathbf{x}(t),\mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) - \mathbf{A}\mathbf{x}(t) \tag{51}$$

Setting the RBF network's inputs $\xi = [\mathbf{x}(t)^{\mathbf{T}}, \mathbf{u}(t)^{\mathbf{T}}]^{\mathbf{T}}$, the problem of system identification can be converted into a nonlinear function approximation problem. Therefore Growing RBF network based system model can be written as

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \sum_{k=1}^{h} \mathbf{w}_k^{*T} \exp(-\frac{1}{\sigma_k^{*2}} \| \boldsymbol{\xi} - \boldsymbol{\mu}_{\mathbf{k}}^{*} \|^2) + \boldsymbol{\varepsilon}_h$$

$$= \mathbf{A}\mathbf{x}(t) + \mathbf{W}^{*\mathbf{T}}\boldsymbol{\varphi}(\mu^*, \sigma^*, \xi) + \boldsymbol{\varepsilon}_{\mathbf{h}} \tag{52}$$

$\mathbf{W}$ is $h \times n$ optimal weight matrix, $\boldsymbol{\varphi}$ is $h \times 1$ Gaussian function vector, μ^* is optimal center and σ^* is optimal width, hereafter the approximation error $\boldsymbol{\varepsilon}_{\mathbf{h}}$ is defined as

$$\boldsymbol{\varepsilon}_{\mathbf{h}} = \mathbf{g}(\mathbf{x},\mathbf{u}) - \mathbf{W}^{*\mathbf{T}}\boldsymbol{\varphi}^* \tag{53}$$

$$\boldsymbol{\varepsilon}_{\mathbf{H}} = \sup_{x \in \bar{X}, u \in \bar{U}} \| \varepsilon_h(x,u) \| \tag{54}$$

And **g(.)** can be approximated by growing RBF network as

$$\hat{\mathbf{g}}(\mathbf{x},\mathbf{u}) = \sum_{k=1}^{h} \hat{\mathbf{w}}_k^T \exp(-\frac{1}{\hat{\sigma}_k^2} \| \boldsymbol{\xi} - \hat{\boldsymbol{\mu}}_{\mathbf{k}} \|^2) = \hat{\mathbf{W}}^{\mathbf{T}}\boldsymbol{\varphi}(\hat{\mu}, \hat{\sigma}, \xi) \tag{55}$$

where $\hat{\mathbf{W}}$ is estimated weight matrix and $\hat{\mu}$, $\hat{\sigma}$ are estimated center and width. After substituting eqn(55), we can get the identification model of

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \hat{\mathbf{W}}^{\mathbf{T}}\hat{\boldsymbol{\varphi}}. \tag{56}$$

Defining the approximation error as $\mathbf{e}=\mathbf{x}-\hat{\mathbf{x}}$

$$\begin{aligned} \dot{\mathbf{e}} &= \mathbf{A}\mathbf{e} + \mathbf{W}^{*\mathbf{T}}\boldsymbol{\varphi}^* - \hat{\mathbf{W}}^{\mathbf{T}}\hat{\boldsymbol{\varphi}} + \boldsymbol{\varepsilon}_{\mathbf{h}} \\ &= \mathbf{A}\mathbf{e} + \hat{\mathbf{W}}^{\mathbf{T}}\tilde{\boldsymbol{\varphi}} - \tilde{\mathbf{W}}^{\mathbf{T}}\hat{\boldsymbol{\varphi}} + \boldsymbol{\varepsilon}_{\mathbf{h}} \end{aligned} \tag{57}$$

where $\tilde{\mathbf{W}} = \mathbf{W}^* - \hat{\mathbf{W}}$, $\tilde{\boldsymbol{\varphi}} = \boldsymbol{\varphi}^* - \hat{\boldsymbol{\varphi}}$.

Stable Parameter Tuning Rules

Choose the following Lyapunov candidate function

$$\mathbf{V} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{P}\mathbf{e} + \frac{\mathbf{1}}{\mathbf{2}}tr(\tilde{\mathbf{W}}^{\mathbf{T}}\tilde{\mathbf{W}}) + \frac{\mathbf{1}}{\mathbf{2}}\tilde{\phi}^{T}\tilde{\phi}\,. \tag{58}$$

The derivative of the Lyapunov function is given by

$$\dot{\mathbf{V}} = \frac{\mathbf{1}}{\mathbf{2}}(\dot{\mathbf{e}}^{\mathbf{T}}\mathbf{P}\mathbf{e} + \mathbf{e}^{\mathbf{T}}\mathbf{P}\dot{\mathbf{e}}) + tr(\tilde{\mathbf{W}}^{\mathbf{T}}\dot{\tilde{\mathbf{W}}}) + \tilde{\phi}^{T}\dot{\tilde{\phi}} \tag{59}$$

$$= -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{Q}\mathbf{e} + \varepsilon_h^{T}\mathbf{P}\mathbf{e} + \tilde{\phi}^{\mathbf{T}}\hat{\mathbf{W}}\mathbf{P}\mathbf{e} + \hat{\phi}^{\mathbf{T}}\tilde{\mathbf{W}}\mathbf{P}\mathbf{e} + tr(\tilde{\mathbf{W}}^{\mathbf{T}}\dot{\tilde{\mathbf{W}}}) + \tilde{\phi}^{T}\dot{\tilde{\phi}}$$

$$= -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{Q}\mathbf{e} + \varepsilon_h^{T}\mathbf{P}\mathbf{e} + \tilde{\phi}^{\mathbf{T}}(\hat{\mathbf{W}}\mathbf{P}\mathbf{e} + \dot{\tilde{\phi}}) + \sum_{i=1}^{n}(\tilde{w}_i\hat{\phi}(\mathbf{P}\mathbf{e})_i + \tilde{w}_i^{T}\dot{\tilde{w}}_i)$$

where $\mathbf{Q = -(P^{T}A + A^{T}P)}$ and $\mathbf{A}$ is Hurwitz. Therefore, letting $\dot{\tilde{w}}_i$, $\dot{\tilde{\phi}}$ be expressed as

$$\dot{\tilde{w}}_i = -\hat{\phi}(\mathbf{P}\mathbf{e})_i \quad i = 1,...,n \tag{60}$$

$$\dot{\tilde{\phi}} = -\hat{\mathbf{W}}\mathbf{P}\mathbf{e} \tag{61}$$

The derivative of $\dot{\mathbf{V}}$ is

$$\dot{\mathbf{V}} = -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{Q}\mathbf{e} + \varepsilon_h^{T}\mathbf{P}\mathbf{e}\,. \tag{62}$$

$$\dot{\mathbf{V}} \le -\frac{\mathbf{1}}{\mathbf{2}} \| \mathbf{e} \| \lambda_{\min}(\mathbf{Q}) \| \mathbf{e} \| + \varepsilon_H^{T} \| \mathbf{P} \| \| \mathbf{e} \|\,. \tag{63}$$

$\dot{\mathbf{V}} \le 0$ if $\| \mathbf{e} \| > \dfrac{2\varepsilon_H \| \mathbf{P} \|}{| \lambda_{\min}(\mathbf{Q}) |} = E_a$. Since $\dot{\tilde{w}}_i = \dot{w}_i^{*} - \dot{\hat{w}}_i$, $\dot{\tilde{\phi}} = \dot{\phi}^{*} - \dot{\hat{\phi}}$ and $\dot{w}_i^{*} = 0$, $\dot{\phi}^{*} = 0$

$$\dot{\hat{w}}_i = \hat{\phi}(\mathbf{P}\mathbf{e})_i \quad i = 1,...,n \tag{64}$$

$$\dot{\hat{\phi}} = \hat{\mathbf{W}}\mathbf{P}\mathbf{e} \tag{65}$$

Direct Adaptive Control Strategy and Application Using RBFN

At this point, the adaptive tuning rules are derived using Lyapunov synthesis approach, which guarantee closed-loop stability. RBFN with all the parameters being updated can capture the system dynamics more quickly and accurately and hence more suitable for aircraft flight control

