To the Graduate Council:

I am submitting herewith a thesis written by Arthur Pedro Heath, Jr. entitled "Object-Oriented Signal Modeling Using an Adaptive Neuro-Fuzzy Inference System." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

J. D. Birdwell, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

OBJECT-ORIENTED SIGNAL MODELING USING AN ADAPTIVE NEURO-FUZZY INFERENCE SYSTEM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Arthur Pedro Heath, Jr.

December 1997

Copyright © Arthur Pedro Heath, Jr., 1997
All rights reserved

ACKNOWLEDGEMENTS

I would like to thank my major advisor, Dr. J. Douglas Birdwell, for his patience and for his suggestions in the research.

I am thankful to Dr. Tse-Wei Wang and Dr. Jack Lawler, for their technical advice and guidance especially in the latter stages of the research.

I am greatful to Sze Wei Ong for his useful suggestions for using LaTex for preparing of this thesis.

I greatly appreciate the successful effort in granting additional time for completion of this thesis rendered on my behalf by the following employees of Sandia National Laboratories, NM: Mr. Harvey Brewster, Mr. Dennis Miyoshi, Mr. Kip Stanley, Mr. J. D. Williams and Mr. Ken Holley.

Most of all, I express my greatest appreciation to God, and my parents for their unconditional love and great support especially in my greatest times of need.

This research project was sponsored in part by grants from E. I. DuPont and the Measurement and Control Engineering Center (MCEC), of the University of Tennessee, Knoxville. All process data was provided by a third party who agreed to its use in such a way that the process details are not disclosed. Software for generating object oriented signals was provided by Data Refining Technologies, Inc. The genetic algoritm used in the simulation is available as free software and can be found at the website listed in [18].

ABSTRACT

This research explores the use of fuzzy rules for capturing the underlying dynamics of object-oriented signals and for extracting relationships between signals. An Adaptive Neuro-Fuzzy Inference System (ANFIS) is used to generate the fuzzy rules. The ANFIS is able to model nonlinear, somewhat chaotic, signals which would be otherwise very difficult to model using traditional modeling techniques. A Matlab implementation of the ANFIS is developed, and the ANFIS is trained using a Genetic Algorithm. The ANFIS is trained and validated using highly nonlinear exothermic reaction process data.

This modeling approach proves to be well suited for the modeling and prediction of object-oriented signals whose characteristics are poorly understood and shows good potential for relationship extraction between pairs of signals. The approach works well for data sets which were available, and should scale well. It is reasonable to believe that using this modeling technique for controlling object-oriented signals is feasible since the use of fuzzy sets has been established as being very appropriate for controls applications.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction	**1**
2. Information Condensation	**5**
2.1 Data Refinement and Relationship Extraction	7
2.2 Summary	9
3. Fuzzy Sets, Membership Functions and the Rulebase	**10**
3.1 Membership Functions	10
3.2 Fuzzy Inference Systems	17
3.2.1 The Sugeno Fuzzy Model	18
3.3 Summary	20
4. Adaptive Networks	**21**
4.1 Neural Networks: Back-propagation	21
4.1.1 Local Minima	23
4.2 Summary	24
5. Adaptive Neural Fuzzy Inference System Architecture	**25**
5.1 The Hybrid Learning Rule	28
5.2 Summary	30

CHAPTER PAGE

6. Optimization: Minimization of Functions 31

6.1 Simulated Annealing 31

6.2 Genetic Algorithms 32

6.2.1 Chromosome Representation 33

6.2.2 Selection Functions 34

6.2.3 Selection Schemes 34

6.2.4 Genetic Operators 35

6.2.5 Population Initialization, Termination, and Evaluation Functions 35

6.2.6 Chromosome Representation Scheme Comparison 36

7. Interaction of the Simulation Model and the GA Optimizer 38

7.1 Training Data Selection 40

7.2 Time Averaged Prediction Error 41

7.3 Simulation Results 46

7.4 Object-Oriented Signal Modeling 47

7.4.1 Relationship Extraction: Case I 56

7.4.2 Relationship Extraction: Case II 75

8. Summary and Future Research Direction 85

8.1 Discussion 85

8.2 Research Summary 87

8.3 Future Research Direction 87

CHAPTER PAGE

BIBLIOGRAPHY **89**

References 90

APPENDICES **92**

VITA **126**

LIST OF FIGURES

FIGURE		PAGE
2.1	Raw Signal	6
2.2	Object-Oriented Signal	6
2.3	Object-Oriented Signal	8
2.4	Refined Object-Oriented Signals	8
3.1	Gaussian Membership Function	12
3.2	Fuzzy Union	14
3.3	Fuzzy Intersection	15
3.4	Fuzzy Complement	16
3.5	Sugeno Reasoning Mechanism	19
4.1	A back-propagation neural network	22
5.1	ANFIS Architecture	26
7.1	Interaction of GA and ANFIS/Simulation Model	39
7.2	Actual and Predicted Object Sequence.	42
7.3	Probability Density Function for x_{pred}	44
7.4	Probability Density Function for x_{actual}	45
7.5	Raw Signal	48
7.6	Initial Membership Functions for the Height and Midpoint	49
7.7	Initial Membership Functions for the Range	50
7.8	Height Cost Function Minimization	51

