Optimization of Turbine Blade Design for Reusable Launch Vehicles

supplement to

Advanced Nuclear Propulsion Studies

Final Report

submitted by

Wei Shyy

Reporting Period: 9/28/98 - 12/31/98

University of Florida
POB 116250
Gainesville FL 32611-6250

NAG8-1251

ABSTRACT

To facilitate design optimization of turbine blade shape for reusable launching vehicles, appropriate techniques need to be developed to process and estimate the characteristics of the design variables and the response of the output with respect to the variations of the design variables. The purpose of this report is to offer insight into developing appropriate techniques for supporting such design and optimization needs. Neural network and polynomial-based techniques are applied to process aerodynamic data obtained from computational simulations for flows around a two-dimensional airfoil and a generic three-dimensional wing/blade. For the two-dimensional airfoil, a two-layered radial-basis network is designed and trained. The performances of two different design functions for radial-basis networks, one based on the accuracy requirement, whereas the other one based on the limit on the network size. While the number of neurons needed to satisfactorily reproduce the information depends on the size of the data, the neural network technique is shown to be more accurate for large data set (up to 765 simulations have been used) than the polynomial-based response surface method. For the three-dimensional wing/blade case, smaller aerodynamic data sets (between 9 to 25 simulations) are considered, and both the neural network and the polynomial-based response surface techniques improve their performance as the data size increases. It is found while the relative performance of two different network types, a radial-basis network and a back-propagation network, depends on the number of input data, the number of iterations required for radial-basis network is less than that for the back-propagation network.

NOMENCLATURE

a : output of the network

AR : aspect ratio

b : bias vector

C_L : lift coefficient

$C_L^{3/2}/C_D$: power index

C_D : drag coefficient

e : error

f : transfer function

J : Jacobian matrix

p : input of the network

Re : Reynolds numbers

W : weighting coefficient matrix

y_c :camber

y_t : thickness ratio

α : angle of attack

1. INTRODUCTION

With the rapid progress in computer-assisted data processing techniques including those generated from computational fluid and structure dynamics codes, and experimental measurements, one often encounters the task of handling a large number of data for analysis, synthesis, and design optimization. Traditionally, the response surface method (RSM), Meyer and Montgomery[1] have proved to be a valuable tool for such purposes. The RSM typically employs the least squares method construct a lower-order polynomial, typically a quadratic function. It can also make reasonable estimates regarding statistical uncertainties. Recently, the neural network method has experienced rapid progress and become an efficient tool in data analysis. They have several advantages that make them efficient tools: First, they are adaptive; therefore, they can learn from existing data. Second, they can generalize after training and can process the data even if they are incomplete, providing a measure of fault tolerance. Third, they are nonlinear in that they

can capture complex interactions among the input variables in a system. Fourth, neural networks are highly parallel and thus can be executed much faster than conventional microprocessors and digital signal processors without losing accuracy[2-3]. Due to these promising features, it has been extensively used in many areas including aeronautical engineering applications[4-11]. Norgaard et at.[4] presented the feasibility of reducing wind tunnel test times by using neural networks to interpolate between measurement and showed that significant cost saving is realized. Reducing wind tunnel data requirements to completely define the aerodynamic performance of a tunnel model by using neural network has been studied by Ross et al[5]. Another effort has been conducted by Protzel et al.[6] to apply neural nets for optimization problems. Sparks and Maghami[7-8] illustrated the efficiency of using neural networks in simulation of nonlinear components for a reaction wheel model[7] and in approximating performance characteristics of a spacecraft[8]. Rai and Madavan[9] studied on aerodynamic design procedure for turbine blade design that incorporates the advantages of both traditional response surface methodology and neural networks and demonstrated the efficiency of using such a procedure. Similarly, Capenter and Barhelemy's paper[10] also applied to both neural nets and polynomial based response surface for several optimization problems and made performance comparison for these two methods. A preliminary effort in using neural networks for time dependent models which predict unsteady fluid flows have been accomplished by Faller and Schreck[11]. The results illustrated that the neural networks can be used to both predict and control unsteady aerodynamics.

In the present work, we assess the relative merits of the response surface method and the neural network method by focusing on the aerodynamic data generated for a two-dimensional airfoil and a three-dimensional wing/blade model. For the two-dimensional airfoil, the simulated data obtained from computational fluid dynamics tool, XFOIL code[12]. The airfoil chosen is CLARK-Y[13] and the simulated data include lift coefficient, C_L, and drag coefficient, C_D, at various Reynolds numbers and angles of attack. Two-dimensional computations based on the coupled inviscid and thin layer flow equations are conducted by Shyy et al.[13]. The purpose is to correlate the aerodynamic performance, measured by aerodynamic efficiency, C_L/C_D, obtained by varying the angle of attack and the Reynolds numbers. The wing/blade model is comprised of a potential flow solver,

based on the PMARC code[14], and a coupled inviscid-viscous flow solver, based on the XFOIL. The lift coefficients, C_L, and drag coefficients, C_D, versus camber, y_c, aspect ratios, AR, and angles-of-attack, α, at fixed Reynolds numbers, Re, and thickness ratio, y_t are obtained from the model. The purpose is to first correlate the aerodynamic performance, measured by power index, $C_L^{3/2}/C_D$, obtained by varying the angle of attack for a given wing and airfoil shape and then identify possible strategies for wing and airfoil optimization. In the present report, we will concentrate on the first part only. In addition to the assessments made for the response surface method and the neural network method, the performance of two different neural network types, radial-basis and back-propagation networks are compared.

2. BACKGROUND OF NEURAL NETWORKS

Neural networks are massively parallel computational systems compromised of simple nonlinear processing elements with adjustable interconnections. They are inspired by and modeled after the learning capability of the biological nervous systems of the brain. The most important property of the neural networks lies in the fact that they are capable of modeling the underlying system dynamics by learning from previous experiences, just as the brain does. The learning process in essence adjusts the weights on the internal connections of the neural networks through a pre-defined training algorithm. The processing ability of the network is stored in the inter-unit connection strengths or weights obtained by a process of adaptation to, or learning from, a set of training patterns[15].

The neuron model and the architecture of a neural network describe how a network transforms its input to an output. In Figure 1, a neuron with a single input and bias is shown. The input, p, is transmitted through a connection that multiplies its strength by weight, w, to form the product w*p. The bias, b, is similar to a weight except that it has a constant input of 1. The effect of the product w*p and b are added at the summing junction to form the net input, n, of the transfer function, F, to be discussed later. The output of the neuron is a=F(w*p+b). Note that both w and b are adjustable scalar parameters of the neuron to train the network to perform a particular function. In Figure 1, a single neuron with R inputs is also shown. Here the inputs, p(1), p(2),..., p(R) are biased

by the weight elements, w(1,1), w(1,2),..., w(1,R) and the weighted values are inputs to the summing junction. Again, a single bias b is used.

In Figure 2, a single layer network with R inputs and S neurons is shown. A layer of network includes the combination of weights, the multiplication and summing operations, the biases and the transfer functions. The array of inputs is not included in a layer. A network can have a number of layers as shown in Figure 2. Each layer has a weight matrix, a bias vector and an output vector. Each layer can be analyzed as a one-layer network noting that the outputs of each intermediate layer are the inputs to the following layer. The layers of multilayer network play different roles. A layer that produces the network output is called an *output layer*. All other layers are called as the *hidden layers*.

The general process of training a neural network can be summarized as follows:

1. *Initialization:*

Initialization is required to generate initial values of weights and biases. If it is multi-layered network, then it takes a matrix of input vectors and sizes and transfer functions of each layer and returns weight and bias for each layer.

2. *Learning:*

In learning stage the network error, the difference between the neuron response and the target vector, is calculated. Based on this error, new values of weights and biases are assigned.

3. *Simulation:*

The appropriate transfer function is applied to transfer input to output.

4. *Training:*

Training is completed by applying *simulation* and *learning* steps repeatedly to present the inputs and to change the weight and bias according to the error so that the network can eventually find weight and bias values that solve the problem.

Further reading on neural networks can be found in References[16-22]. In this study, *Radial-basis Networks* and *Back-propagation Networks* are going to be applied to construct the neural network-based response surface by using Matlab[17].

(i) Back-propagation Networks

Back-propagation networks are created by generalizing the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. These networks are two or more-layer networks with hidden layers of sigmoid transfer function and a linear output layer as shown in Figure 3. In this study, only one hidden layer is considered. An R×1 input vector, p, and the corresponding S2×1 output vector, a, are used to train the network until it can approximate a continuous function to any degree of accuracy[15]. The output layer has S2 nodes or neurons, corresponding to the elements of output vector and S1 is the number of neurons in the hidden layer. The network output equation for such a single hidden layer and a linear output layer back-propagation network is given by

$$a2 = W2(f(W1\Delta+b1))+b2 \qquad (1)$$

where

- W1 : S1×R weighting coefficient matrix for hidden layer
- W2 : S2×S1 weighting matrix for output layer
- b1 : bias vector for hidden layer
- b2 : bias vector for output layer
- Δ : S1×1 matrix denoting the collection of R×1 input vector, p
- a2 : S2×1 output of the network
- f : transfer function for the hidden layer

Back-propagation networks often use the log-sigmoid transfer function or tan-sigmoid transfer function. The transfer functions of the back-propagation networks should be differentiable. In this study, tan-sigmoid transfer is considered and it is properties shown in Figure 3. The tan-sigmoid function itself and a single input neuron with bias when the tan-sigmoid function is applied are also presented in Figure 3.

In back-propagation network design, the number of neurons in the hidden layer is an important parameter. It should be chosen large enough to have convergence of the

network to the functional relationship but not too large to cause overmapping. Once it has been chosen, the network design is reduced to adjusting the weighting coefficient matrices, W1 and W2 and the weighting bias vectors, b1 and b2. These parameters for back-propagation networks are usually adjusted using a gradient method, named gradient method or a pseudo Newtonian approach such as Levenberg-Marquardt technique. In Matlab, back-propagation networks can be trained by using three different training functions, *trainbp, trainbpx* and *trainlm*. First two of them are based on gradient method. Since it requires small learning rates for stable learning, simple back-propagation with *trainbp* is very slow. *Trainbpx* applying momentum or adaptive learning rate can be considered as faster method than *trainbp* but *trainlm* applying Levenberg-Marquardt optimization is the most efficient one since it includes improvement techniques to increase the speed and reliability of simple back-propagation networks. Levenberg-Marquardt update rule is

$$\Delta W = (J^T J + \mu I)^{-1} J^T e \qquad (2)$$

where

J : Jacobian matrix of the derivatives of each error to each weight

μ : Scalar

e : error vector

If scalar μ is too large the above expression approximates gradient descent, while if it is small it reduces to Gauss-Newton method. The Gauss-Newton method is faster and accurate near an error minimum, so the aim is to shift towards to Gauss-Newton method as quickly as possible. Therefore, μ is decreased after each successful step and increased only when a step increases the error[17].

The network error is defined as the difference between the desired output or the target value and the output of neural network approximation, for a given set of inputs. During training, the weighting matrices and bias vectors are adjusted to minimize the cost function that is a function of the error of the network. If q sets of points are used to train

the network, the cost function in terms of the sum square error of the network can be written as[8]

$$E = \sum_{k=1}^{qS2} e(k)^2 = \sum_{j=1}^{q} \sum_{i=1}^{S2} (a_{desired}(i,j) - a2(i,j))^2 \qquad (3)$$

where

$a2_{true}$: the desired output

The training continues until the error goal is reached, the minimum error gradient occurs, the maximum value of μ occurs or the maximum number of epochs has met.

(ii) Radial-Basis Neural Networks

Radial-basis neural networks are two-layer networks with a hidden layer of radial-basis transfer function and a linear output layer as shown in Figure 4. Radial-basis networks may require more neurons than standard back-propagation networks, but they can be designed in a fraction of time it takes to train the standard back-propagation networks. The are efficient when there are many training vectors available. The network output equation for a single hidden radial-basis layer and a linear output layer radial-basis network can also be represented by Equation 1. The transfer function for radial-basis networks is radial-basis transfer function as shown in Figure 4. A radial-basis neuron receives as net input vector distance between its weight vector, w and input vector p, multiplied by the bias b. Figure 4 also shows how the radial-basis function has a maximum of 1 when its input is 0 and how the radial-basis transfer function can be used with a neuron having weight and bias[17]. The radial-basis function can be represented as

$$f(n,b)=radbas=\exp(-(b.n)^2) \qquad (4)$$

In Matlab, radial-basis networks can be designed by using two different design functions, *solverb* and *solverbe*. *Solverbe* designs a network with zero error on training vectors by creating as many radial-basis neurons as there are input vectors. A more efficient design

in terms of network size, is obtained from *solverb*, which utilizes an iterative procedure to minimize the number of neurons required to obtain a user specified root-mean-square error. This study investigates neural networks designed by using *solverb* to map the power index as a function of camber and aspect ratio since the networks designed with *solverb* have less neurons and less training time than *solverbe* networks. On the other hand, it must be noted that the networks designed with *solverbe* gives more accurate results since it calculates the exact values.