Problem Formulation

The system dynamics is presented by the form of equation (66)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \quad x(0) = 0 \tag{66}$$

Partitioning **x**, the dynamics can be written

$$\begin{pmatrix} \dot{\mathbf{x}}_t \\ \dot{\mathbf{x}}_r \end{pmatrix} = \begin{pmatrix} \mathbf{f}_t(\mathbf{x},\mathbf{u}) \\ \mathbf{f}_r(\mathbf{x},\mathbf{u}) \end{pmatrix} \tag{67}$$

The objective is to set up the neurocontroller which the plant state $\mathbf{x_t}$ can track the desired state $\mathbf{x_{dt}}$ and desired control input can be expressed as

$$\mathbf{u_d(t)} = \bar{\mathbf{f}}_\mathbf{t}(\mathbf{x_d}, \dot{\mathbf{x}}_\mathbf{dt}) . \tag{68}$$

where $\bar{\mathbf{f}}_\mathbf{t}$,($p \times 1$) smooth function, is the inverse function of $\mathbf{f_t}$, and $\mathbf{x_d} = \left[\mathbf{x}_\mathbf{dt}^\mathbf{T}, \mathbf{x}^\mathbf{T}_\mathbf{dr} \right]$

Stable Tuning Rule Using RBFN (YanLi, Narasimahan 2001)

The Control Strategy updating rule for a fully tuned RBFN controller can be derived based on a feedback-error-learning scheme.

The error dynamics are defined as

$$\dot{\mathbf{e}} = \dot{\mathbf{x}} - \dot{\mathbf{x}}_\mathbf{d} = \mathbf{f}(\mathbf{x},\mathbf{u}) - \mathbf{f}(\mathbf{x_d}, \mathbf{u_d}) \tag{69}$$

$$\dot{\mathbf{e}} = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}^\mathbf{T}} \Big|_{\mathbf{x_d},\mathbf{u_d}} (\mathbf{x} - \mathbf{x_d}) + \mathbf{O}(\mathbf{x} - \mathbf{x_d}) + \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}^\mathbf{T}} \Big|_{\mathbf{x_d},\mathbf{u_d}} (\mathbf{u} - \mathbf{u_d}) + \mathbf{O}(\mathbf{u} - \mathbf{u_d}) \tag{70}$$

O represents the higher order term and neglect the higher order term then,

$$\dot{\mathbf{e}} = \mathbf{A}(t)\mathbf{e} + \mathbf{B}(t)(\mathbf{u} - \mathbf{u_d}) . \tag{71}$$

This approximation is crucial to make the model affine in the control.

With only the linear proportional controller $\mathbf{u}= \mathbf{K_P}(t)\,\mathbf{e}$, the error dynamics is

$$\dot{\mathbf{e}} = (\mathbf{A}(t)\mathbf{e} + \mathbf{B}(t)\mathbf{K_P}(t))\mathbf{e} - \mathbf{B}(t)\mathbf{u_d} \tag{72}$$

Put the RBFN controller signal together with proportional controller.

$$\mathbf{u} = \mathbf{K_P}(t)\mathbf{e} + \mathbf{u_{nn}} . \tag{73}$$

Let us look at the RBFN Approximation and Error Dynamics. Setting the RBFN's inputs $\xi = [\mathbf{x_d}^{\mathbf{T}} \dot{\mathbf{x}}_{\mathbf{dt}}^{\mathbf{T}}]$, $\mathbf{u_d}$ can be approximated by an RBFN controller through on-line learning

$$\begin{aligned} \mathbf{u_d} &= \mathbf{u}^*_{\mathbf{nn}} + \boldsymbol{\varepsilon}_{\mathbf{h}} \\ &= \sum_{k=1}^{h} \mathbf{w}_k^{*T} \exp(-\frac{1}{\sigma_k^{*2}} \| \boldsymbol{\xi} - \boldsymbol{\mu}_{\mathbf{k}}^* \|^2) + \boldsymbol{\varepsilon}_h \\ &= \mathbf{W}^{*\mathbf{T}} \boldsymbol{\varphi}(\mu^*, \sigma^*, \xi) + \boldsymbol{\varepsilon}_{\mathbf{h}} \end{aligned} \tag{74}$$

$\boldsymbol{\varepsilon}_{\mathbf{h}}$ is bounded by a constant $\boldsymbol{\varepsilon}_H$ and

$$\boldsymbol{\varepsilon}_{\mathbf{H}} = \sup_{x \in \bar{X}, u \in \bar{U}} \| \varepsilon_h(x,u) \| \tag{75}$$

With RBFN controller control input vector **u** is

$$\begin{aligned} \mathbf{u} &= \sum_{k=1}^{h} \hat{\mathbf{w}}_k^T \exp(-\frac{1}{\hat{\sigma}_k^2} \| \boldsymbol{\xi} - \hat{\boldsymbol{\mu}}_{\mathbf{k}} \|^2) + \mathbf{K_P}(t)\mathbf{e} \\ &= \hat{\mathbf{W}}^T \hat{\boldsymbol{\varphi}} + \mathbf{K_P}(t)\mathbf{e} \end{aligned} \tag{76}$$

By substituting **u** into eqn(71), the error dynamics becomes

$$\dot{\mathbf{e}} = (\mathbf{A}(t)\mathbf{e} + \mathbf{B}(t)\mathbf{K_P}(t))\mathbf{e} + \mathbf{B}(t)(\hat{\mathbf{W}}^{\mathbf{T}}\hat{\boldsymbol{\varphi}} - \mathbf{W}^{*\mathbf{T}}\boldsymbol{\varphi}^* - \boldsymbol{\varepsilon}_{\mathbf{h}}) . \tag{77}$$

By defining $\mathbf{J}(t) = \mathbf{A}(t)\mathbf{e} + \mathbf{B}(t)\mathbf{K_P}(t)$, $\tilde{\mathbf{W}} = \mathbf{W}^* - \hat{\mathbf{W}}$, $\tilde{\varphi} = \varphi^* - \hat{\varphi}$ the error dynamics may be written as

$$\begin{aligned}\dot{\mathbf{e}} &= \mathbf{J}(t)\mathbf{e} - \mathbf{B}(t)(\hat{\mathbf{W}}^{\mathbf{T}}\tilde{\varphi} + \tilde{\mathbf{W}}^{\mathbf{T}}\hat{\varphi} + \tilde{\mathbf{W}}^{\mathbf{T}}\tilde{\varphi}) - \mathbf{B}(t)\varepsilon_{\mathbf{h}} \\ &\approx \mathbf{J}(t)\mathbf{e} - \mathbf{B}(t)(\hat{\mathbf{W}}^{\mathbf{T}}\tilde{\varphi} + \tilde{\mathbf{W}}^{\mathbf{T}}\hat{\varphi}) - \mathbf{B}(t)\varepsilon_{\mathbf{h}}\end{aligned} \tag{78}$$

where $\mathbf{B}(t)(\hat{\mathbf{W}}^{\mathbf{T}}\tilde{\varphi} + \tilde{\mathbf{W}}^{\mathbf{T}}\hat{\varphi})$ represent the learning error $\mathbf{E_t}$.