FIGURE **PAGE**

7.9 Midpoint Cost Function Minimization 52

7.10 Final Membership Functions for the Height 53

7.11 Final Membership Functions for the Height 54

7.12 ANFIS Prediction 55

7.13 Object-Oriented Signal 57

7.14 Initial Membership Functions for the Height and Midpoint 58

7.15 Initial Membership Functions for the Range 59

7.16 Midpoint Cost Function Minimization 60

7.17 Height Cost Function Minimization 61

7.18 Final Membership Functions for the Height 62

7.19 Final Membership Functions for the Midpoint 63

7.20 ANFIS Prediction 64

7.21 Unrefined Object-Oriented Signals for Case I. 66

7.22 Refined Object-Oriented Signals for Case I. 67

7.23 Initial Membership Functions for the Height and Midpoint. 68

7.24 Initial Membership Functions for the Range 69

7.25 Height Cost Function Minimization 70

7.26 Midpoint Cost Function Minimization 71

7.27 Final Height Membership Functions 72

7.28 Final Midpoint Membership Functions 73

7.29 ANFIS Prediction 74

7.30 Unrefined Object-Oriented Signals for Case II. 76

FIGURE **PAGE**

7.31 Refined Object-Oriented Signals for Case II. 77

7.32 Initial Membership Functions for the Height and Midpoint 78

7.33 Initial Membership Functions for the Range 79

7.34 Height Cost Function Minimization 80

7.35 Midpoint Cost Function Minimization 81

7.36 Final Height Membership Functions 82

7.37 Final Midpoint Membership Functions 83

7.38 ANFIS Prediction 84

CHAPTER 1

Introduction

Industrial process control, from small to large scale, often involves huge amounts of signal data that must be measured and controlled. Today's processes may involve manipulation of thousands of signals which may be sampled at a rate of one second or more over a period of years. With such huge amounts of data, data information extraction is no trivial task. Due to the volume of data, information extraction is often obstructed. This information, which in some cases could be used for preventive measures and quality control, is often ignored [1].

Modeling a typical industrial process may involve collecting from one month to twelve months of data. This requires an incredible amount of storage space in terms of megabytes. Consider the following: One thousand signals which are sampled once per second would require 345 Mbytes of storage space for one day of monitoring. Viewed on a monthly basis, storage of such amounts would be impractical and costly. In addition, the data contain noise. Reduction of signal noise and identification of transient data, for example, could be easy with the human eye; however, this task has proven to be quite difficult to automate [1].

Raw data relationships for cases considered here are typically nonlinear and "fat"; that is, the relationship can not be modeled as a curve since it is dependent upon

variables which are unobserved and unmodeled. Thus, standard statistical methods, such as Auto-regressive Moving Average (ARMA) or nonlinear parametric models for raw data analysis are inappropriate [1].

This thesis proposes a fuzzy inference system coupled with neural network learning capabilities for modeling the "fat" characteristics of signals. A method developed by Data Refining Technologies, Inc. called condensation, is used to reduce the size of the modeling problem by reducing, by one to two orders of magnitude, the quantity of data used for modeling. This condensation step, however, produces discrete signals which have values represented by three real-valued parameters rather than a single number. Each value can be visualized as a rectangle enclosing a range of values of the original "raw" signal over a time interval. These object-oriented signals introduce a new type of system modeling problem which has not been addressed in the literature. Informally stated, the problem is to create an algorithm which uses the parameters describing one or more objects of an input signal stream to predict the parameters of the next object to be generated on an output signal stream.

This thesis describes one of two research projects which has attempted to model dynamic of single signals and relationships between object-oriented signals. Both approaches propose a structure reminiscent of finite impulse response (FIR) filters—the predicted parameter values depend only upon the parameter values on the last k input objects, where k is a fixed positive integer. The other research project utilizes a Markov Chain approach and has been reported in [17]. This research utilizes neural network concepts in combination with fuzzy relations to form a linear combination of linear

models derived using least squares methods, where the coefficients which appear in the linear combination are generated by the neural-fuzzy network. Both approaches appear promising; however, the limited amount of process data available and the inability to run either model against the actual process behavior preclude a definitive judgment. More research is needed to make an informed choice of modeling approach.

Chapter 2 examines the idea that the data can be represented as signal objects which capture these "fat" characteristics. These objects have many of the same characteristics as the raw signal; however, the amount of represented data is far less, and relationships between variable object data can be developed.

Chapter 3 provides a background of fuzzy set theory and fuzzy reasoning. Membership functions, the fuzzy rulebase and fuzzy reasoning are discussed. The Sugeno fuzzy model is presented as it relates fuzzy set theory and fuzzy reasoning.