The radial-basis networks can be compared with the standard back-propagation networks in terms of training time and size as follows:

1. Radial-basis networks, even when designed efficiently with *solverb*, tend to have many times more neurons than a comparable back-propagation with tan-sigmoid or log-sigmoid neurons in the hidden layer. The basic reason for this is sigmoid neurons can have outputs over a large region of the input space, while radial-basis neurons only respond to relatively small regions of the input space. The larger the input space means the more radial-basis neurons required.

2. Designing a radial-basis network often takes less time than training a back-propagation network and can sometimes result in fewer neurons being used.

2. PROPERTIES OF THE AERODYNAMIC DATA SETS

(i) Two-Dimensional Airfoil

In the first part of this study, *a radial-basis neural network scheme* is designed to access the capability for processing the aerodynamic data for low Reynolds number (10^4-10^5) airfoils. The simulated data obtained from computational fluid dynamics tool, XFOIL, as reported in References[12-13]. The airfoil chosen is CLARK-Y[13] and the simulated data include lift coefficient, C_L, and drag coefficient, C_D, at various Reynolds numbers and angles of attack. Two-dimensional computations based on the coupled inviscid and thin layer flow equations are conducted by Shyy et. al[13].

The input data set is organized so that Reynolds number and angle of attack form the input vector required, p, for *MATLAB Neural Network*[17] and the C_L and C_D are used to obtain C_L/C_D that form the target vector, a.

$$p = \begin{bmatrix} Re \\ \alpha \end{bmatrix}_{765\times 2} \qquad a = \left[C_L / C_D\right]_{765\times 1} \tag{5}$$

The aim is to train the network in such a way that there is one-to-one mapping between the output vector and the target vector. In total, 1530 data as inputs and 765 data as outputs are available. In the data set, there are 14 different Reynolds number ranging between, 7.5×10^4 to 3.5×10^5, for angle of attack ranges from –3° to 20°.

(ii) Wing/Blade Model

The aerodynamic data generated from a low Reynolds number wing/blade model that is obtained by using a potential flow solver, PMARC, and a coupled inviscid-viscous flow solver, XFOIL. The lift coefficients, C_L, and drag coefficients, C_D, at various camber, y_c, aspect ratios, AR, and angles-of-attack, α, at fixed Reynolds number, $Re=2.\times10^5$, and thickness ratio, y_t=5%, are used to correlate the aerodynamic performance, measured by power index, $C_L^{3/2}/C_D$.

The results obtained from the model are used to train the network and three different training data sets are chosen as shown in Table 1. Table 2 summarizes the test data sets composed of interpolated y_c and AR values. After neural network is training by using one of the training data set of Table 1, it is going to be tested by one of the test data of Table 2 to demonstrate its generalization capabilities in terms of design parameters.

3. POLYNOMIAL BASED RESPONSE SURFACE METHOD (RSM)

The approach of RSM is to perform a series of experiments, or numerical analyses, for a prescribed set of design points, and to construct a global approximation (response surface)

of the measured quantity (the response) over the design space. For two-dimensional airfoil, the response is the lift to drag ratio, C_L / C_D, and the design space consists of Reynolds Number and angle of attack, α. Up to 4th order polynomials are tested for the response surface approximations. The response surface is fit by standard least square regression using *JMP*, statistical analysis software. A series of models of polynomials are tested for the best response surface approximations and the resultant root mean square (RMS) errors are calculated based on analysis using t-statistics[1] for each of the model. According to the results presented in Table 3, Model 11 gives the smallest RMS for two-dimensional airfoil case but RMS is still high. These results may be improved by using higher order polynomials. However, instead of applying higher order polynomial fitting to full data set, polynomial fitting is applied to a set of network output at fixed Reynolds number ($Re=2.5\times10^5$). For this case, since the network is trained at 16 neurons with the data set of $Re=2.5\times10^5$, 16th order polynomial-based response surface is fitted by using Matlab.

For the wing/blade model, the response is the flight power index, $C_L^{3/2} / C_D$, and the design space consists of a set of design variables including camber, y_c, and wing aspect ratio, AR. Quadratic, cubic and 4th order polynomials are also tested for the response surface approximations for this case. A series of models of polynomials are tested for the best response surface approximations of 9-points, 15-points and 25-points data set cases and the resultant root mean square (RMS) errors are calculated based on analysis using t-statistics[1] for each of the model. According to the results presented in Table 4, Model 4 gives the smallest RMS for 9-points and 15-points data set whereas Model 12 enables the smallest RMS for 25-points data set. Therefore, the resulting equations representing the response surface are given by,

$$C_L^{3/2}/C_D = -0.1252 + 35.58971 y_c + 2.3037 AR - 278.3526 y_c^{\,2} + 32.1442 AR y_c - 0.1364 AR^2 - 239.7351 AR y_c^2 \qquad (6)$$

(For 9-Points Case)

$$C_L^{3/2}/C_D = -0.0920 + 35.9873 y_c + 2.3062 AR - 286.3705 y_c^2 + 31.9954 AR y_c - 0.1382 AR^2 - 238.1881 AR y_c^2 \quad (7)$$

(For 15-Points Case)

$$C_L^{3/2}/C_D = -0.1842 + 144.7277 y_c + 2.3668 AR - 5518.7150 y_c^2 - 57.2096 AR y_c - 0.1463 AR^2 + 78040.0120 y_c^3 - 947.7483 AR y_c^2 - 365906.1 y_c^4 + 4574.1781 AR y_c^3 \quad (8)$$

(For 25-Points Case)

3. RESULTS AND DISCUSSIONS

(i) Comparison of radial-basis design *Solverb* and *Solverbe* for Two-Dimensional Airfoil Case

Training of a network requires repeated cycle through the data, each time adjusting the values of the weights and biases to improve performance. Each pass through the training data is called an epoch and the neural network learns through the overall change in weights accumulating over many epochs. Training continues until the error target is met or until the maximum number of neurons is exceeded. Testing is performed after training usually with less data than those used in training stage. In this study, to use *solverb* as a design function, a range of maximum number of neurons from 50 to 765 and a range of spread constants from 1.5 to 6 are used. It is found that *solverb* cannot satisfy an error goal less than 0.1. Figure 5 shows an example of such network error behavior, i.e., sum-squared network error versus epoch number for C_L/C_D of a non-converging training exercise.

Although the root-mean-square network error drops until 200th epoch, after that the training input matrix became *rank deficient* and therefore the error begins to increase since the network receives different information from almost same points. Noting that other than the spread constant, the error goal is the only design parameter for radial-basis networks designed with *solverb*, one can conclude that the solution with *solverb* cannot

converge with the data set of 765, presumably because of the size and the characteristics of the data. We will visit this issue with a reduced data in the next section.

In contrast to *solverb*, *solverbe* is capable of training the network with the same data set, as shown in the Figure 6. Figure 6 shows the comparison of the target values of C_L/C_D and the trained C_L/C_D and these two sets are in excellent agreement. Figure 6 also shows the errors or the differences between the target values of C_L/C_D and the trained C_L/C_D and the error $O(10^{-10})$. It appears that *solverbe* is far more efficient in handling either the full or the partial data set.

It is important to make use of *solverb* since a critical goal during training is to find a network that is large enough to learn the task but small enough to generalize. With *solverbe*, the network creates as many neurons as the input data. As already mentioned *solverb* uses fewer network to process the data. In order to train the network by using *solverb* as a design function, the data set is reduced in such a way that there is a uniform angle of attack grid distribution with $\Delta\alpha\cong0.5°$. Finally the input set is reduced to 510 data from 1530 data and with this reduced data set *solverb* was successful in training the network. Figure 7 shows an example of the network error behavior of a converging run. From this figure, the network is able to satisfactorily handle the data with 253 neurons. For this case, the results of the network run with *solverb* are shown in Figure 8. The target and network calculated values of C_L/C_D, are almost mapped one-to-one. Figure 8 also shows the errors of C_L/C_D and the order of the max error is $O(10^{-5})$. The reduced data set is also trained with *solverbe* and it appears that both *solverb* and *solverbe* can perform satisfactorily with the reduced data set.

After training is completed, the network is to be used for generalization to determine whether it can process the data correctly to the patterns that are only qualitatively similar to the original training patterns. Generalization is useful because the real world data is noisy, distorted and often incomplete.

Previously presented test sets contain the data chosen from the training data set to check the neural network scheme developed. In order to generalize the scheme a new test data

set is generated in such a way that the data set does not include any data from the training set. The test matrix and several data sets used within this study are summarized in Table 5.

Finally, the relative errors for Test set#3 with *solverbe* and *solverb* are compared in Figure 9 for C_L/C_D. For this case, maximum error is 2 with *solverbe* and 1 for *solverb*. If these values are compared with target values of the corresponding data, the maximum percentage error is calculated as 3% with *solverbe* and 1% for *solverb*. It should be noted that for this case, maximum error of *solverb* is less than that of *solverbe*.

(ii) Comparison of Neural Network and Polynomial-Based Response Surface Methods

For wing model, the outputs of the *solverb* neural network, along with the results of RSM, are compared to demonstrate the efficiency and the merits of the neural network approach in data analysis. Figure 11(a) illustrates the comparison between RSM and neural network output results based on 9 points training data. For this case, both methods predicted the original 9 points accurately but both failed to predict accurately the interpolation points at yc=0.025 and 0.075. Figure 11(b) shows that adding 6 new points at AR=2 and 4 at y_c=0., 0.05 and 0.1 (15-points training data set) does not significantly improve the 6 interpolated values. However, with the addition of 10 new points at yc=0.025 and 075 at AR=1,2,3,4 and 5, (25-points training data set) both neural network and polynomial RSM accurately capture the overall behavior of the aerodynamic data as shown in Figure 11(c). The generalization of the neural network with 25 data points is further assessed by comparing additional interpolated values at different y_c and AR at yc=0.0125 and 0875 at AR=1,2,3,4 and 5. The error norms of both methods for different number of data simulations are plotted in Figure 12. These comparisons illustrate that both neural and conventional polynomial fitting methods are doing a good job as the number of points is increased.

For two-dimensional airfoil case, the network outputs obtained by *solverb* are compared at fixed Reynolds number ($Re=2.5\times10^5$). Figure 10 shows this comparison including the target values. This figure illustrates that network results are closer to target values than the case for the polynomial fitting results.

(iii) Comparison of Radial-Basis Neural Network and Back-propagation Networks

In order to be able to make comparisons between the performance of radial-basis and back-propagation networks, the training time histories of these networks are summarized in Table 6 and Table 7. These tables shows that both are efficient in training of 9-points, 15 points and 15-points training data sets in terms of training time since the required number epochs or iterations is not high. As far as accuracy concerned, applying radial-basis networks is more advantages for interpolations in y_c as shown in Figure 13 whereas applying back-propagation networks gives better results for interpolations in AR as shown in Figure 14. However, both of the networks perform well as the number of points increases in training data.

4. SUMMARY AND CONCLUSIONS

In this study, we applied the polynomial and the neural network based response surface techniques by considering the low Reynolds number aerodynamic data. We used Matlab to construct neural network based response surface and JMP for polynomial based response surface. In the first part of the study, we have considered aerodynamic data gathered for low Reynolds Number airfoils. For this case, we designed a radial-basis network using two different design functions: *solverb* and *solverbe*. With the beginning data set provided by Shyy et. al[13], it is possible to design a neural network for low Reynolds number flows with *solverbe* with almost zero error on training vectors, whereas it is not possible to train a network using *solverb*. This difficulty was avoided by reducing the density of the input data set to prevent the network from receiving different information from closely spaced data. The reduced data set includes 510 data from 255 simulation instead of 1530 data set from 765 simulation and network is designed and trained by using both *solverb* and *solverbe*. The outputs of the network are compared with the target values of C_L/C_D for training purposes.

After the training stage of the network was completed, new data sets obtained from XFOIL were used for prediction purposes. The new test data set did not include any data

from training data set. The networks provided results with a maximum error of 3% for C_L/C_D. This means the network can generalize with a degree of fault tolerance. The network results are compared with the polynomial based response surface results.