Stable Adaptive Tuning Rule for RBFN (Narasimahan, 2001)

Let us choose the following Lyapunov candidate function

$$\mathbf{V} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{P(t)e} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{tr}(\tilde{\mathbf{W}}^{\mathbf{T}}\Theta\tilde{\mathbf{W}}) + \frac{\mathbf{1}}{\mathbf{2}}\tilde{\varphi}^{\mathbf{T}}\Lambda\tilde{\varphi} \tag{79}$$

$\mathbf{P}$(t) is $n \times n$ time varying, symmetric, and positive definite matrix and Θ, Λ are $h \times h$ nonnegative definite matrices. The derivative of $\mathbf{V}$ is given by

$$\dot{\mathbf{V}} = \frac{\mathbf{1}}{\mathbf{2}}(\dot{\mathbf{e}}^{\mathbf{T}}\mathbf{P}(t)\mathbf{e} + \mathbf{e}^{\mathbf{T}}\mathbf{P}(t)\dot{\mathbf{e}} + \mathbf{e}^{\mathbf{T}}\dot{\mathbf{P}}(t)\mathbf{e}) + tr(\tilde{\mathbf{W}}^{\mathbf{T}}\Theta\dot{\tilde{\mathbf{W}}}) + \tilde{\varphi}^{\mathbf{T}}\Lambda\dot{\tilde{\varphi}} \tag{80}$$

$$\begin{aligned}\dot{\mathbf{V}} &= -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{Q}(t)\mathbf{e} - \varepsilon_h^T\mathbf{B}(t)\mathbf{P}(t)\mathbf{e} - \tilde{\varphi}^{\mathbf{T}}\hat{\mathbf{W}}\mathbf{B}(t)\mathbf{P}(t)\mathbf{e} \\ &\quad - \hat{\varphi}^{\mathbf{T}}\tilde{\mathbf{W}}\mathbf{B}(t)\mathbf{P}(t)\mathbf{e} + tr(\tilde{\mathbf{W}}^{\mathbf{T}}\Theta\dot{\tilde{\mathbf{W}}}) + \tilde{\varphi}^{\mathbf{T}}\Lambda\dot{\tilde{\varphi}} \\ &= -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{Q}(t)\mathbf{e} - \varepsilon_h^T\mathbf{B(t)}^T\mathbf{P}(t)\mathbf{e} + \tilde{\phi}^{\mathbf{T}}(-\hat{\mathbf{W}}\mathbf{B}(t)^T\mathbf{P}(t)\mathbf{e} + \Lambda\dot{\tilde{\varphi}}) \\ &\quad + \sum_{i=1}^{p}(-\tilde{\mathbf{w}}_i\hat{\varphi}\mathbf{B}_i\mathbf{(t)}^T\mathbf{P}_i(t)\mathbf{e} + \tilde{\mathbf{w}}_i^{\mathbf{T}}\Theta\dot{\tilde{\mathbf{w}}}_i)\end{aligned} \tag{81}$$

where $\mathbf{Q} = -\frac{1}{2}(\mathbf{J(t)}^{\mathbf{T}}\mathbf{P(t)} + \mathbf{P(t)J(t)} + \dot{\mathbf{P}}\mathbf{(t)})$.

If we select

$$\dot{\tilde{\mathbf{w}}}_i = \Theta^{-1}\hat{\phi}\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e} \quad i = 1,...,n \tag{82}$$

$$\dot{\tilde{\boldsymbol{\varphi}}} = \boldsymbol{\Lambda}^{-1}\hat{\mathbf{W}}\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e} \tag{83}$$

then derivative of $\mathbf{V}$ is

$$\dot{\mathbf{V}} = -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{Q}(t)\mathbf{e} - \varepsilon_h^T\mathbf{B}(t)\mathbf{P}(t)\mathbf{e} \tag{84}$$

and this can be negative if

$$\dot{\mathbf{V}} \le -\frac{\mathbf{1}}{\mathbf{2}} \| \mathbf{e} \| \lambda_{\min}(\mathbf{Q}) \| \mathbf{e} \| - \varepsilon_H^T\mathbf{P} \| \mathbf{B}(t) \| \varepsilon_H . \tag{85}$$

Since $\dot{\tilde{w}}_i = \dot{w}_i^* - \dot{\hat{w}}_i,\ \dot{\tilde{\phi}} = \dot{\phi}^* - \dot{\hat{\phi}},\ and\ \ \dot{w}_i^* = 0,\quad \dot{\phi}^* = 0$

$$\dot{\hat{\mathbf{w}}}_i = -\boldsymbol{\Theta}^{-1}\hat{\boldsymbol{\varphi}}\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e} \quad i = 1,..., p \tag{86}$$

$$\dot{\hat{\boldsymbol{\varphi}}} = -\boldsymbol{\Lambda}^{-1}\hat{\mathbf{W}}\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e} \tag{87}$$

Implementation of the Tuning Rule for Dynamic Adaptive Controller (Li and Sundararajan, 2001, see Appendix C)

The Gaussian function $\hat{\boldsymbol{\varphi}}$ is embedded with the centers' locations and widths. Combining all the adaptable parameters into a composite parameter vector, $\chi = [\hat{\mathbf{w}}_1, \hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\sigma}}_1, ..., \hat{\mathbf{w}}_h, \hat{\boldsymbol{\mu}}_h, \hat{\boldsymbol{\sigma}}_h]$ a simple updating law is derived. Firstly, the weight can be converted into

$$\begin{aligned} &\dot{\hat{\mathbf{w}}}_i = -\Theta^{-1}\hat{\phi}\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e} \quad i = 1,..., p \\ &\Rightarrow \dot{\hat{\mathbf{w}}}_k^T = -\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e}\hat{\phi}^T \\ &\Rightarrow \dot{\underline{\hat{\mathbf{w}}}}_k^T = -\mathbf{B}_i(t)^T\mathbf{P}_i(t)\mathbf{e}\hat{\phi}_k \quad k = 1,...,h \ \ \boldsymbol{\Theta} = I \end{aligned} \tag{88}$$

$\hat{\mathbf{g}} = \hat{\mathbf{W}}^{\mathbf{T}}\hat{\boldsymbol{\varphi}}$ is defined the output of the RBFN and $\hat{\boldsymbol{\varphi}}_k$ is the derivative of the $\hat{\mathbf{g}}$ to the weight $\hat{\mathbf{w}}_k^T$. Then weight updated form of discrete form is

$$\hat{\underline{\mathbf{w}}}_{\mathbf{k}}^{\mathbf{T}}(n+1) = \hat{\underline{\mathbf{w}}}_{\mathbf{k}}^{\mathbf{T}}(n) - \tau \frac{\partial \hat{\mathbf{g}}}{\partial \hat{\underline{\mathbf{w}}}_{\mathbf{k}}^{\mathbf{T}}} \mathbf{B}(n)^T \mathbf{P}(n)\mathbf{e}(n) \quad k = 1...h \tag{89}$$

similarly the updating law rule for center and width are

$$\hat{\boldsymbol{\mu}}_k(n+1) = \hat{\boldsymbol{\mu}}_k(n) - \tau\eta_1 \frac{\partial \hat{\mathbf{g}}}{\partial \hat{\boldsymbol{\mu}}_k^T} \mathbf{B}(n)^T \mathbf{P}(n)\mathbf{e}(n) \quad k = 1...h \tag{90}$$

$$\hat{\sigma}_k(n+1) = \hat{\sigma}_k(n) - \tau\eta_2 \frac{\partial \hat{\mathbf{g}}}{\partial \hat{\sigma}_k^T} \mathbf{B}(n)^T \mathbf{P}(n)\mathbf{e}(n) \quad k = 1...h \tag{91}$$

Integrating all the parameters we can write

$$\chi(n+1) = \chi(n) - \eta\tilde{\mathbf{\Pi}}(n)\mathbf{B}(n)^T \mathbf{P}(n)\mathbf{e}(n), \quad \tilde{\mathbf{\Pi}}(n) = \nabla_\chi \hat{g}(\xi_n) \tag{92}$$

where η is learning late and $\eta < \min(\tau, \eta_1, \eta_2)$.