Chapter 4 considers adaptive network artchitecture and commonly used adaptive network-based models. Learning procedures such as the back propagation learning algorithm are presented. In this thesis, we consider using an Adaptive Neural Fuzzy Inference System or ANFIS [2] for identifying variable relationships and for making object parameter predictions. The ANFIS is capable of achieving highly nonlinear mappings; thus it is ideal for making nonlinear parameter predictions which Jang successfully illustrates in [4]. Secondly, the ANFIS is able to model the central dynamics of a system without *a priori* information. Lastly, ANFIS may require far less data for prediction than linear predictive methods, but may achieve much greater prediction precision as noted in [2]. Chapter 5 provides the architectural details of the ANFIS

which blends fuzzy set representation, fuzzy reasoning and neural network learning capability. The hybrid learning method, which combines least-squares estimation and the minimization of the error function, is presented.

Generally, a learning rule for an adaptive network is an optimization technique applied to minimize a network error function. Chapter 6 discusses an optimization method known as the Genetic Algorithm and unconstrained optimization in general.

The tests of the neural-fuzzy method using the available data are presented in Chapter 7, with a summary in Chapter 8.

CHAPTER 2

Information Condensation

An approach considered by Data Refining Technologies [1] is to view a signal as "sets of interval measures rather than as point values". Signals represented in this fashion are known as object oriented signals. This representation effectively condenses the signal resulting in interval measurements that are as detailed as the original signal, thus retaining signal information and preserving variable relationships, unlike moving average methods which reduce data but distort variable relationships. Unlike compression technologies, condensed signals can be utilized directly without decompression to their original forms.

Consider the signal in Figure 2.1. The first step in condensation is to "scrub" the signal. By doing so, transient, steady state and noisy portions of the signal can be identified. We can then develop a relationship between variables that is largely dependent on "quasi-steady state" data. By "scrub", we mean that once processed, the data is now described as a sequence of rectangular objects where each object's parameters mainly depend upon raw data transient and "quasi-steady state" characteristics for the object's interval. Each object, as shown in Figure 2.2, is represented by three parameters: height, midpoint, and range. Once the data are scrubbed, a signal object has the similar characteristics to the raw data within the object, but the volume of data has

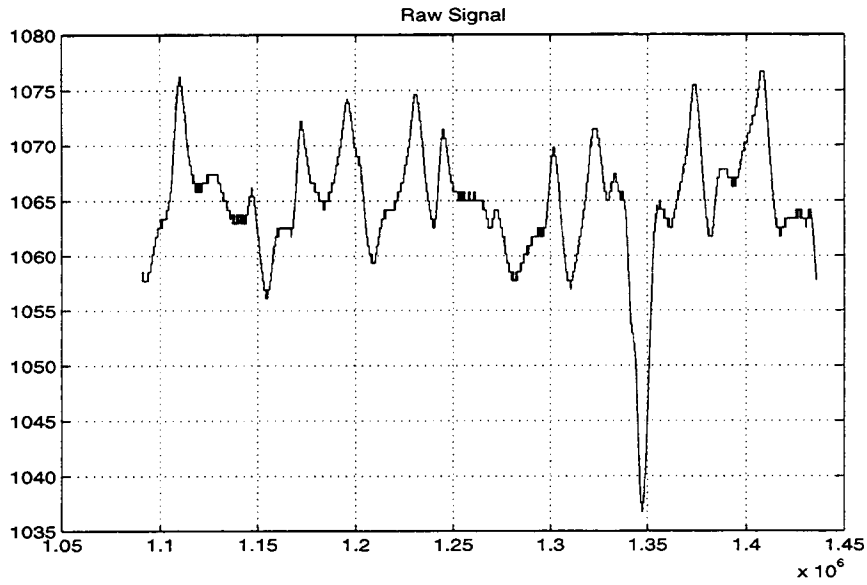


Figure 2.1: Raw Signal

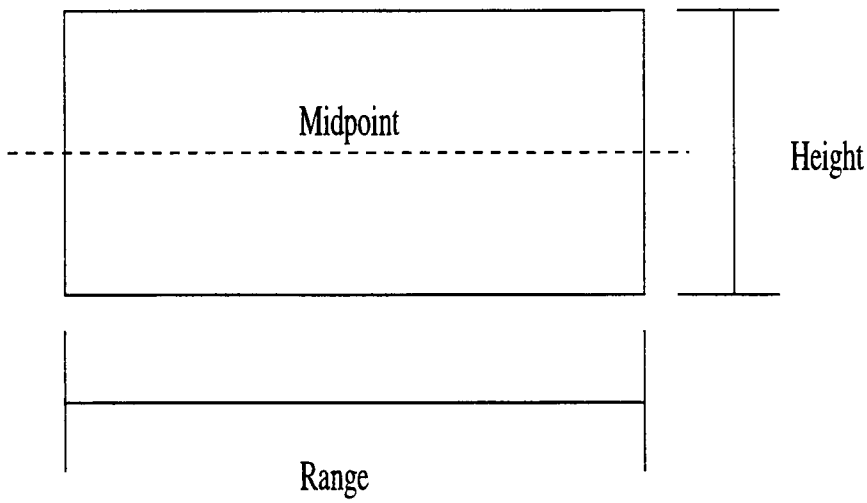


Figure 2.2: Object-Oriented Signal

been reduced since the signal description is now represented by object data instead of raw data. Volume reductions of as much as 500:1, raw data to object data, have been achieved [1].