In the second part of the study, we have considered aerodynamic data gathered for low Reynolds Number wing model. For this case, we applied radial-basis neural networks and compared the outputs of the radial-basis network with the polynomial response surface results. The comparisons demonstrated that neural network based response surfaces capture the behavior of the aerodynamic data obtained from computational simulations more accurately. For this case, we also investigated the relative features radial-basis neural network and back-propagation networks. It is observed from the results that both networks can correlate the aerodynamic performance as a function of design parameters precisely if sufficient number of data is used to train the network. These results encourage the further applications of the neural network techniques for wing and airfoil design optimization.

6. REFERENCES

[1]Myers, R. H., and Montgomery, D. C. (1995). *Response Surface Methodology – Process and Product Optimization Using Designed Experiments*, New York: John Wiley & Sons, Inc.

[2]Hammerstrom, D., "Neural Networks at Work", *IEEE Spectrum*, pp. 26-32, June 1993.

[3]Hammerstrom, D., "Working with Neural Networks", *IEEE Spectrum*, pp.46-53, July 1993.

[4]Norgaard, M., Jorgenson, C. C., and Ross, J. C., "Neural Network Prediction of New Aircraft Design Coefficients", NASA TM-112197, 1997.

[5]Ross, J. C., Jorgenson, C. C., and Norgaard, M., "Reducing Wind Tunnel Data Requirements Using Neural Networks", NASA TM-112193, 1997.

[6]Protzel, P. W., Palumbo, D. L., and Arras, M. K., "Fault Tolerance of Artificial Neural Networks with Applications in Critical Systems", NASA Technical Paper 3187.

[7]Sparks Jr., D.W., Maghami, P. G., "Neural Networks for Rapid Design and Analysis", AIAA-98-1779.

[8]Maghami, P. G., Sparks Jr., D.W., "Design of Neural Networks for fast Convergence and Accuracy", AIAA-98-1780.

[9]Rai, M.M. and Madavan, N.K., "Aerodynamic Design Using Neural Networks", AIAA Paper No. 98-4928.

[10]Carpenter, W.C. and Barthelemy, J.-F.M., "A comparison of Polynomial Approximations and Artificial Neural Nets as Response Surface", *Structural Optimization 5*, pp.166-174, Springer-Veriag, 1993

[11]Faller, W.E. and Schereck, S. J., "Unsteady Fluid Mechanics Applications of Neural Networks" AIAA 95-0529.

[12]Drela, M., "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils" , Lecture Notes in Engineering, Vol.54, pp.1-12, Springer-Verlag, New York, 1989.

[13]Shyy, W., Klevebring, F., Nilsson, M., Sloan, J., Carrol, B.F. and Fuentes, C., A study of Rigid and Flexible Low Reynolds Number Airfoils, Dept. of Aerospace, University of Florida, 1998.

[14]Ashby, D.L., Dudley, M.R., Iguchi, S.K., Browne, L., and Katz, J., "Potential Flow Theory and Operation Guide for Panel Code PMARC_12"

[15]Fan, X., Herbert, T., and Haritonidis, J.H., "Transition Control with Neural Networks", AIAA-95-0674.

[16]Papila, N, Fitz-Coy N., and Shyy Wei, "Neural Network-Based Techniques For Computational Data Analysis", University of Florida Technical Report (unpublished), AeMES-TR-98-3-01, 1998.

[17]Demuth, H. and Beale, M., *Matlab Neural Network Toolbox*, The Math Works Inc, 1992.

[18]Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach To Machine Intelligence*, Prentice Hall, 1992.

[19]Jang, J.-S. R., Sun, C.-T. and Mizutani, E., *Neuro-Fuzzy And Soft Computing: A Computational Approach To Learning And Machine Intelligence*, Prentice Hall,1997.

[20]Anderson, A.J., *Handbook Of Neural Computation*, IOP Publishing and Oxford University Press, 1997.

[21]Hertz, J., Krogh, A. and Palmer, R. G., *Introduction to the Theory of Neural Computation*, A Lecture Notes Volume in the Santa Fe Institute Studies in the Sciences of Complexity, 1991.

[22] Greenman, R. M., "Two-Dimensional High-Lift Aerodynamic Optimization Using Neural Network", NASA/TM-1998-112233.

7. LIST OF TABLES

8. LIST OF FIGURES

Table 1. Training Data Sets for wing model

9-Points Training Data			15-Points Training Data			25-Points Training Data		
AR	y_c	$C_L^{3/2}/C_D$	AR	y_c	$C_L^{3/2}/C_D$	AR	y_c	$C_L^{3/2}/C_D$
1	0	2.001123	1	0	2.001123	1	0.0	2.001123
1	0.05	4.12244	1	0.05	4.12244	1	0.025	4
1	0.1	3.686585	1	0.1	3.686585	1	0.05	4.12244
3	0	5.639781	2	0	4.03	1	0.075	3.99
3	.05	9.68733	2	.05	7.12	1	0.1	3.686585
3	0.1	8.680556	2	0.1	6.34	2	0.0	4.03
5	0	7.941356	3	0	5.639781	2	0.025	7.07
5	0.05	14.09417	3	0.05	9.68733	2	0.05	7.12
5	0.1	12.89511	3	0.1	8.680556	2	0.075	6.89
			4	0	6.92	2	0.1	6.34
			4	.05	11.99	3	0.0	5.639781
			4	0.1	10.87	3	0.025	9.64
			5	0	7.941356	3	0.05	9.68733
			5	0.05	14.09417	3	0.075	9.39
			5	0.1	12.89511	3	0.1	8.680556
						4	0.0	6.92
						4	0.025	11.86
						4	0.05	11.99
						4	0.075	11.66
						4	0.1	10.87
						5	0.0	7.941356
						5	0.025	13.83
						5	0.05	14.09417
						5	0.075	13.73
						5	0.1	12.89511

Table 2. Test Data Sets for wing model based on AR and y_c

Test Set#1 for y_c		Test Set#2 for y_c		Test Set#3 for y_c		Test Set#1 for AR		Test Set#2 for AR	
AR	y_c	AR	y_c	AR	y_c	AR	y_c	AR	y_c
1	0.025	1	0.025	1	0.0125	2	0	2	0
1	0.075	1	0.075	1	0.0875	2	0.05	2	0.025
3	0.025	2	0.025	2	0.0125	2	0.1	2	0.05
3	0.075	2	0.075	2	0.0875	4	0	2	0.075
5	0.025	3	0.025	3	0.0125	4	0.05	2	0.1
5	0.075	3	0.075	3	0.0875	4	0.1	4	0
		4	0.025	4	0.0125			4	0.025
		4	0.075	4	0.0875			4	0.05
		5	0.025	5	0.0125			4	0.075
		5	0.075	5	0.0875			4	0.1

Table 3. Root Mean Square (RMS) for airfoil case

Model No	MODEL	RMS
1	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6$	3.5474
2	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3$	3.3554
3	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7Re^3$	3.4186
4	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7ARy_C^2+c_8Re^3$	3.1644
5	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7Re^3+c_8\alpha Re^2+c_9Re\alpha^2$	2.9551
6	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2$	2.544
7	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2+c_{10}Re^3$	2.4737
8	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2+c_{10}Re^3+c_{11}\alpha^4$	2.4126
9	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2+c_{10}Re^3+c_{11}Re^4$	2.4606
10	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2+c_{10}Re^3+c_{11}\alpha^4+c_{12}Re^4$	2.3988
11	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2+c_{10}Re^3+c_{11}\alpha^4+c_{12}Re^4+c_{13}Re^2\alpha^2+c_{14}Re^3\alpha$	2.2600
12	$c_1\alpha^2+c_2\alpha+c_3\alpha Re+c_4Re+c_5Re^2+c_6+c_7\alpha^3+c_8\alpha Re^2+c_9Re\alpha^2+c_{10}Re^3+c_{11}\alpha^4+c_{12}Re^4+c_{13}Re^2\alpha^2+c_{14}Re^3\alpha+c_{15}Re\alpha^3$	2.2647

Table 4. Root Mean Square (RMS) for wing model: 9-points, 15-points and 25-points data sets

Model No	MODEL	RMS 9 data	RMS 15 data	RMS 25 data
1	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6$	0.8047	0.5172	0.7800
2	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7AR^3$	-	0.5475	0.8007
3	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7y_C^3$	-	-	0.5524
4	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7ARy_C^2$	0.1162	0.0738	-
5	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7y_C^3+c_8ARy_C^2$	-	-	0.3207
6	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7y_C^3+c_8ARy_C^2+c_9y_CAR^2$	-	-	0.3262
7	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7AR^3+c_8ARy_C^2+c_9AR^3$	-	-	0.6961
8	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7y_C^3+c_8ARy_C^2+c_9y_CAR^2+c_{10}AR^3$	-	-	0.3350
9	$c_1AR+c_2ARy_C+c_3y_C+c_4y_C^2+c_5+c_6y_C^3+c_7ARy_C^2$	-	-	0.4248
10	$c_1AR+c_2ARy_C+c_3y_C+c_4y_C^2+c_5+c_6AR^2y_C$	-	-	0.8044
11	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7y_C^3+c_8ARy_C^2+c_9y_C^4$	-	-	0.2383
12	$c_1AR^2+c_2AR+c_3ARy_C+c_4y_C+c_5y_C^2+c_6+c_7y_C^3+c_8ARy_C^2+c_9y_C^4+c_{10}ARy_C^3$	-	-	0.1073

Table 5. Summary of the data analyzed for two-dimensional airfoil

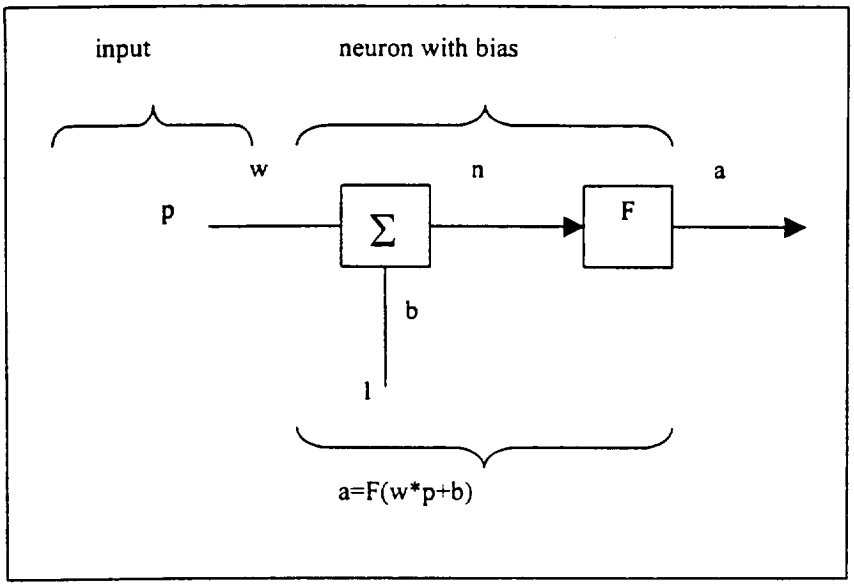
Trained Data Set	**Test Data Set**	**Total Number of Data in Trained Data Set**	**Total Number of Data in Test Data Set**
Full data set	Test #1: data chosen from full data set	1530	322
Reduced Data Set	Test #2: data chosen from reduced data set	510	182
Reduced Data Set	Test #3: data chosen from full data set	510	210

Table 6. Training History of Radial Basis Networks with *Solverb*

Neural Network No.	**No. of data**	**No. of Neurons**	**No. of Epochs**	**Steady State Error**	**Spread Constant**
1	9	7	7	10^{-4}	1.175
2	15	13	13	10^{-4}	3.25
3	25	25	23	10^{-4}	1.0

Table 7. Training History of Backpropagation Networks with *Trainlm*

Neural Network No.	**No. of data**	**No. of Neurons**	**No. of Epochs**	**Steady State Error**
1	9	20	11	3.69×10^{-5}
2	15	20	10	4.5×10^{-5}
3	25	20	97	8.37×10^{-4}