Evaluation of Control Error in Terms of Neural Network Learning Error

We can rewrite the plant eqn(50) by

$$\dot{\mathbf{x}}(t) + f(\mathbf{x}, t) = \mathbf{u}(t), \quad t \geq 0 \tag{93}$$

where $\mathbf{x}(t)$ is output signal , $f(\mathbf{x},t)$ is unknown static nonlinear function and $\mathbf{u}(t)$ is input signal. Desired reference model is represented by following equation.

$$\dot{\mathbf{x}}_m(t) + \mathbf{A}_m \mathbf{x}_m(t) = \mathbf{K}_m \mathbf{r}(t) \tag{94}$$

where $\mathbf{x}_m$ is output reference signal $\mathbf{r}(t)$ is reference input and $\mathbf{A}_m, \mathbf{K}_m > 0$.

The objective of the control law is to obtain a controller which can follow the reference model within the limit of

$$\lim_{t\to\infty} \left| \mathbf{x}(t) - \mathbf{x}_m(t) \right| \leq \varepsilon . \tag{95}$$

where $\varepsilon > 0$. Then the control law can be proposed as

$$\mathbf{u}(t) \;=\; -\mathbf{A}_m\mathbf{x}_m(t) + \mathbf{K}_m\mathbf{r}(t) + N_f\;[\mathbf{x}(t),\;\mathbf{w}(t)] \;=\; \boldsymbol{\theta}^T\boldsymbol{\varphi}\;(t)\,. \tag{96}$$

$\mathbf{A}_m, \mathbf{K}_m, \mathbf{r}(t)$ is chosen to get desired trajectory and $\boldsymbol{\theta}$ is denoted $[\mathbf{A}_m, \mathbf{K}_m, 1]^T$, and $\phi\,(t)$ is defined by $[-\mathbf{x}, \mathbf{r}, N_f]^T$. N_f is approximated using RBFN network. Let us define the error.

$$\mathbf{e}(t) \cong\; \mathbf{x}\,(t) - \;\mathbf{x}_m\,(t) \tag{97}$$

If the neural network approximation exactly presents the function f, $N_f = f$ then, the error equation is simply written as

$$\dot{\mathbf{e}}(t) + \;\mathbf{A}_m\mathbf{e}(t) \;=\; 0\,. \tag{98}$$

However this is not true in real systems. The control objective, therefore, is $e(t) \to 0$ as $t \to \infty$. If we consider the neural network learning error term,

$$\Delta(\mathbf{x},\;\mathbf{w}) \cong\; N_f\;[\mathbf{x}(t),\;\mathbf{w}(t)] - f[\mathbf{x},t]\;. \tag{99}$$

Now we substitute eqn(99),(94) and (96) into eqn (93), then we get the closed loop system equation of

$$\dot{\mathbf{e}}(t) + \;\mathbf{A}_m\mathbf{e}(t) \;=\; \Delta(\mathbf{x},\;\mathbf{w}) \tag{100}$$

If we understand the neural network adaptive controller in this sense, as the learning error in eqn(100) goes to zero then the control error $e(t)$ also converge to zero. Therefore as long as we can decrease the error enough to close 0, then we can guarantee the convergence of the error in control signal. We can write the learning error using RBFN in terms of adjusting the weight parameter.[21,37]

$$f(\mathbf{x},t) = \sum_{i=1}^{h} \mathbf{w}_i\phi_i\left(x_m, \dot{x}_m, \sigma_i, \mu_i\right) = \mathbf{W}^{*T}\boldsymbol{\Phi}^* \tag{101}$$

where $\mathbf{W}$ is weight vector with estimate value of $\hat{\mathbf{W}}$, $\mathbf{\Phi}$ represent radial basis function and $\tilde{\mathbf{W}} = \hat{\mathbf{W}} - \mathbf{W}^*$ and $\tilde{\mathbf{\Phi}} = \hat{\mathbf{\Phi}} - \mathbf{\Phi}^*$. From this we can rewrite the error equation with

$$\begin{aligned} \dot{\mathbf{e}} &= \mathbf{A}_m \mathbf{e} + \left[\hat{\mathbf{W}} \hat{\mathbf{\Phi}} - \mathbf{W}^* \mathbf{\Phi}^* \right] \\ &\approx \mathbf{A}_m \mathbf{e} - \left[\tilde{\mathbf{W}} \hat{\mathbf{\Phi}} + \hat{\mathbf{W}} \tilde{\mathbf{\Phi}} \right] \end{aligned} \tag{102}$$

Let us define a candidate Lyapunov's function as

$$\mathbf{V} = \frac{1}{2} \mathbf{e}^T \mathbf{P} \mathbf{e} + \frac{1}{2} Tr \left[\tilde{\mathbf{W}}^T \mathbf{\Gamma}_1^{-1} \tilde{\mathbf{W}} \right] + \frac{1}{2} \tilde{\mathbf{\Phi}}^T \mathbf{\Gamma}_2^{-1} \tilde{\mathbf{\Phi}} \tag{103}$$

derivative of Lyapunov's function is

$$\dot{\mathbf{V}} = -\frac{1}{2} \mathbf{e}^T \mathbf{Q} \mathbf{e} + \tilde{\mathbf{\Phi}}^T \left(\mathbf{\Gamma}_2^{-1} \dot{\tilde{\mathbf{\Phi}}} - \mathbf{W} \mathbf{P} \mathbf{e} \right) + Tr \left[\tilde{\mathbf{W}} \left(\mathbf{\Gamma}_1^{-1} \dot{\tilde{\mathbf{W}}} - \mathbf{P} \mathbf{e} \hat{\mathbf{\Phi}} \right) \right] \tag{104}$$

To be stable ($\dot{\mathbf{V}} \le 0$), we can get the following adaptation control law

$$\dot{\tilde{\mathbf{\Phi}}} = \mathbf{\Gamma}_2 \hat{\mathbf{W}}^T \mathbf{P} \mathbf{e} \tag{105}$$

$$\dot{\tilde{\mathbf{W}}} = \mathbf{\Gamma}_1 \mathbf{P} \mathbf{e} \hat{\mathbf{\Phi}} \tag{106}$$

Derivation of the Control Law for SJA Pitch Moment (Junkins et al, 2003)

For plant model,

$$\begin{aligned} \dot{\alpha} &= q \\ \dot{q} &= \frac{\overline{qc}}{J} C_M (\alpha, M, \omega) \end{aligned} \tag{107}$$

For reference model,

$$\begin{aligned} \dot{\alpha}_r &= q_r \\ \dot{q}_r &= \frac{\overline{qc}}{J} C_M (\alpha_r, M_r, \omega_r) \end{aligned} \tag{108}$$

where, α is the Angle of Attack, M is the Mach Number, q is the Pitch rate, q is the wing loading, c is the chord and J is the Moment of Inertia. Subscript r corresponds to the reference values. Moment coefficient can be modeled as

$$C_M(\alpha, M, \omega) = C_{M_0} + C_{M_\alpha}\alpha + C_{M_\omega}\omega + \bar{C}_M(\alpha, M, \omega). \tag{109}$$