The resulting object-oriented signal is shown in Figure 2.3.

2.1 Data Refinement and Relationship Extraction

After scrubbing sets of data, the time intervals of the data must be refined. For most processes, the number of, for example, input and output objects will not be the same. Refinement truncates both ends of each object sequence as required so that all object sequences cover the same time interval with common starting and ending times. The objects of all signals are then segmented such that object endpoints for corresponding objects of the two sequences are equal. The final result is a set of signals whose relationship is now one-to-one.

The refined object-oriented signals are shown in Figure 2.4. The question of modeling object signals and extracting relationships between sets of object data arises. As previous noted, we will use an ANFIS for making parameter predictions. The following chapters provide background information for understanding the structure of the ANFIS.

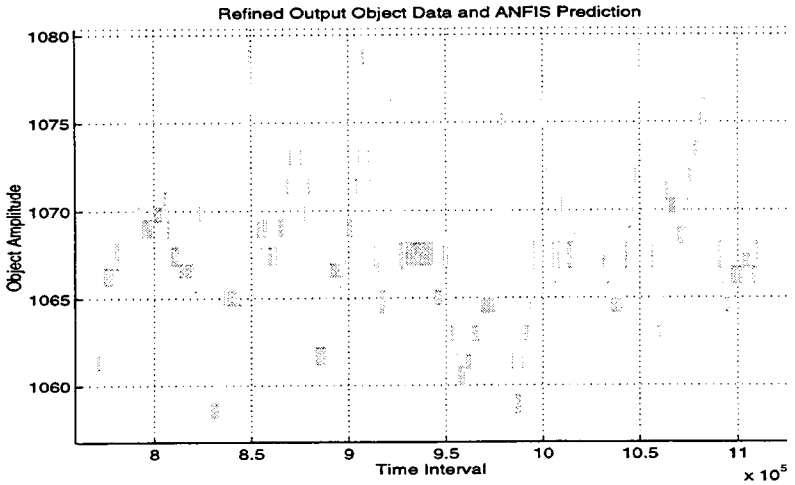


Figure 2.3: Object-Oriented Signal

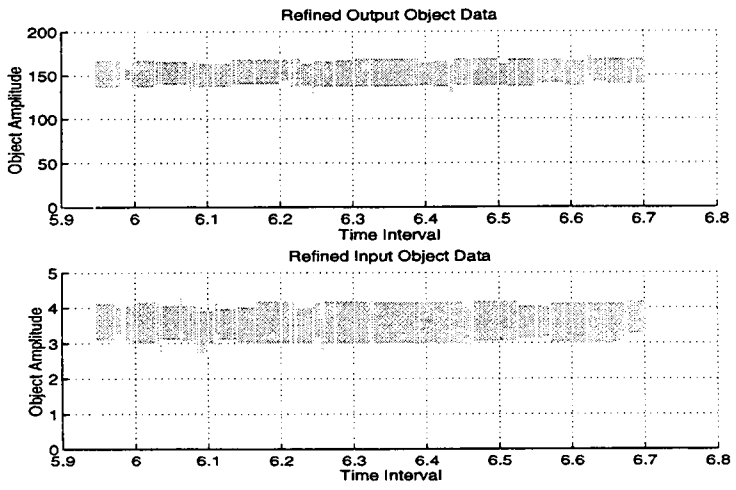


Figure 2.4: Refined Object-Oriented Signals

2.2 Summary

We have briefly discussed information condensation as a means of data reduction and the use of object oriented signals for data representation as opposed to single point values. By using interval data representation, the volume of data can be greatly reduced and traditional indentification methods can be used to model this representation.

CHAPTER 3

Fuzzy Sets, Membership Functions and the Rulebase

Fuzzy set theory, introduced by Zadeh [5], explores the use of "uncertain and imprecise" set descriptions. Unlike classical sets, fuzzy sets are without crisp, or clearly defined boundaries. Though set descriptions may be crisp, having definite boundaries or fuzzy, fuzzy sets are better suited for modeling imprecise behavior. Given a continuous or discrete set X, each member of X can be assigned a value or grade which is usually on the closed interval $[0, 1]$. Such mappings are referred to as membership functions or MF and are denoted by μ. The set X is referred to as the universe of discourse. Membership functions provide a transitional description of X, whereas the representation for classical sets would define only at the set boundaries and would be the characteristic function. Once the universe of discourse is defined, membership function parameters can be appropriately chosen.

3.1 Membership Functions

Commonly used membership functions include triangular, trapezoidal, and the generalized bell curve. Each MF has a parameter set which is defined by the universe of discourse. For this discussion, the Gaussian MF will be used. This choice is arbitrary

and the MF has the following form:

$$\mu_A = e^{-(\frac{x-c}{a})^2} \tag{3.1}$$

where "a" is the width of each curve and "c" is the center or mean. The Gaussian MF is shown in Figure 3.1.