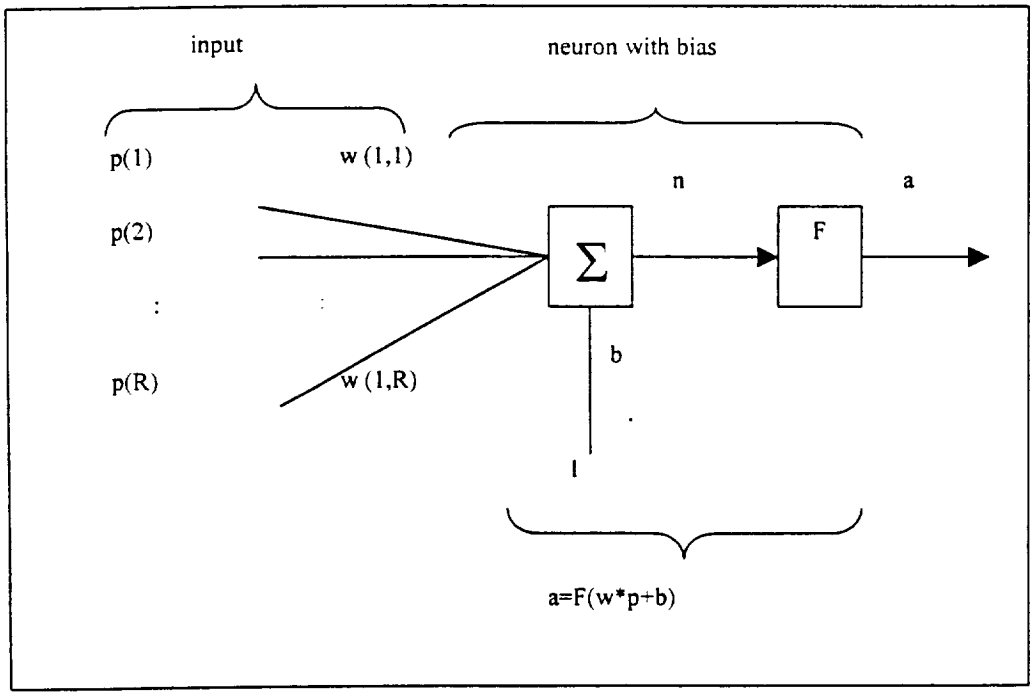
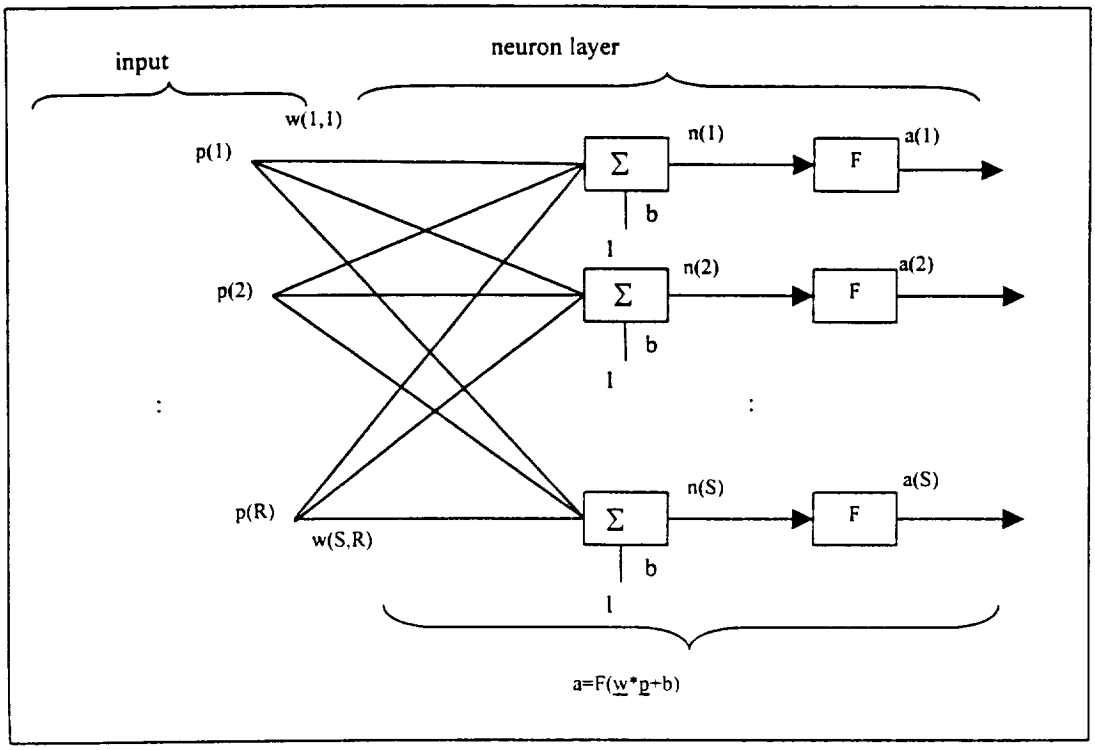


Figure 1. Illustrations of single and multiple input neuron



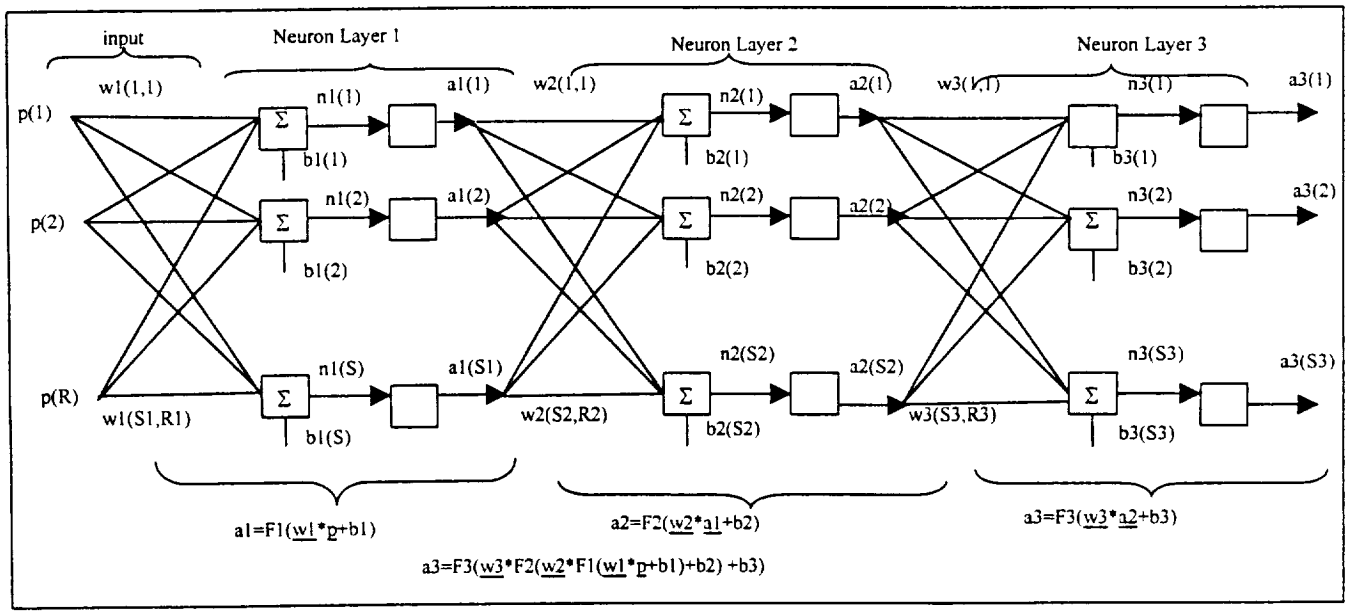


Figure 2. Illustrations of single and multiple network layers of neurons

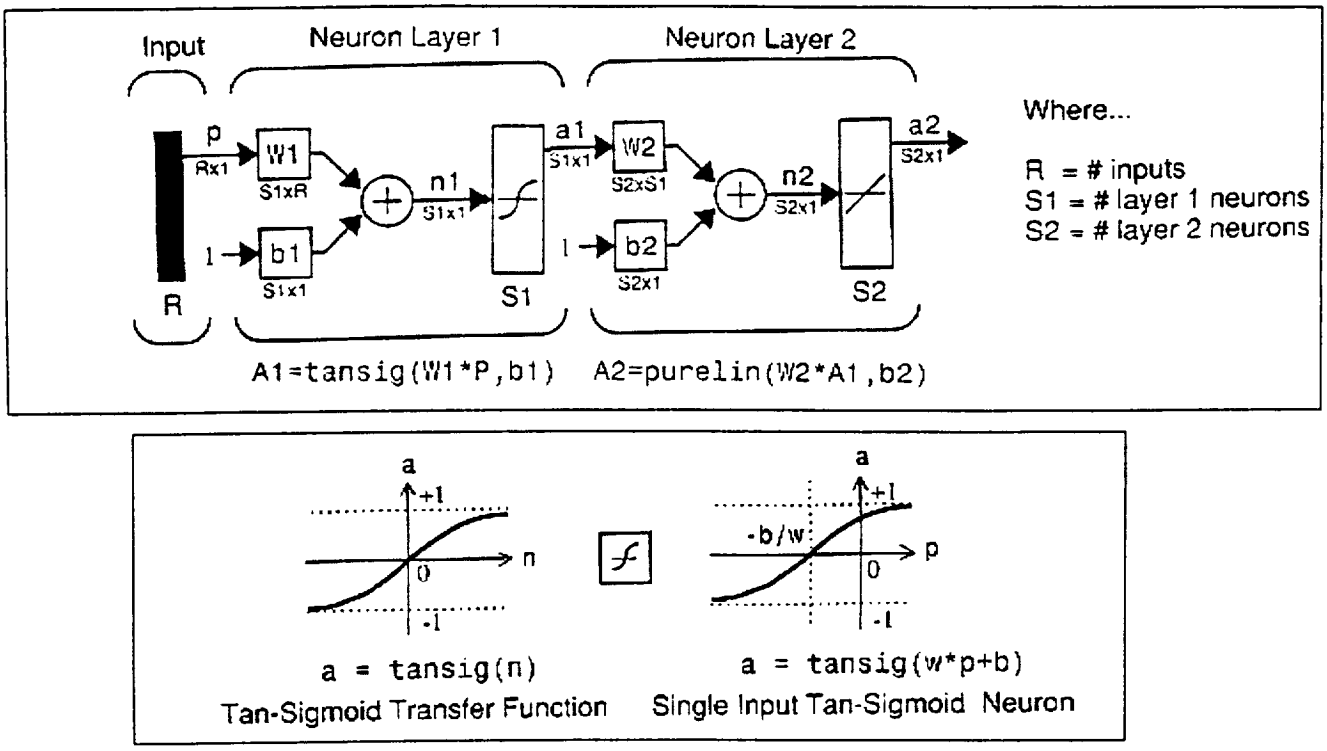


Figure 3. Backpropagation Neural Network Architecture and Tan-sigmoid Transfer function for Backpropagation Neural Networks

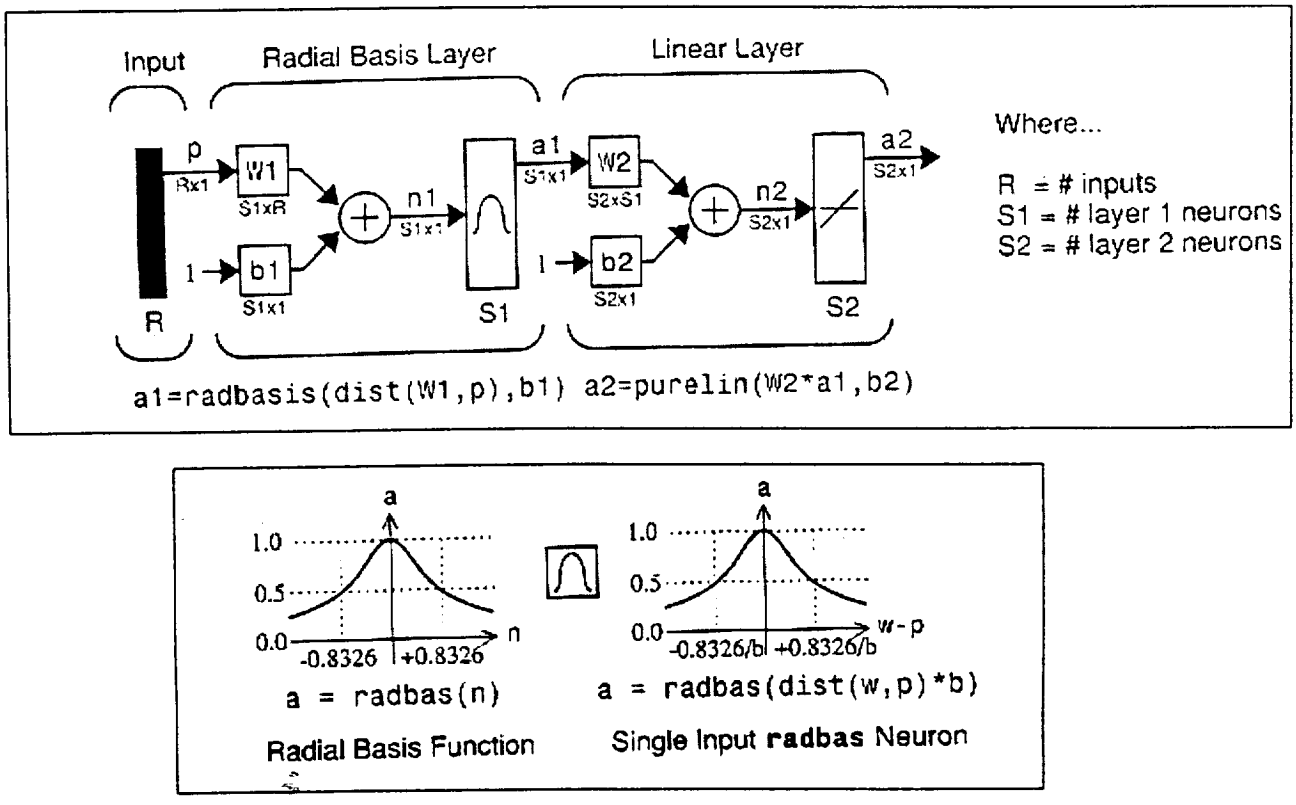


Figure 4. Radial Basis Neural Network Architecture and Radial Basis Transfer function for Radial Basis Neural Networks