$\bar{C}_M(\alpha, M, \omega)$ is the higher order term and can be approximated with radial basis function network as we already did in Chapter V using several algorithms. Also we can get the approximate coefficient of the linear terms from the linear least square results. Higher order terms can be expressed as

$$\bar{C}_M(\mathbf{x}) = \sum_{i=1}^{h} w_i \exp\left(-\frac{(\mathbf{x}-\mu_i)^T R^{-1}(\mathbf{x}-\mu_i)}{2}\right) + \varepsilon_h = \mathbf{W}^T\mathbf{\Phi} + \varepsilon_h. \tag{110}$$

Let us define the tracking errors of the pitch dynamics.

$$e_\alpha = \alpha - \alpha_r, \quad e_q = q - q_r \tag{111}$$

We can take the derivation of the errors and with the RBFN correction, closed loop error dynamics becomes

$$\dot{e}_\alpha = e_q \tag{112}$$

$$\dot{e}_q = -(C_{M\alpha}K_\alpha - C_{M\omega})e_\alpha - C_{M\omega}K_q e_q + C_{M\omega}\omega_{NN} + \mathbf{W}^{\mathbf{T}}\mathbf{\Phi}^T + \varepsilon \tag{113}$$

where $C_{M\alpha} - C_{M\omega}K_\alpha < 0, \quad C_{M\omega}K_q > 0$ and $\omega = \omega_a + \omega_{NN}$.

The equation can be written in vector form as

$$\dot{\mathbf{x}} = \mathbf{A}_m\mathbf{x} + \mathbf{B}(-\hat{\bar{\mathbf{W}}}^T\hat{\mathbf{\Phi}} + \bar{\mathbf{W}}^T\mathbf{\Phi} + \bar{\varepsilon}) \tag{114}$$

$$\mathbf{A}_m = \begin{bmatrix} 0 & 1 \\ -(C_{M\alpha}K_\alpha - C_{M\omega}) & -C_{M\omega}K_q \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ C_{M\omega} \end{bmatrix}. \tag{115}$$

If we formulate the same Lyapunov function in Chapter VI and apply the tuning rule we derived Chapter VI, fourth section, we get the control law

$$\omega = \omega_r + C_{M\omega}^{-1}\,\mathbf{W}^T\boldsymbol{\Phi}\Big|_r - K_\alpha e_\alpha - K_q e_q - C_{M\omega}^{-1}\hat{\mathbf{W}}^T\hat{\boldsymbol{\Phi}}\,. \tag{116}$$

For weight vector and RBFN, we have the adaptive update laws

$$\dot{\hat{\mathbf{w}}}_i = \boldsymbol{\Gamma}_1\hat{\boldsymbol{\Phi}}\mathbf{B}_i^T\mathbf{P}\mathbf{x}, \quad i = 1,\ldots p \tag{117}$$

$$\dot{\hat{\boldsymbol{\Phi}}} = \boldsymbol{\Gamma}_2\hat{\mathbf{W}}\mathbf{B}^T\mathbf{P}\mathbf{x} \tag{118}$$

Simulation Result of Neural Network Adaptive Controller with SJA Pitch Moment Coefficient Data

In this simulation example, we use the model reference adaptive neural network control law introduced in Chapter VI. For simplification, scalar dynamic model of $\dot{x} = f(x,t) + g(x)u + e(t)$ is estimated on-line using Gaussian type RBFN with inputs of angle of attack and SJA frequency and output of the neural approximation function with these inputs. In error dynamics, error is defined as between pitch moment and its reference models. The control input *u* includes SJA frequency. The reference signal is smooth and bounded input signal along [-1 0]. The number of hidden units is preset to 100 with a spread of 1. The centers are distributed uniformly between the range of AOA and SJA frequency. We apply only weight adapted rule to see if the on line neural network controller does converge. The weight vector is adjusted by above tuning rule and simulation time is 100s. Γ value is 0.01. This model is successfully converged with RBFN controller in the simulation (figure 6.2). The final converged error is 0.0185 and

the measured controller output is shown in figure 6.3, which is almost linear.

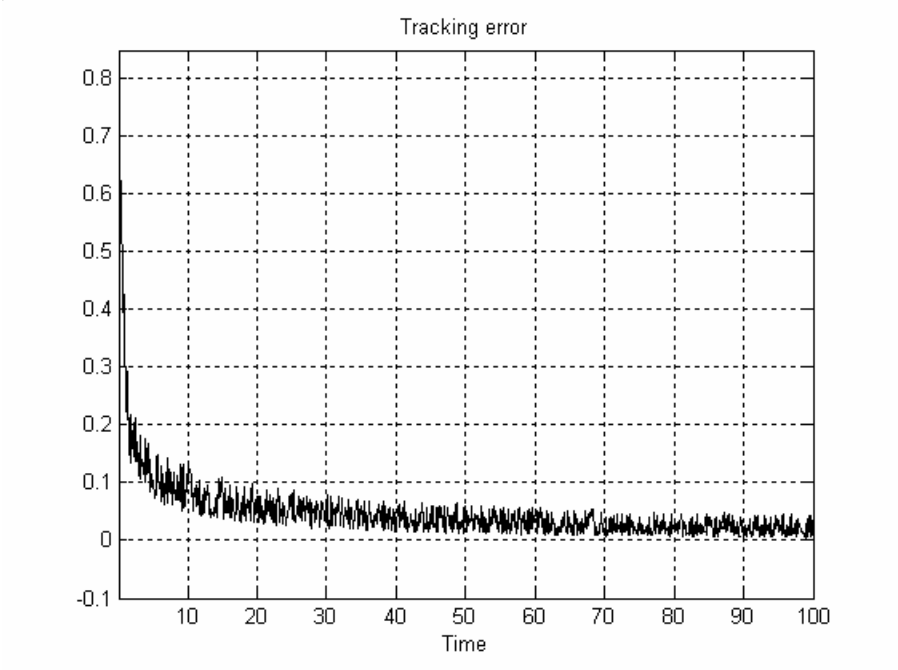

Figure 6.2 Tracking error

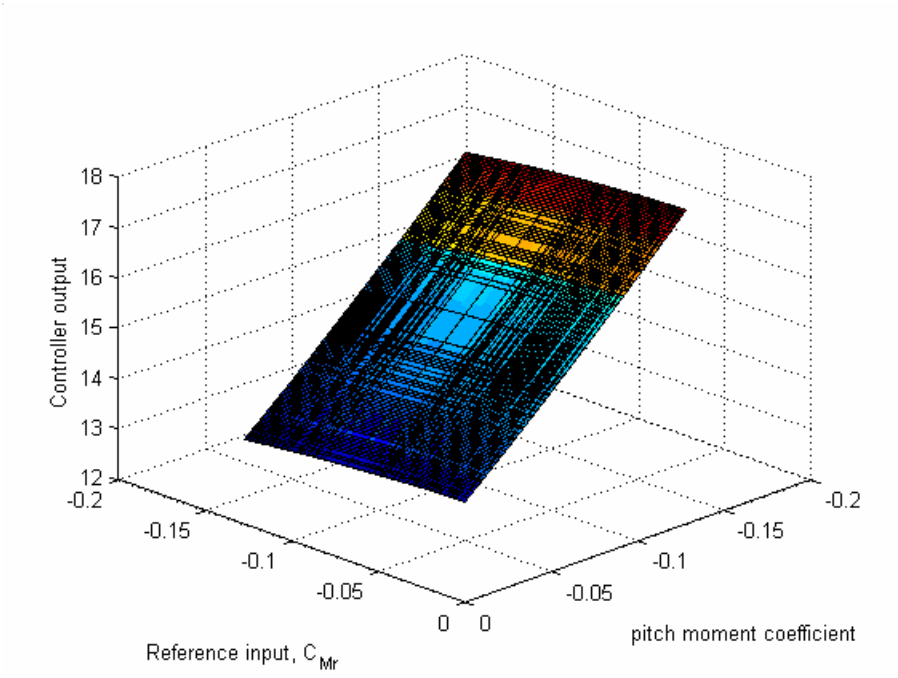

Figure 6.3 Controller inputs vs. adaptive neural controller output

CHAPTER VII

SUMMARY AND CONCLUSION

Summary

The main goal of this thesis has been to develop a systematic model which emphasizes the computational aspects of neural networks in highly nonlinear function approximation and set up the control law using neural networks. The underlying premise is that real-time learning from input-output measurements (not on parametric model physics based) will converge adequately for stable control on many problems. Clearly the ability of the neural models and update laws strongly affect the validity of this premise. In the approach adopted the neurocontrollers are composed of two steps. First a neural network is used to estimate the dynamic model (could be off-line) and then another network is adapted in real time to estimate the inverse dynamics of the system. The main focus in this thesis is the first part. We studied existing and modified the function approximation using several learning algorithm to adjust the SJA parameters so that we could see the possibility of how close we can efficiently approach real system model. The highly nonlinear and nonparameteric nature of the SJA modeling problem is well suited for using a Radial Basis Neural Network.