Membership functions can provide descriptions for vague concepts such as "hot", "cold" and "warm". This is done by defining a MF which is non-zero over the range of likely values the underlying variable will assume if described by that concept. For example, an MF for temperature could be defined as follows: the universe of discourse or range of values for "cold" could represent temperatures from -60 to 20°C and "warm" could represent temperatures from 60 to 75°C. How these ranges are defined is subjective. Consequently, phrases such as "the air is warm" or "the humidity is high" can be modeled using linguistic expressions that determine "how warm" and "how high" via the rulebase. The rulebase can be constructed from human expert knowledge or by inference and provides a system description relating, for example, inputs and outputs, or current state and next state. The rulebase is generally derived from input/output data.

The rulebase consist of if-then rules of the following format:

$$If\ y\ is\ A, then\ z\ is\ B$$

where A and B are MFs defined on universe of discourse Y and Z, respectively. "y is A" is known as the antecedent and "z is B" is known as the consequence. Fuzzy rules

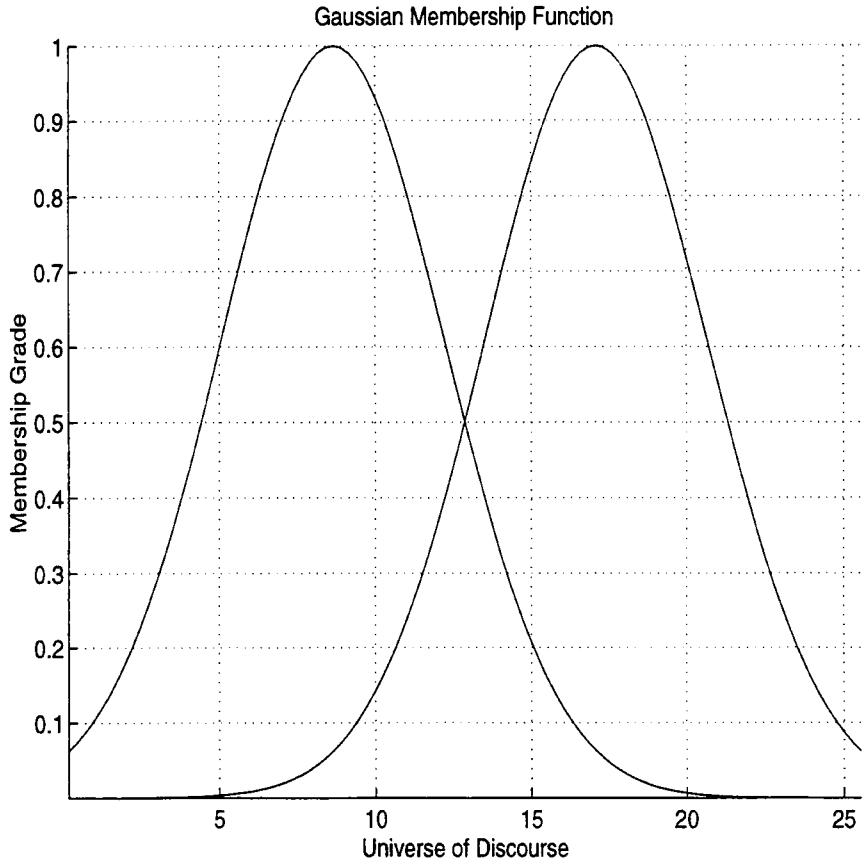


Figure 3.1: Gaussian Membership Function

can have a single antecedent and a single consequence, or multiple antecedents and a single consequence.

We now consider basic fuzzy set properties and operations. We define containment as follows: Fuzzy set A is contained in or is a subset of set B if and only if $\mu_A(x) \leq \mu_B(x)$ for all x. The first of the basic operations is the union or disjunction:

Union Given two fuzzy sets A and B, with $\mu_A(x) \Leftrightarrow A$ and $\mu_B(x) \Leftrightarrow B$, where "$\Leftrightarrow$" implies that for every element of set A or B, there exist a corresponding membership grade defined by μ_A or μ_B, and vice versa, the union of A and B yields fuzzy set C and can be written as C = A $\cup$ B = A OR B. The equivalent MF for C can be expressed as

$$\mu_C(x) = max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x). \tag{3.2}$$

The union is shown in Figure 3.2.

Intersection Given two fuzzy sets A and B, with $\mu_A(x) \Leftrightarrow$ A and $\mu_B(x) \Leftrightarrow$ B, the union of A and B yields fuzzy set C and can be written as C = A $\cap$ B = A AND B. The equivalent MF for C can be expressed as

$$\mu_C(x) = min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x). \tag{3.3}$$

Figure 3.3 demonstrates the fuzzy intersection.

The last of the operators is the complement or negation.

Complement Given fuzzy set A, and $\mu_A(x) \Leftrightarrow$ A, the complement of A is denoted by $\overline{A}$ and is defined as follows and is shown in Figure 3.4.