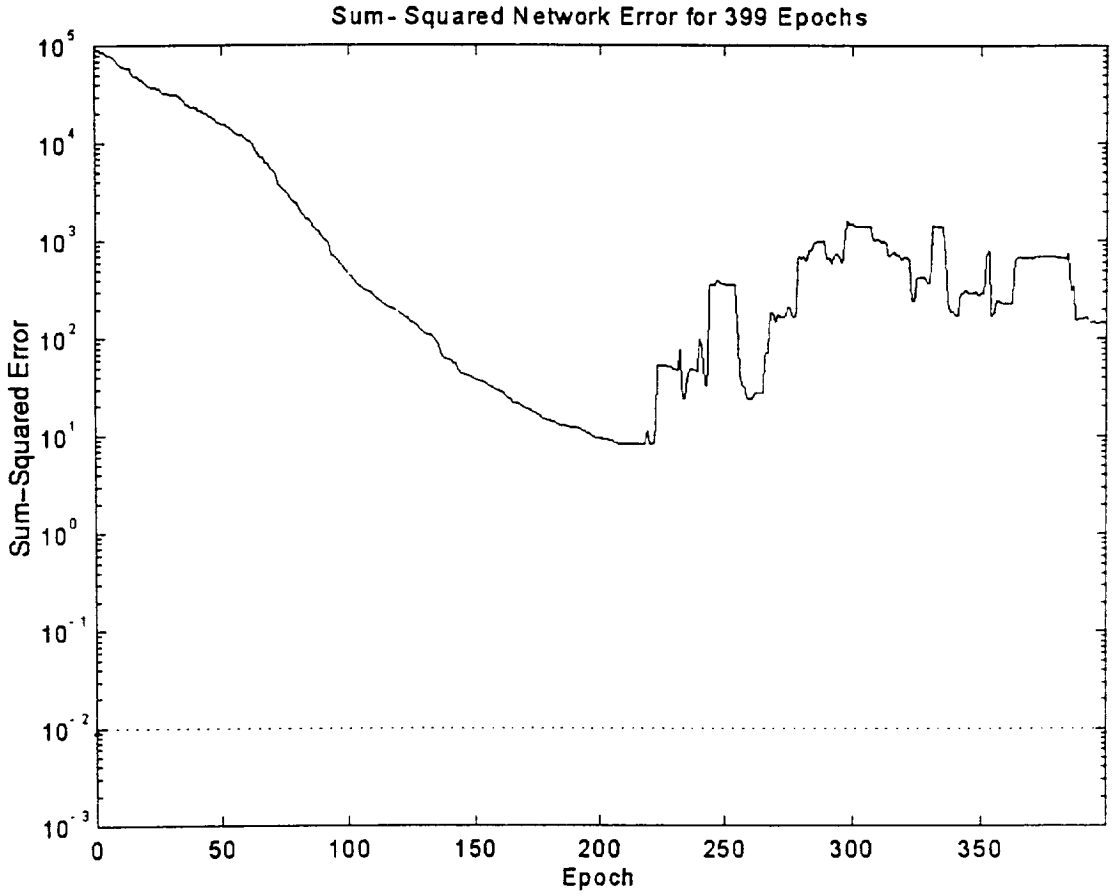


Figure 5. Non-converged root-mean-square network error behavior

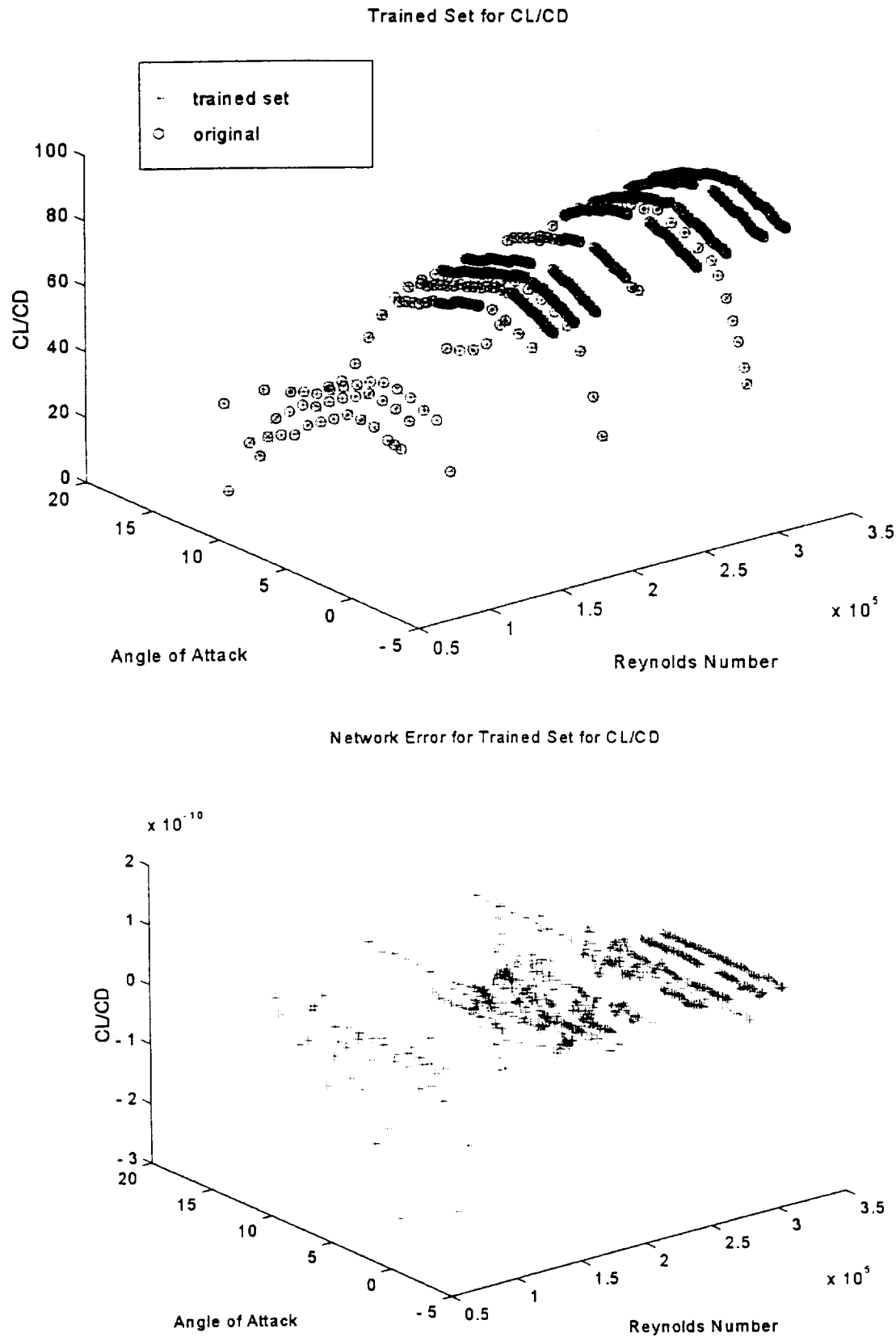


Figure 6. Comparison of the Results and Absolute Errors for C_L/C_D with *solverbe* for full data set