We presented learning algorithms using RBFNs that allow improvement of the generalization performance of single linear regression estimator. In the first algorithm of FS-ROLS, Generalized Cross Validation is used for choosing the variance and an automatic tuning approach for the stabilization parameter λ for generalization is

introduced. In the second algorithm of RAN-EKF the adaptation parameters include the centers, weights and variances; these are adapted by EKF. The advantage of the RAN and FS-ROLS are that they learn quickly to forms a more compact representation. The simulation and analysis showed that RBFN with the algorithm of FS-ROLS works better than RAN-EKF in the application of the SJA wing data. However both algorithms matches well in SJA modeling, and are much better than the algorithm without learning and this enhanced RBFN also showed that great results compared with ones from a multilayer neural network such as backpropagation. This means that RBFN has broad potential for modeling nonlinear dynamical systems. This RBFN approach may prove practical if it is combined with an adaptive controller for real time model. However, there remain unknown factors that could effects SJA performance by changing their subtle action. In particular, poor controllability near low angle of attack suggests no control approach can yield truly desirable result. It may be that a new, more controllable experimental configuration is required. As of now it is very hard to determine what are the shortcomings of our approach which results in poor controllability of the system. There should be more experiments with the SJA wing, including improved designs and more studies of available RBFN approaches.

Conclusions

Existing training algorithms RAN and FS-ROLS are simulated for SJA modeling purposes. There are several important and desirable attributes to result in good model learning. The method must be computationally efficient, converge fast and

accurately, also the approach should form a compact representation. There are several algorithms that may achieve these objectives, but for high dimensional problem all face obstacles and require more analytical computational studies. The RAN and FS-ROLS algorithm can find the desirable solutions in less computational time compared to other neural networks. These developments described in this thesis lead to a several successful implementation, reasonable errors and convergence rates for the SJA wing data set. If the dimensions of the state, control and parameter space is high then we are not optimistic any of the approaches studied will be practical without further refinement. However as presented in this thesis for up to 4 dimensioned state and control spaces, these methods do appear practical. As an important connection to this class of inverse dynamic model off-line learning algorithms, we introduce the neurocontrollers to adjust on line to track prescribed reference trajectories. As a consequence of the non-affine nature of the dependence on the control vector, linear approximation of the control dependence was necessary to complete the formulation. The controller type in Chapter VI is model reference direct adaptive controller and to compensate the nonlinearities in the plant radial basis function network is used. The simulation results in the Chapter VI tell us that the discrepancy of reference and actual plant model can be used as a signal for parameter adjusting law. A reasonable convergence rate is achieved. However no criterion was found in this work or the literature on the convergence of centers and variances of RBFN (Jacob, 1997). They are continuously adjusted, and as a consequence, the RBFN gives only local approximation. To apply these ideas to real systems, we need better understanding of the theoretical aspects to accept or reject the locally affine

control approximation. The neural network adaptive controller structure design combined with a pre-trained neural network can greatly increase the potential for applications of intelligent and reconfigurable control.

Recommendations

The recommendation for future work is that we need to develop more sophisticated and refined radial basis neural network approximation algorithm to minimize the number of Gaussian basis functions which can result much improvement in the previous simulation of SJA. This refined method should better manage to localize learning, high dimensionality and uncertainty of model. More significant, higher dimensional dynamical systems should be studied.

With the aspect to adaptive control approach, validation of the control law derived in Chapter VI by experiment and simulations should be performed and further attention to other means of accommodating the case of non affine control.

REFERENCES

[1] Chen, S., Cowman, C., and Grant, P., "Orthogonal least squares learning algorithm for radial basis function networks" *IEEE Transaction on Neural Networks*, Vol. 2, 1991, pp. 302–309.

[2] Mark J. L. Orr. "Regularised centre recruitment in radial basis function networks", Research Paper 59, Center for Cognitive Science, Edinburgh University, Edinburgh, Scotland, 1993.

[3] Bishop, C., "Improving the generalization properties of radial basis function neural networks", *Neural Computation*, Vol. 3(4), 1991, pp.579-588.

[4] Broomhead, D. S., and Lowe, D., "Multivariate functional interpolation and adaptive networks", *Complex systems*, Vol. 2, 1998, pp. 321-355.

[5] Platt, J., "A resource allocating network for function interpolation," *Neural Computation*, Vol. 3. 1992, pp. 213-225.

[6] Moody, J. and Darken, C., "Fast learning in network of locally-tuned processing units", *Neural Computation*, Vol. 1, 1989, pp. 281–294.

[7] Arisariyawong, S., and Charoenseang, S., "Self-organized learning in complexity growing of radial basis function networks", Mechnical Engineering Department, King Mongkut's University of Technology Thonburi, Bangkok, Thailand, 2002.

[8] Bose, N.K., and Liang, P., *Neural network fundamentals with graphs, algorithms and applications*, McGraw Hill International Editions, New York, NY, 1996.

[9] Irwin, G.W., Warwick, K., and Hunt, K.J., *Neural Network Applications in Control*,

IEEE Control Engineering Series, Vol.53, London, U.K, 1995.

[10] Kadirkamanathan, V., and Niranjan, M., "A function estimation approach to sequential learning with neural networks", Cambridge University Engineering Department Technical Report, 1993.

[11] Spooner, J. T., Maggiore, M., Ordonez, R., and Passino, K. M., *Stable Adaptive Control and Estimation for Nonlinear Systems*, Wiley Interscience, New York, NY, 2002.

[12] Norgaard, M., Ravn, O., Poulsen, N.K., and Hansen, L.K., *Neural Networks for Modelling and Control of Dynamic Systems*, Springer, New York, NY, 2000.

[13] Moody, J., and Darken, C. J., "Fast learning in networks of locally-tuned processing units", *Neural Computation*, Vol.1, 1989, 281-294.

[14] Junge, T. F., and Unbehauen, H., "On-line identification of nonlinear time variant systems using structurally adaptive radial basis function networks", *Proceedings of the American Control Conference*, Albuquerque, New Mexico June 1997.

[15] Warwick, K., and Craddock, R., "An introduction to radial basis function for system identification a comparison with other neural network methods", *Proceedings of the 36th Conference on Decision and Control*, Kobe, Japan, December 1996, pp 464-469.

[16] Reisenthel, P. H., "Development of a nonlinear indicial model using response function generated by a neural network", *AIAA*, Vol.3, 1997, pp. 1-13.