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x). \tag{3.4}$$

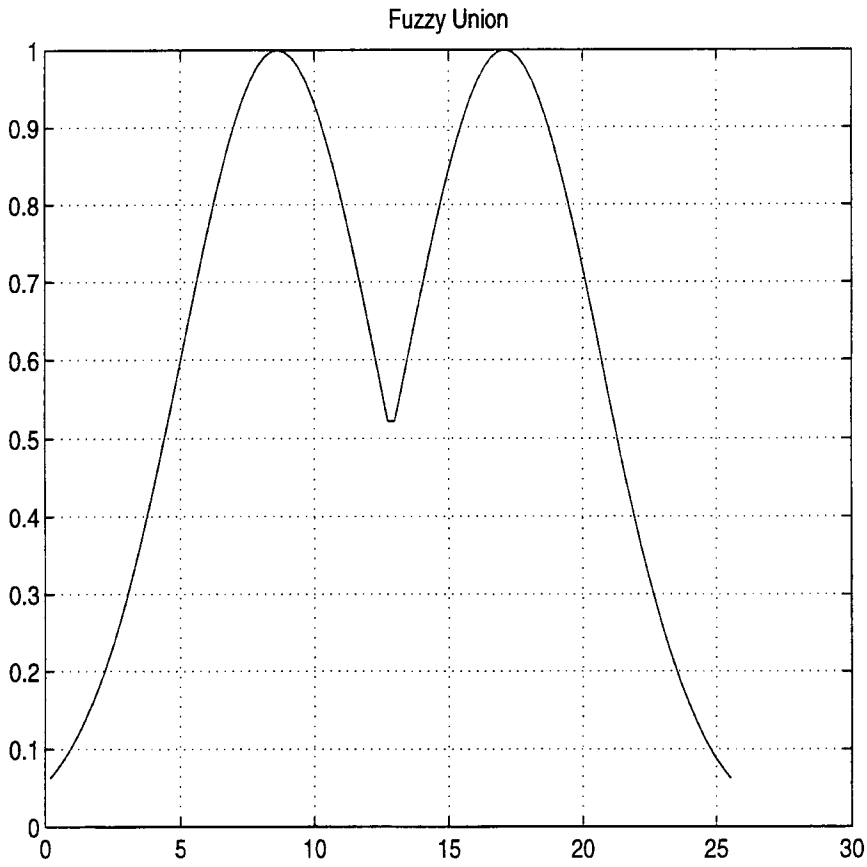


Figure 3.2: Fuzzy Union

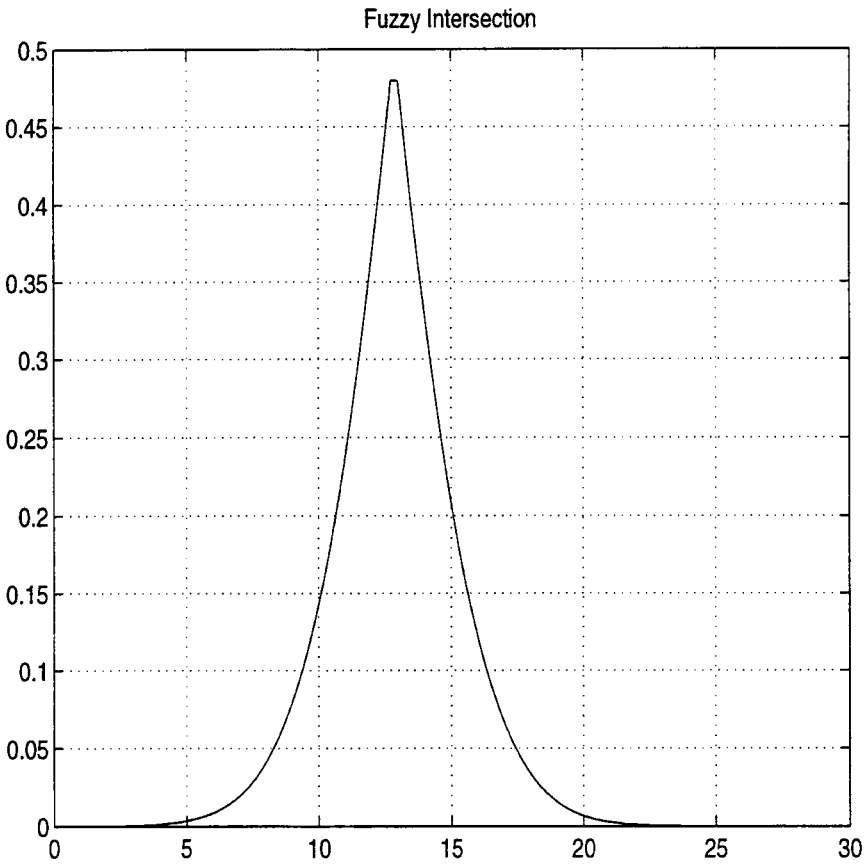


Figure 3.3: Fuzzy Intersection

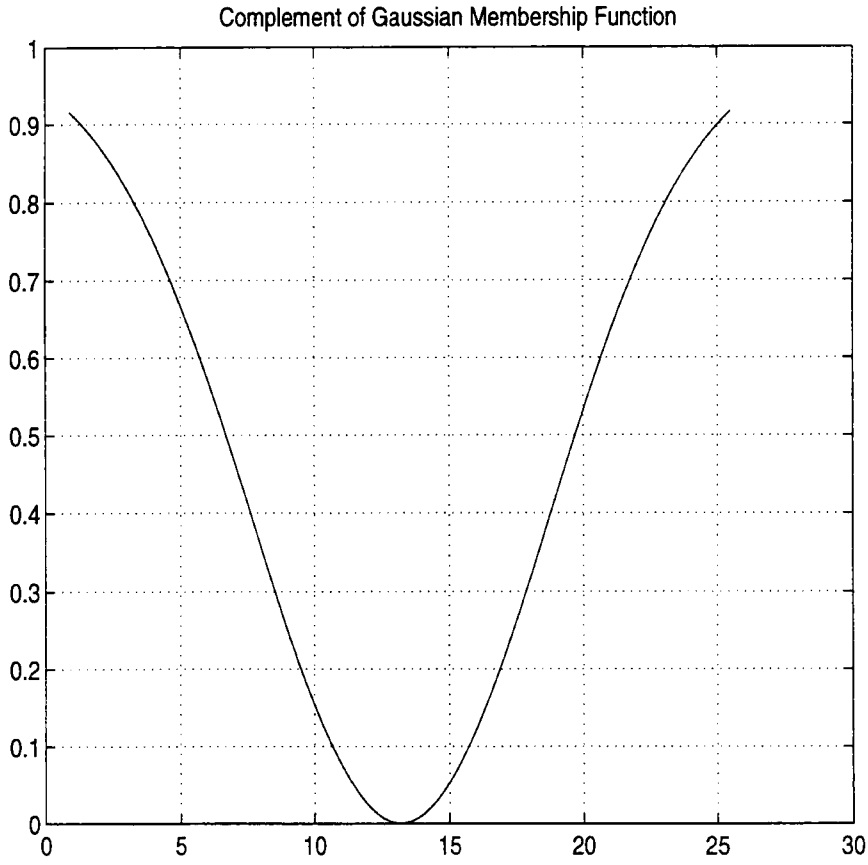


Figure 3.4: Fuzzy Complement

3.2 Fuzzy Inference Systems

A fuzzy inference system, or fuzzy model, is defined by a rulebase, which is a representation of fuzzy rules, a database, which defines the membership functions and associated parameters, and a reasoning mechanism, which performs the reasoning procedure based on the fuzzy rules and produces a "reasonable" output [3].

As previously noted, set descriptions may be crisp or fuzzy. Fuzzy inference systems may accept crisp inputs or fuzzy inputs. However, the outputs of most systems are fuzzy. Therefore, a defuzzification method is needed in order to have a crisp output. There are several strategies for defuzzifying fuzzy outputs. Two popular methods are the centroid method and the bisector method. It is preferred that fuzzy system inputs and outputs be crisp, in which case the fuzzy model serves as a direct nonlinear map from the input space to the output space.

There are three basic fuzzy inference models: the Mamdani fuzzy model, the Tsukamoto fuzzy model, and the Sugeno or TSK fuzzy model. The first model has fuzzy outputs and requires a defuzzification strategy. The Tsukamoto and Sugeno models have crisp outputs so defuzzifying the outputs is not required. In the following section, we introduce the Sugeno fuzzy model, which is our model of interest. The fact that this model does not requires a defuzzification strategy makes it attractive. In addition, it's inference mechanism is intuitively reasonable.