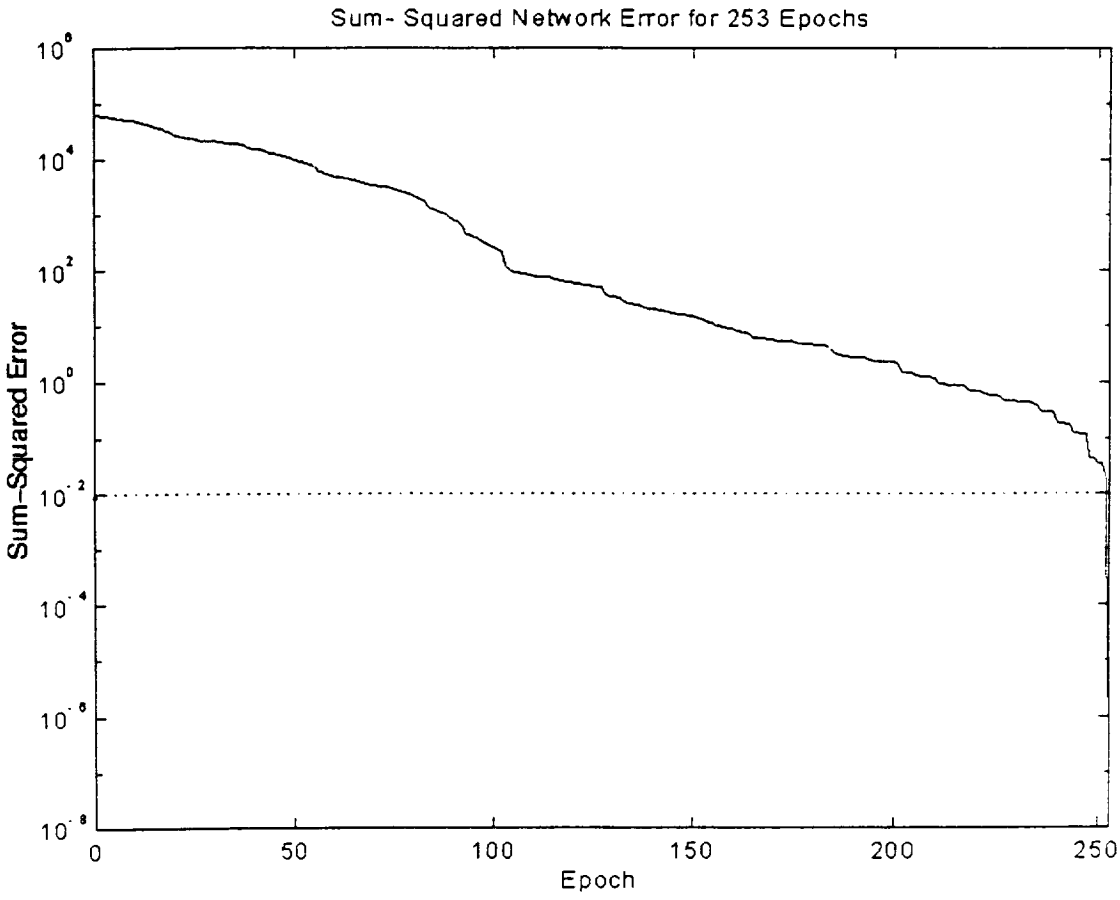


Figure 7. Converging sum-squared network error behavior

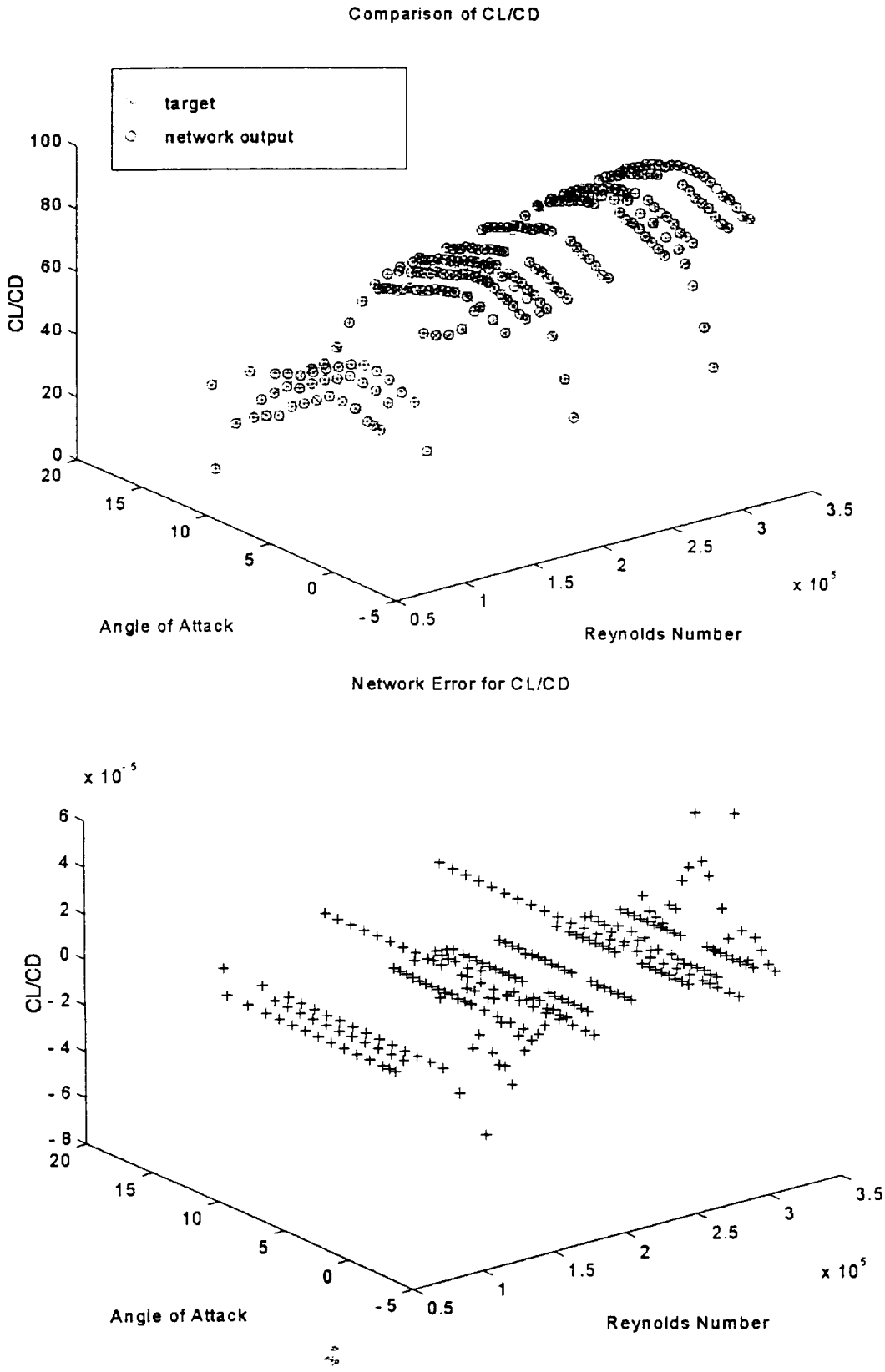


Figure 8. Comparison of the Results and Errors for C_L/C_D with *solverb* for 510 data

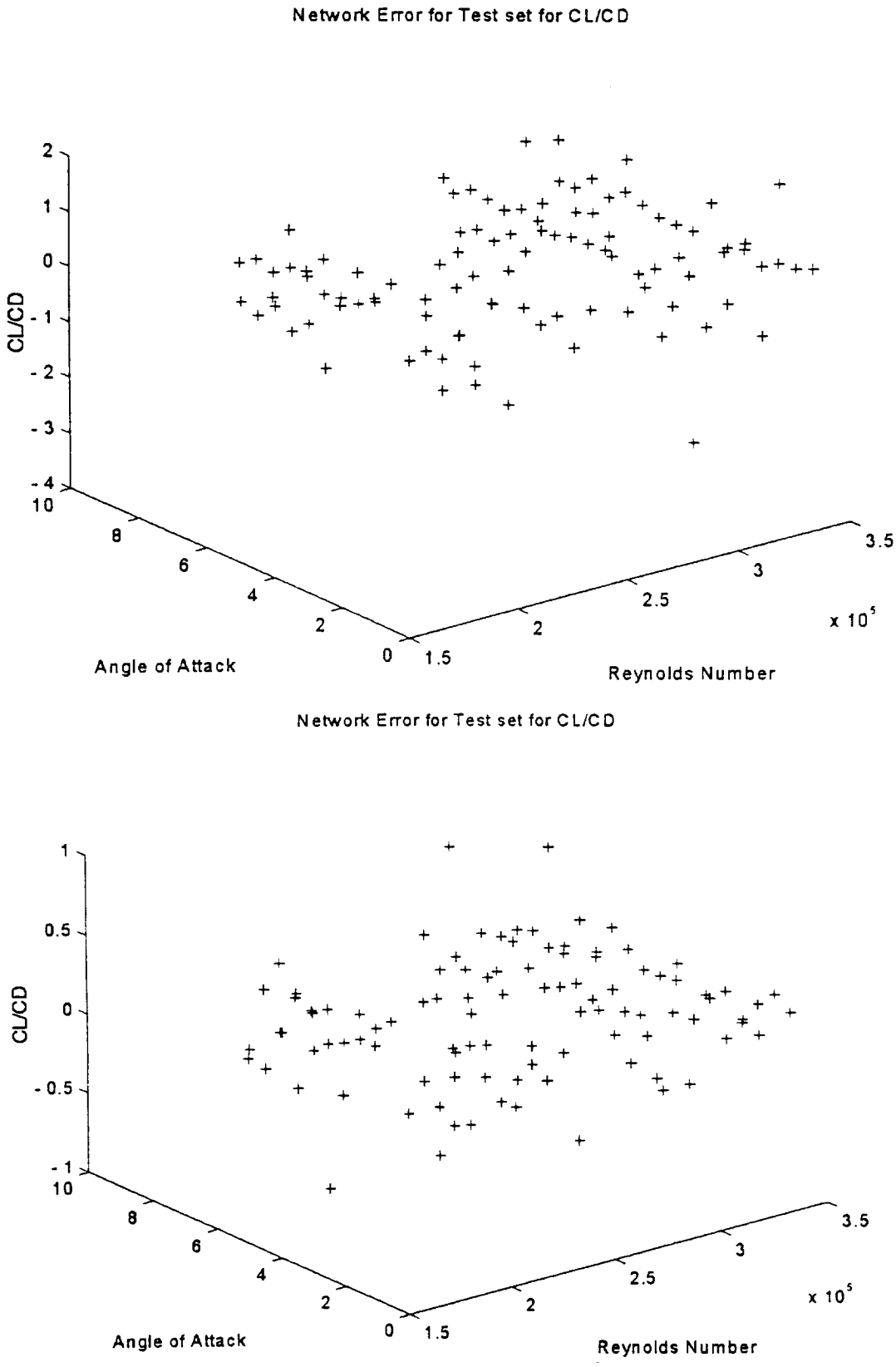


Figure 9. Error Comparison for test set #3 for C_L/C_D with *solverbe* and *solverb*

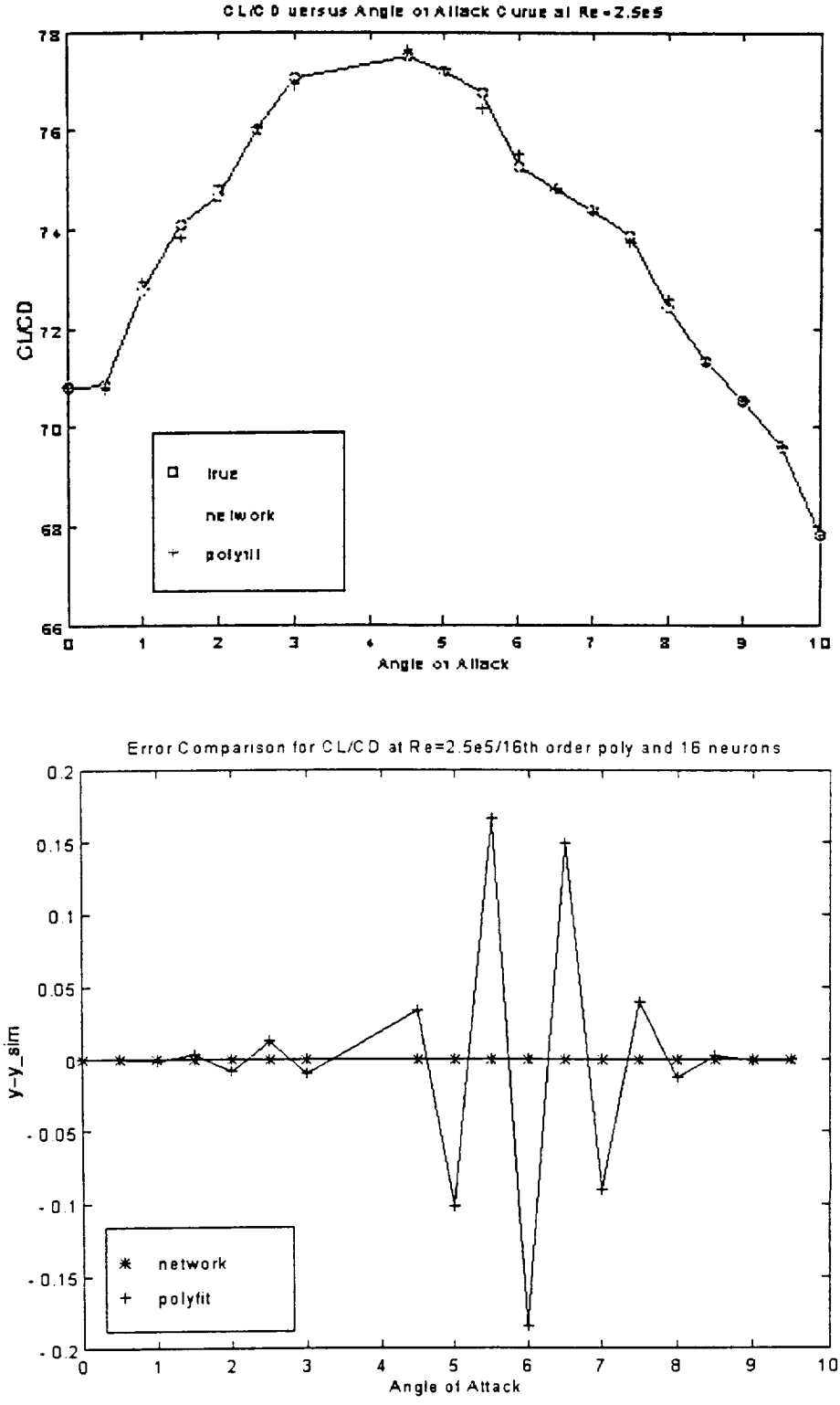
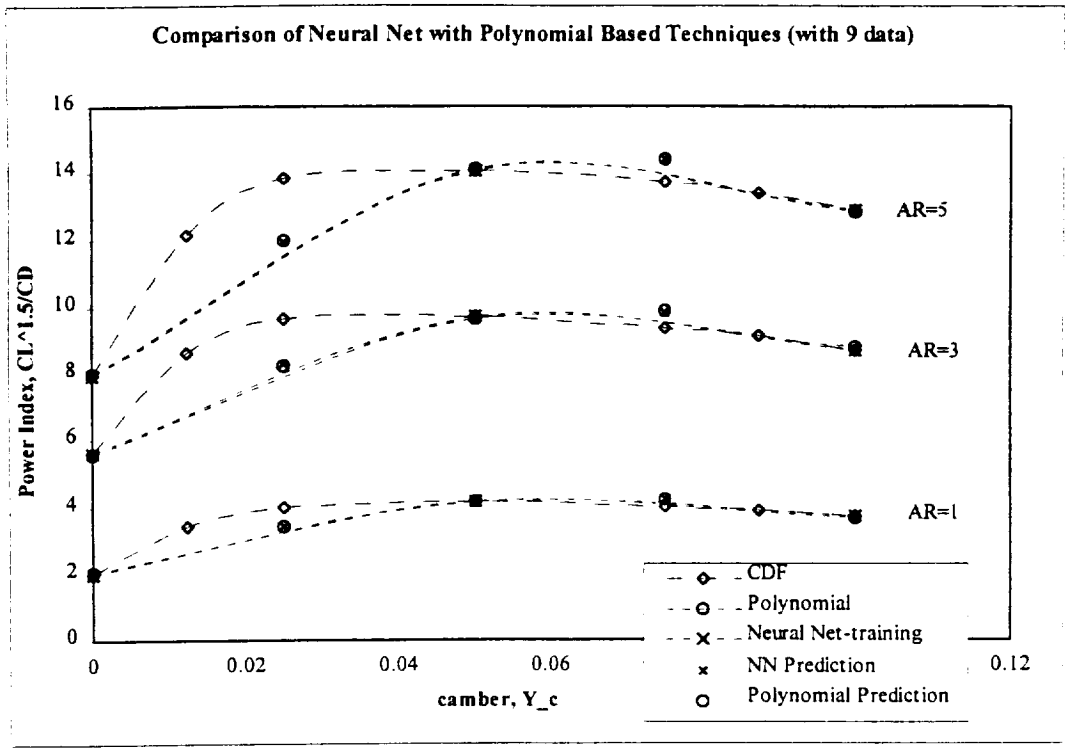
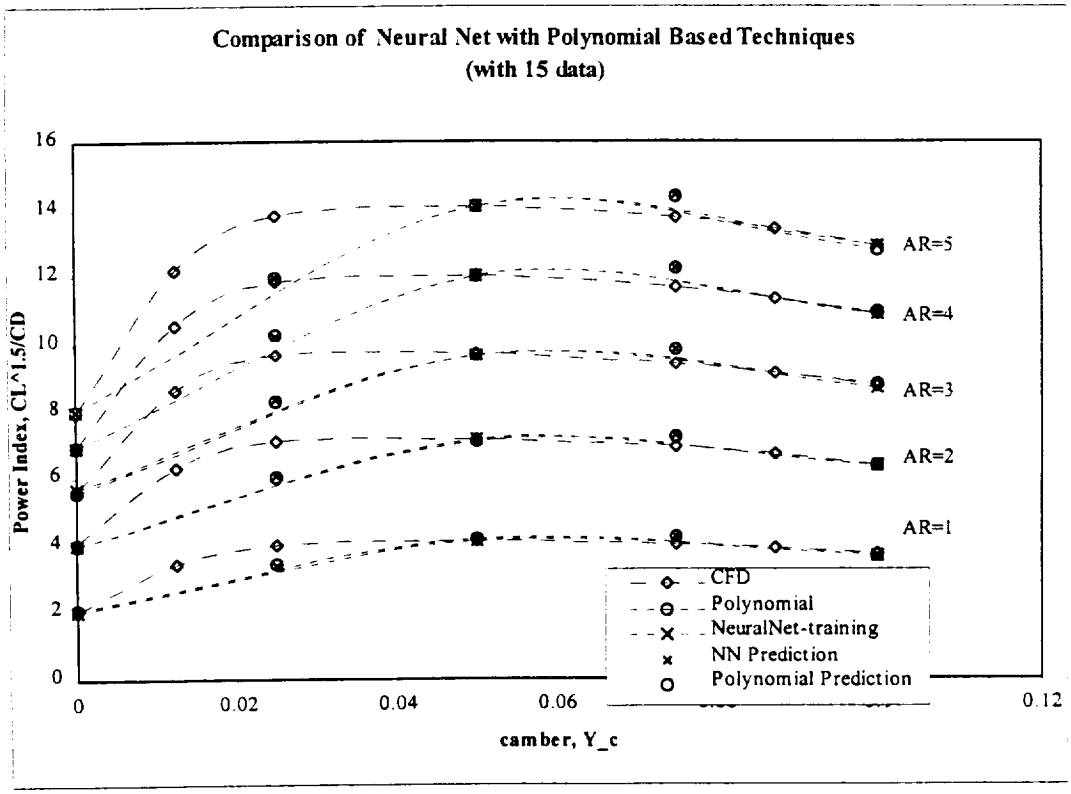