[17] Park, J., and Sandberg, I. W., "Approximation and radial basis function networks", *Neural Computation* Vol.5, 1993, pp. 305-316

[18] Astrom, K. J., and Wittenmark, B., *Adaptive Control*, Addison-Wesley, Reading,

MA, 1989.

[19] Sundararajan, N., Saratchandran, P., Li, Y., *Fully Tuned Radial Basis Function Neural Networks For Flight Control*, Kluwer Academic Publisher, Boston, MA, 2002.

[20] Saad, M., Dessaint, L. A., Bigras, P., and Al-Haddad, K., "Adaptive versus neural adaptive control: application to robotics", *International Journal of Adaptive Control and Signal Processing*, Vol. 8, 1994, 223-236.

[21] Patino, H.D., and Liu, D., "Neural network-based model reference adaptive control system" *IEEE Transaction on Systems*, man and cybernetics-part B: Cybernetics, Vol. 30, No.1, February 2000, pp. 198-204.

[22] Lewis., F. L., and Parisini, T., "New developments in neurocontrol", *Proceedings of the 1998 IEEE, International Conference in Control Applications*, Trieste, Italy 1-4 September 1998.

[23] Narendra, K. S., "Neural networks for control: theory and practice", *Proceedings of The IEEE*. Vol. 84, No.10, October 1996, pp.1385-1403.

[24] Narendra, K. S., and Mukhopadhyay, S., "Adaptive control using neural networks and approximate models", *IEEE Transactions on Neural Networks*, Vol.8, No. 3, May 1997, pp. 355-359.

[25] Smith, R. M., and Sbarbaro, D., "Nonlinear adaptive control using nonparametric gaussian process prior models" *Technical paper in Department of Computing Science,* University of Glasgow, Glasgow, Nov., 2002.

[26] Chiang, C. Y., and Youssef, H. M., "*Neural network approach to aerodynamic coefficients estimation and aircraft failure isolation design*" *AIAA*, Vol.3, 1994, pp. 500-

509.

[27] Yingwei, L., Sundararajan, N., and Saratchandran, P., “A sequential learning scheme for function approximation using minimal radial basis function neural networks”, *Neural Computation* Vol.9, 1997, pp. 461-478.

[28] Behera, L., Gopal, M., and Chaudgury, S., “Adaptive control of nonlinear systems”, *IEE Proc.-Control Theory Appl*., Vol. 142, No. 6, November 1995, pp. 617-624.

[29] Golub,G. H., Heath, M., and Wahba., G., “Generalised cross-validation as a method for choosing a good ridge parameter”, *Technometrics*, 21(2), 1979, pp. 215-223.

[30] Sarle, W. S., “Neural networks and statistical models”, In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC, 1994, pp.1538-1550.

[31] Calise, A. J., “Neural networks in nonlinear aircraft flight control”, *IEEE AES Systems Magazine,* July 1996, pp 5-10.

[32] Li, Y., Sundararajan, N., and Saratchandran, P., “Stable neuro-flight-controller using fully tuned radial basis function neural networks”, *Journal of Guidance, Control, and Dynamics*, Vol.24, No.4, July-August 2001, pp. 665-674.

[33] Subbarao, K., and Junkins, J. L., “Structured model reference adaptive control for a class of nonlinear systems”, *Journal of Guidance, Control, and Dynamics*, Vol. 26, No. 4, July-August 2003, pp. 551-557.

[34] Narendra., K. S., and Mukhopadhyay, S., “Adaptive control using neural networks and approximate models”, *Proceedings of the America Control Conference* Seattle, Washington, June 1995, pp. 475-485.

[35] Kalkkuhl, J., Hunt, K. J., Zbikowski, R., and Dzielinski, A., *Applications of neural adaptive control technology*, *World Scientific Series in Robotics and Intelligent Systems*, Vol.17, River Edge, N.J., 1997.

[36] Spooner, J. T., and Passino, K. M., "Decentralized adaptive control of nonlinear systems using radial basis neural networks", *IEEE Transactions on Automatic Control*, Vol. 44, No. 11, 1999, pp 2050-2057.

[37] Ordonez, R., and Passino, K. M., "Adaptive control for a class of nonlinear systems with a time-varying structure", *IEEE Transactions on Automatic Control*, Vol.46, No.1, January 2001, pp. 1-13.

APPENDIX

A. Automatic Estimation of λ

Using regularization as well as using Generalized Cross Validation can decrease the overfit more. Simple re-estimation formula which is integrated into ROLS for letting the data choose a value for the regularization parameter is derived (Mark J. L. Orr, 1996) here.

Let us start from the eqn(40). Differentiating eqn(40) with respect to λ set the equation to zero to get the minimum value.

$$\mathbf{y}^T \tilde{\mathbf{P}}_m \frac{\partial \tilde{\mathbf{P}}_m \mathbf{y}}{\partial \lambda} trace(\tilde{\mathbf{P}}_m) = \mathbf{y}^T \tilde{\mathbf{P}}_m^2 \mathbf{y} \frac{\partial trace(\tilde{\mathbf{P}}_m)}{\partial \lambda} \tag{119}$$

From the eqn(36), we can substitute the $\tilde{\mathbf{P}}_m$ value to left hand side of eqn(119) and then we get

$$\mathbf{y}^T \tilde{\mathbf{P}}_m \frac{\partial \tilde{\mathbf{P}}_m \mathbf{y}}{\partial \lambda} = \lambda \tilde{\mathbf{w}}_m^T (\tilde{\mathbf{H}}_m^T \tilde{\mathbf{H}}_m + \lambda \mathbf{I}_m)^{-1} \tilde{\mathbf{w}}_m . \tag{120}$$

where $\mathbf{y}^T \tilde{\mathbf{P}}_m^2 \mathbf{y} = \mathbf{y}^T \mathbf{y} - \sum_{j=1}^{m} \frac{(2\lambda + \tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j)(\mathbf{y}^T \tilde{\mathbf{h}}_j)^2}{(\lambda + \tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j)^2}$ and $trace(\tilde{\mathbf{P}}_m) = p - \sum_{j=1}^{m} \frac{\tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j}{(\lambda + \tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j)^2}$.

We substitute $\lambda \tilde{\mathbf{w}}_m^T (\tilde{\mathbf{H}}_m^T \tilde{\mathbf{H}}_m + \lambda \mathbf{I}_m)^{-1} \tilde{\mathbf{w}}_m$ in eqn(119) and rearrange it with respect to λ, to obtain

$$\lambda = \frac{[\partial trace(\tilde{\mathbf{P}}_m) / \partial \lambda] \mathbf{y}^T \tilde{\mathbf{P}}_m^2 \mathbf{y}}{trace(\tilde{\mathbf{P}}_m) \tilde{\mathbf{w}}_m^T (\tilde{\mathbf{H}}_m^T \tilde{\mathbf{H}}_m + \lambda \mathbf{I}_m)^{-1} \tilde{\mathbf{w}}_m} \tag{121}$$

where, $\frac{trace(\tilde{\mathbf{P}}_m)}{\lambda} = \sum_{j=1}^{m} \frac{\tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j}{(\lambda + \tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j)^2}$ and $\tilde{\mathbf{w}}_m^T (\tilde{\mathbf{H}}_m^T \tilde{\mathbf{H}}_m + \lambda \mathbf{I}_m)^{-1} \tilde{\mathbf{w}}_m = \sum_{j=1}^{m} \frac{(\mathbf{y}^T \tilde{\mathbf{h}}_j)^2}{(\lambda + \tilde{\mathbf{h}}_j^T \tilde{\mathbf{h}}_j)^3}$.

Using the eqn(126), new λ value can be evaluated after each forward selection step from the previous value and initial value of λ is 0.