3.2.1 The Sugeno Fuzzy Model

The Sugeno fuzzy model was first introduced by Tagaki, Sugeno, and Kang (hence the acronym TSK) [3]. The basic rule structure for the Sugeno fuzzy model is

$$if\ x\ is\ A\ and\ y\ is\ B, then\ z = f(x, y).$$

The output z is typically a linear combination of the inputs, which are in this case x and y. When the model output is a linear combination of the model inputs, the model is said to be first order. For a constant output, the model is said to be *zero order*.

Figure 3.5 illustrates the reasoning mechanism of the Sugeno fuzzy model. Notice that the output is crisp; so, no computationally burdensome defuzzification strategy is required. The defuzzification method has been replaced by the weighted average operation [3]. This reasoning mechanism determines weights, w_i and $\overline{w_i}$ (which is w_i normalized) by finding the fuzzy intersections of A_i and B_i. The parameters $\{p_i, q_i, r_i\}$ of function f_i are generally found by a least squares method. Thus, the model provides a straightforward systematic way of producing a desired input-output mapping.

A problem arises for the case where two fuzzy rules overlap. This means that for the same input, the two rules would produce different results. The TSK fuzzy model is, itself, a possible solution to this problem. Associated with each rule is a weight. These weights govern which rules contribute most to the overall output of the model and the weight distribution is chosen such that the overall error is minimized.