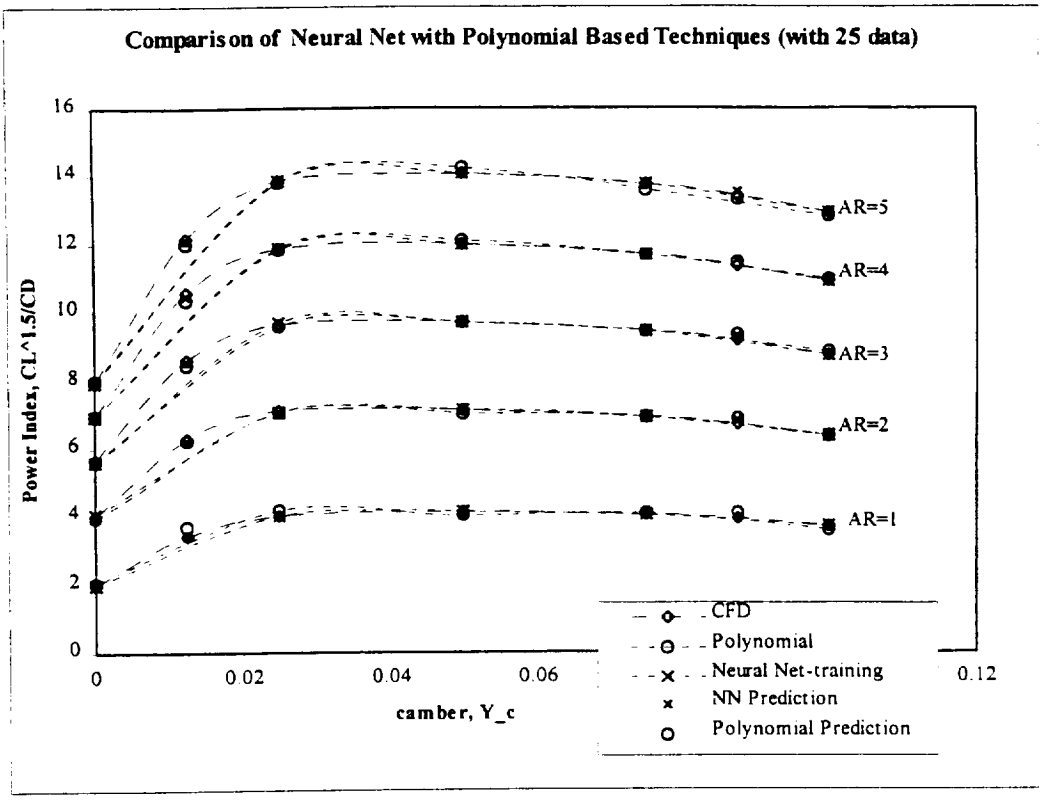
Figure 10. Comparison of Results and Errors of the Network Outputs with 2D Polynomial Fitting of 16^{th} order at Re=$2.5x10^5$


Comparison of Neural Net with Polynomial Based Techniques (with 9 data)
Power Index, CL^1.5/CD
camber, Y_c
AR=5
AR=3
AR=1
CDF
Polynomial
Neural Net-training
NN Prediction
Polynomial Prediction

(a)


Comparison of Neural Net with Polynomial Based Techniques
(with 15 data)
Power Index, CL^1.5/CD
camber, Y_c
AR=5
AR=4
AR=3
AR=2
AR=1
CFD
Polynomial
NeuralNet-training
NN Prediction
Polynomial Prediction

(b)



(c)

Figure 11. Comparison of Radial Basis Neural Network Results with Polynomials for 9- Points Training: Neural Network #1 (a), for 15- Points Training: Neural Network #2 (b), and for 25-Points training: Neural Network #3 (c)

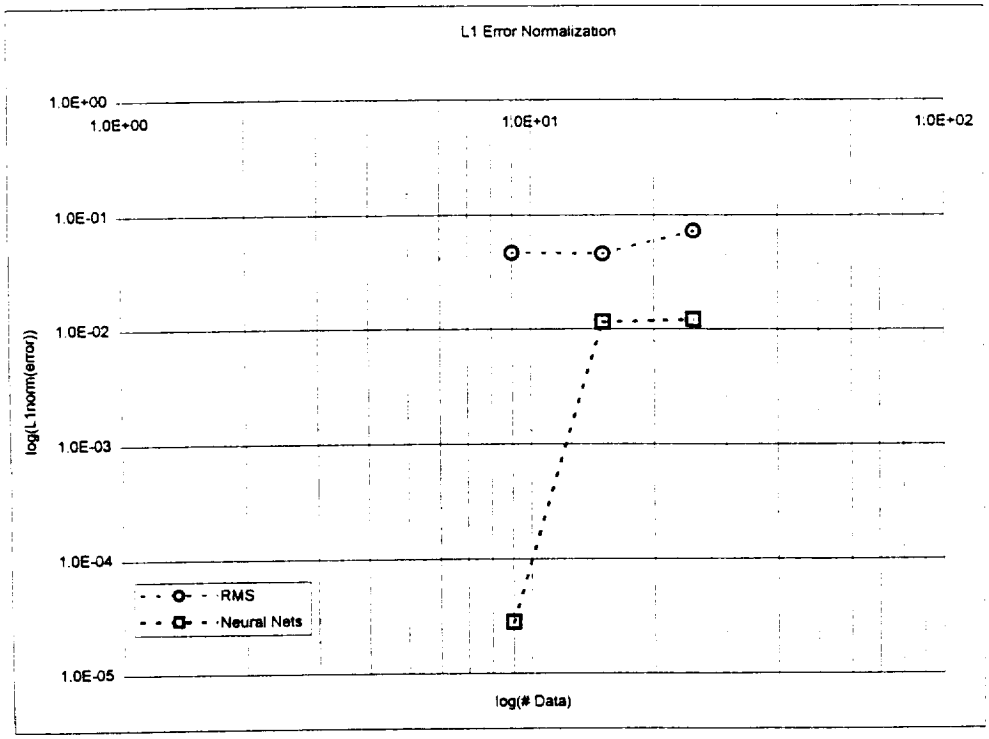
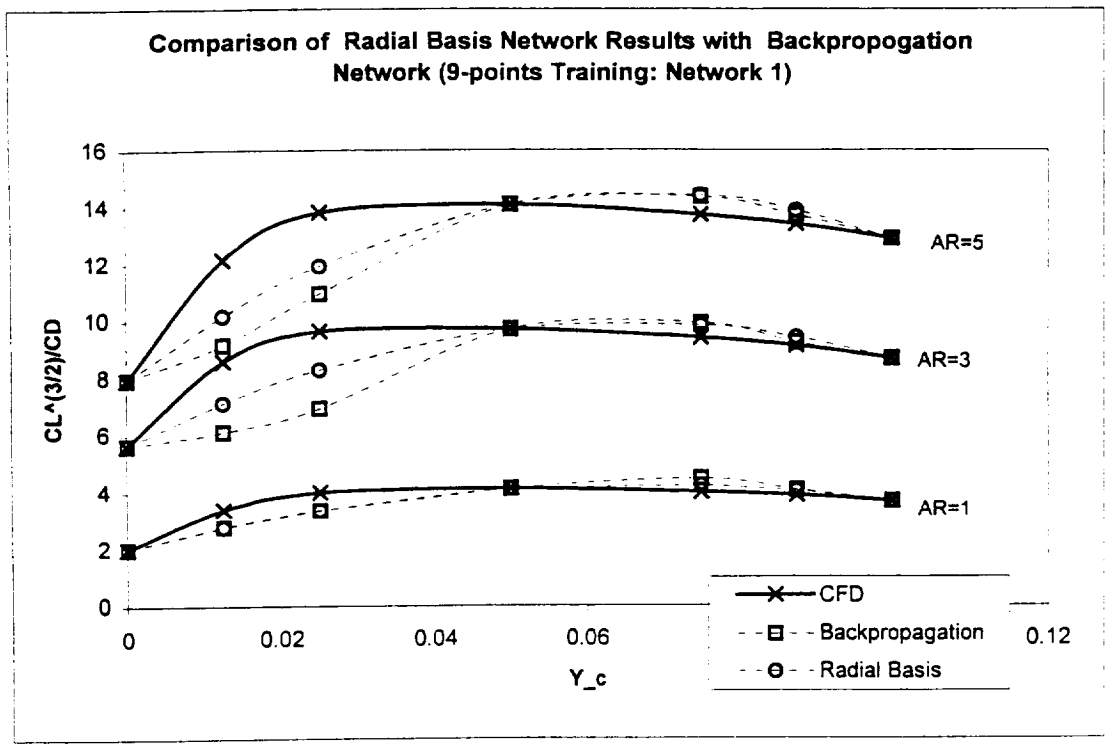
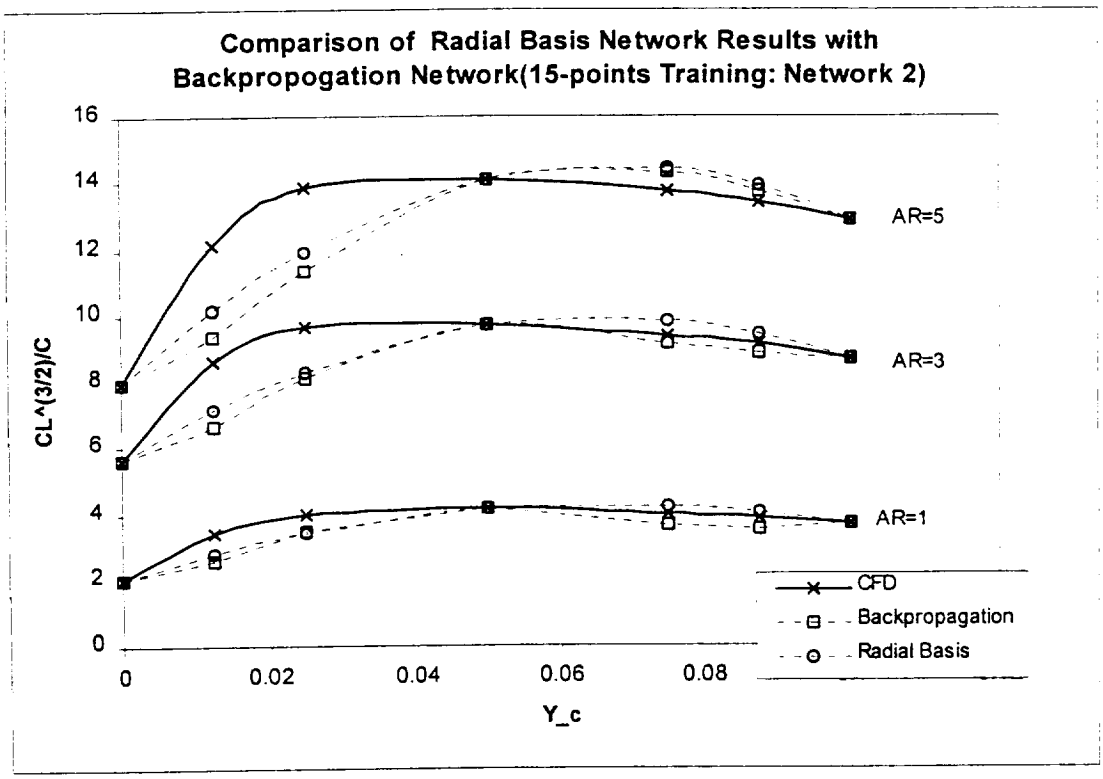