B. Some useful dynamic theory for formulating Neural Network adaptive control laws.

Dynamic Inversion

This technique is for control law design in which feedback rule is used to simultaneously cancel system dynamics and achieve desired response characteristics.

$$\dot{\mathbf{x}} = f(\mathbf{x}) + G(\mathbf{x})\mathbf{u} \tag{122}$$

$$\mathbf{u} = G^{-1}(\mathbf{x})(-f(\mathbf{x}) + \dot{\mathbf{x}}_\mathbf{d}) \tag{123}$$

$\mathbf{x}$ is state vector and control law $\mathbf{u}(m\times 1)$ which yield the desired state $\dot{\mathbf{x}}_\mathbf{d}$. In this way, dynamic inversion control law presents an attractive alternative to the system with complicated nonlinearities. However we need exact knowledge of $f(\mathbf{x})$ and $G(\mathbf{x})$ to solve the equation in terms of controls, and this seldom feasible in practice due to model errors.

Feed back linearization

Feed back linearization is one approach for nonlinear control design. The idea is to algebraically transform a nonlinear system dynamics into a fully or partially linear one so that linear control techniques can be applied. Consider the single input single output nonlinear systems.

$$\frac{d}{dt}\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ \vdots \\ x_n \\ f(\mathbf{x}) + g(\mathbf{x})u \end{bmatrix}, \quad x^{(n)} = f(\mathbf{x}) + g(\mathbf{x})u \tag{124}$$

Let the scalar control input u as

$$u = \frac{1}{g}(v - f) \tag{125}$$

We can cancel the nonlinearities and obtain an input output relation

$$x^n = v \tag{126}$$

$$v = x_d^{(n)} - k_1 e - k_2 \dot{e} - \cdots - k_{n-1} e^{(n-1)} \tag{127}$$

$$e = x(t) - x_d(t) \tag{128}$$

where e is tracking error. The key assumption of feedback linearization is that the system nonlinearities are known a priori and of course, this the key source of non-robust behavior is practice.

C. Implementation of the Tuning rule

Firstly, let us see the tuning rule for weight

$$\begin{aligned} &\dot{\hat{w}}_i = \hat{\phi}(\mathbf{Pe})_i \quad i = 1,...,n \\ &\Rightarrow \dot{\hat{\mathbf{W}}}^T = \mathbf{Pe}\hat{\phi}^T \Rightarrow \dot{\underline{\hat{w}}}_k^T = \mathbf{Pe}\hat{\phi}_k,\ k = 1,...,h \end{aligned} \tag{129}$$

Since the estimate value of $\hat{\phi}_k$ is the derivative of $\hat{g}()$ to the weight $\underline{\hat{w}}_k$ this equation can be converted into a discrete form.

$$\underline{\hat{\mathbf{w}}}_{\mathbf{k}}^{\mathbf{T}}(n'+1) = \underline{\hat{\mathbf{w}}}_{\mathbf{k}}^{\mathbf{T}}(n') + \tau \frac{\partial \hat{g}}{\partial \underline{\hat{w}}_k^T} \mathbf{Pe}(n'). \tag{130}$$

where $n' = n+m$.

The tuning rule for centers and width are from the eqn (65)

$$\dot{\hat{\phi}}_k = \underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe} \tag{131}$$

$$\hat{\phi}_k = \exp(-\frac{1}{\sigma_k^{*2}} \| \boldsymbol{\xi} - \boldsymbol{\mu}_\mathbf{k}^* \|^2) \tag{132}$$

for given input $\boldsymbol{\xi}$, derivative of equation (131) can be expressed like

$$\dot{\hat{\phi}}_k = \frac{\partial \hat{\phi}_k}{\partial \hat{\boldsymbol{\mu}}_k^T} \dot{\hat{\boldsymbol{\mu}}}_k + \frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k} \dot{\hat{\sigma}}_k . \tag{133}$$

To obtain the tuning rule Equation (133) can be partitioned to

$$\frac{\partial \hat{\phi}_k}{\partial \hat{\boldsymbol{\mu}}_k^T} \dot{\hat{\boldsymbol{\mu}}}_k = \lambda \underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe} \tag{134}$$

$$\frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k} \dot{\hat{\sigma}}_k = (1-\lambda)\underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe}, 0 < \lambda < 1 \tag{135}$$

Set $\mathbf{r_k} = (\frac{\partial \hat{\phi}_k}{\partial \hat{\boldsymbol{\mu}}_\mathbf{k}^\mathbf{T}})^T = \frac{\partial \hat{\phi}_k}{\partial \hat{\boldsymbol{\mu}}_\mathbf{k}}$ with column vector of $n+m$ and then tuning rule for center is

$$\begin{aligned} \dot{\hat{\boldsymbol{\mu}}}_k &= \frac{\lambda}{n+m} \mathbf{r_k}^{(-1)} \underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe} \\ &= \frac{\lambda}{n+m} (\mathbf{r_k}^{(-2)} \cdot \frac{\partial \hat{\phi}_k}{\partial \hat{\boldsymbol{\mu}}_\mathbf{k}}) * \underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe} \end{aligned} \tag{136}$$

Similarly a tuning rule for width can be derived.

$$\begin{aligned} \dot{\hat{\sigma}}_k &= (1-\lambda)(\frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k})^{-1} \underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe} \\ &= (1-\lambda)(\frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k})^{-2} \frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k} \underline{\mathbf{w}}_\mathbf{k}^\mathbf{T}\mathbf{Pe} \end{aligned} \tag{137}$$

Let $\alpha = \frac{\lambda}{n+m} \mathbf{r_k}^{(-2)}$, $\beta = (1-\lambda)(\frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k})^{-2}$ and converted to discrete form then,

$$\begin{aligned}\hat{\mu}_k(n'+1) &= \hat{\mu}_k(n') + \tau(\alpha .* \frac{\partial \hat{\phi}_k}{\partial \hat{\mu}_k})\underline{\mathbf{w}}_{\mathbf{k}}^{\mathbf{T}}\mathbf{Pe}(n') \\ &\approx \hat{\mu}_k(n') + \eta_1 \frac{\partial \hat{\phi}_k}{\partial \hat{\mu}_k}\underline{\mathbf{w}}_{\mathbf{k}}^{\mathbf{T}}\mathbf{Pe}(n')\end{aligned} \tag{138}$$

$$\begin{aligned}\hat{\sigma}_k(n'+1) &= \hat{\sigma}_k(n') + \tau\beta \frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k}\underline{\mathbf{w}}_{\mathbf{k}}^{\mathbf{T}}\mathbf{Pe}(n') \\ &\approx \hat{\sigma}_k(n') + \eta_2 \frac{\partial \hat{\phi}_k}{\partial \hat{\sigma}_k}\underline{\mathbf{w}}_{\mathbf{k}}^{\mathbf{T}}\mathbf{Pe}(n')\end{aligned} \tag{139}$$

η_1, η_2 are positive scalar to be selected properly a priori. From the above equation, we can generalize the tuning rule.

$$\chi(n'+1) = \chi(n') + \eta\mathbf{\Pi}(n')\mathbf{Pe}(n'), \tag{140}$$

where $\mathbf{\Pi}(n') = \nabla_{\chi}\hat{g}(\xi_n)$.

VITA

Hee Eun Lee is the daughter of Pil Gu Lee and Sung Jin Lee, Republic of Korea. She finished her B.S degree in Physics and Astronomy in Yonsei University in Seoul, Korea in February, 2002. After that she attended Texas A&M University in the United states. She studied under Dr. John L. Junkins receiving her master's degree with emphasis on dynamics and controls in the field of aerospace engineering. Hee Eun Lee's permanent address is 1258 Sangamdong Mapogu Seoul, Korea