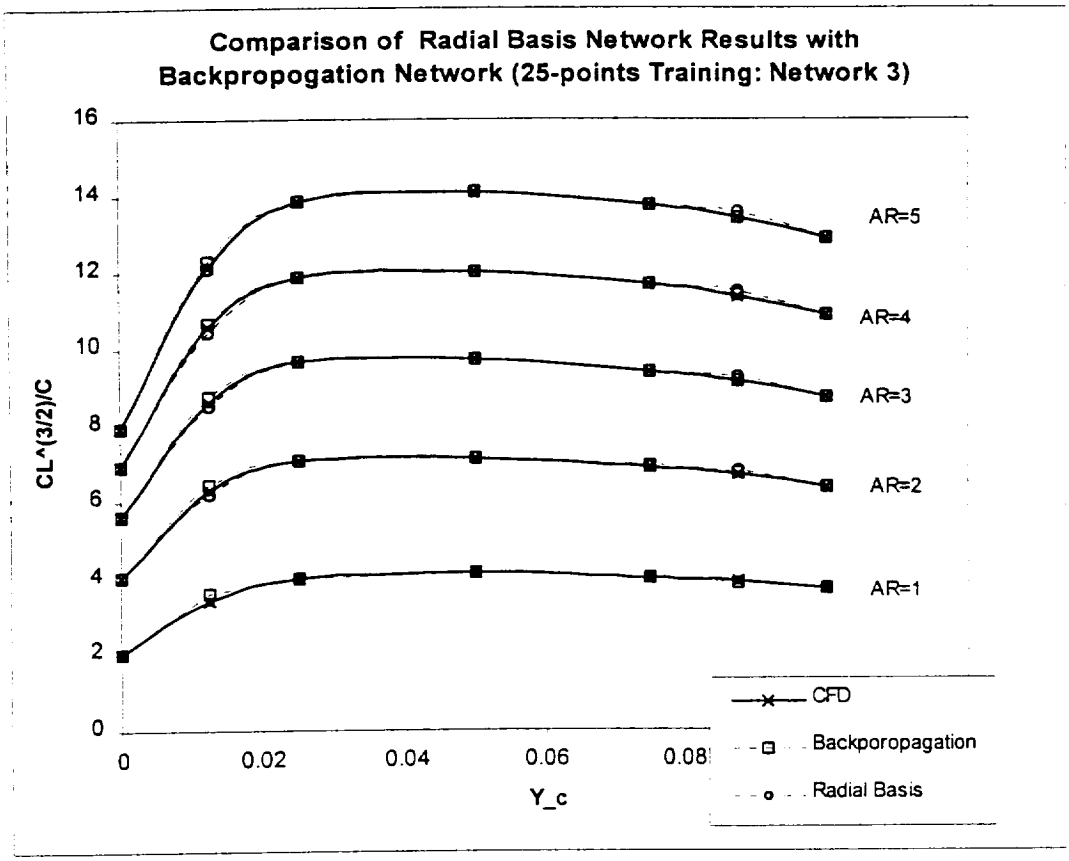
Figure 12. Comparison of Error Norms of Radial Basis Neural Network Results with Polynomials For Different Training Data Sets


Comparison of Radial Basis Network Results with Backpropogation Network (9-points Training: Network 1)
CL^(3/2)/CD
16
14
12
10
8
6
4
2
0
0
0.02
0.04
0.06
0.12
Y_c
AR=5
AR=3
AR=1
CFD
Backpropagation
Radial Basis

(a)

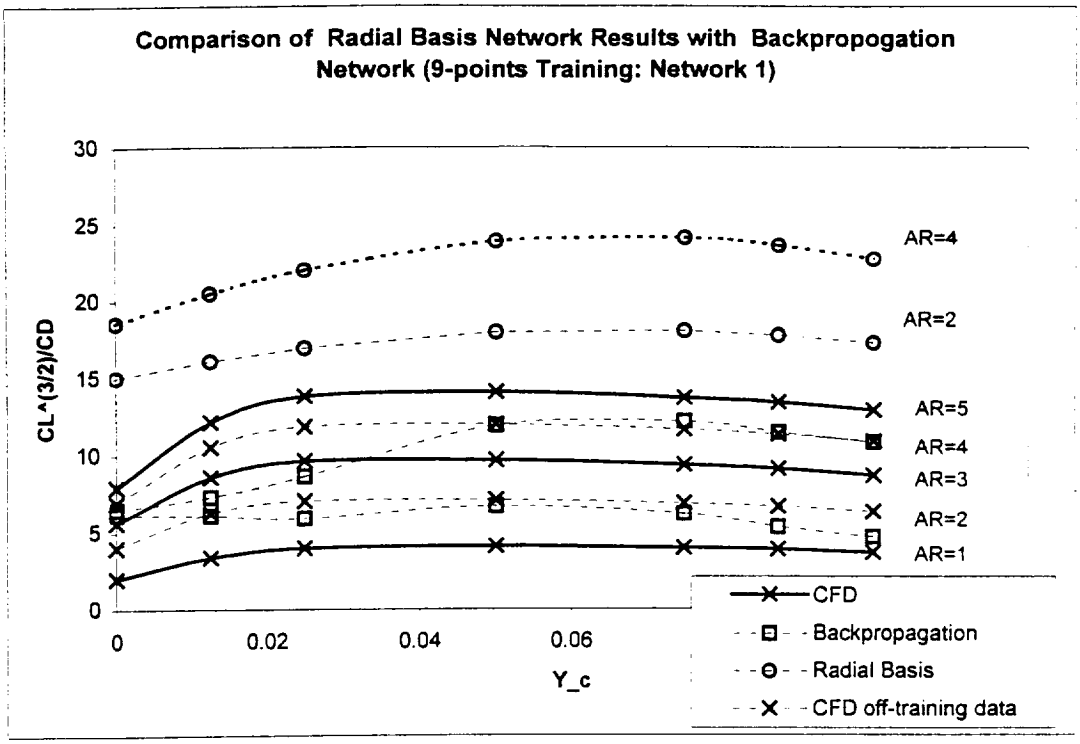

Comparison of Radial Basis Network Results with Backpropogation Network(15-points Training: Network 2)
CL^(3/2)/C
16
14
12
10
8
6
4
2
0
0
0.02
0.04
0.06
0.08
Y_c
AR=5
AR=3
AR=1
CFD
Backpropagation
Radial Basis

(b)

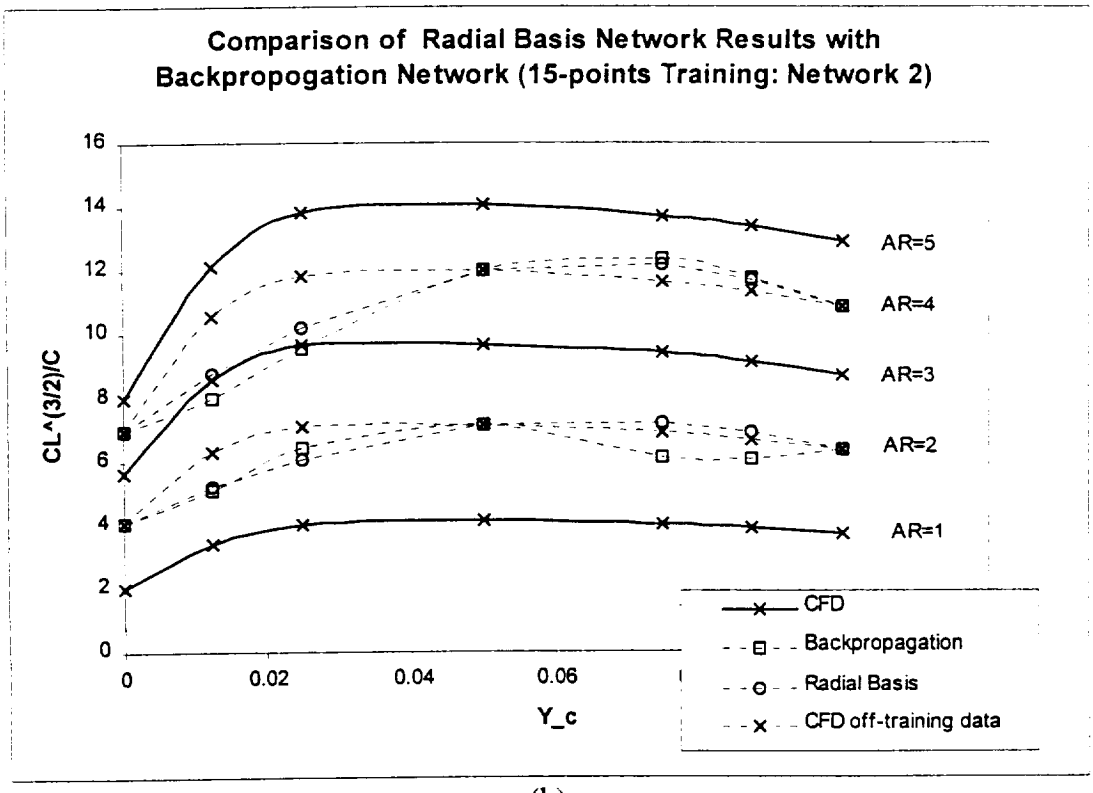


(c)

Figure 13. Comparison of Radial Basis Network with Backpropagation Network Results for 9-Points Training: Neural Network#1 (a), for 15- Points Training : Neural Network#2 (b), and for 25- Points Training : Neural Network#3 (c) (for y_c interpolation)



(a)



(b)

Figure 14. Comparison of Radial Basis Network with Backpropagation Network Results for 9-Points: Neural Network#1 (a), and for 15- Points Training: Neural Network#2 (b) (for AR interpolation)