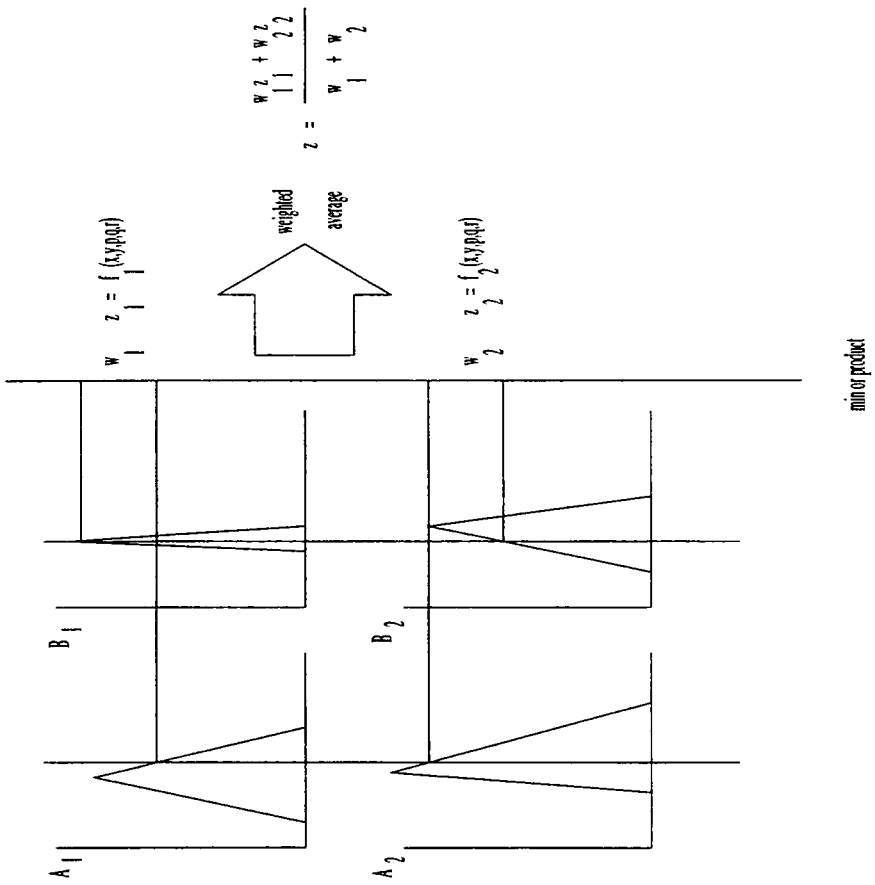


Figure 3.5: Sugeno Reasoning Mechanism

3.3 Summary

We have presented information which is necessary for understanding fuzzy inference systems, which began with a discussion of fuzzy sets and fuzzy operations, membership functions and the rulebase. The Sugeno fuzzy model, a type of inference system, was discussed.

CHAPTER 4

Adaptive Networks

We now briefly discuss a common architecture and learning algorithm for adaptive networks, from which the ANFIS derives its learning capabilities. An adaptive network is "a network structure whose overall input-output behavior is determined by the values of a collection of modifiable parameters" [3]. The structure is composed of directly linked nodes which may be fixed or adaptive. By fixed we mean that the parameters for the node are fixed. Similiarly, adaptive node parameter sets are modifiable.

4.1 Neural Networks: Back-propagation

From the adaptive network architecture come the two most popular network configurations: the back-propagation neural network (BPNN) [6] and the radial basis function network (RBFN) [6]. We are primarily interested in the concept of the learning procedure of the former, so that we may examine its disadvantages and explore possible alternatives.

A back-propagation neural network is shown in Figure 4.1. Figure 4.1 is a typical architecture for a back-propagation neural network with three inputs, two outputs, and three hidden nodes. Each node has a node function which, for this type of architecture, is the same for each node.

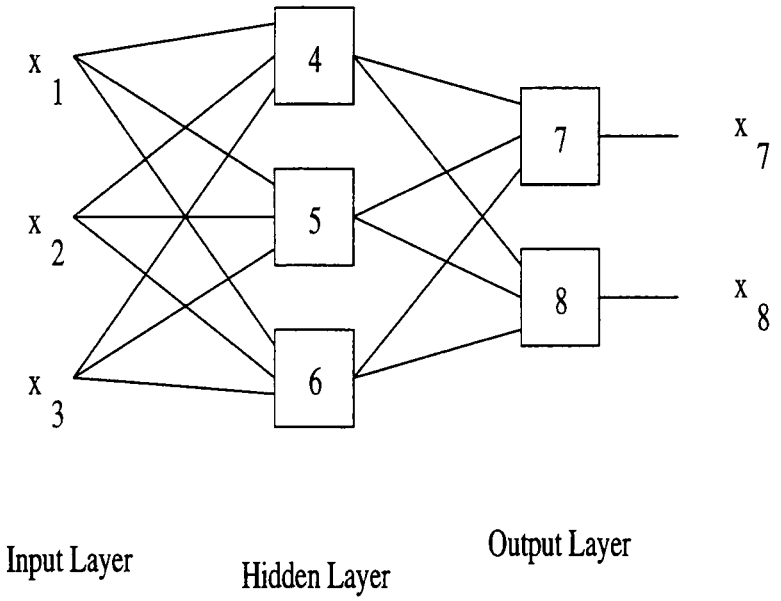


Figure 4.1: A back-propagation neural network

The node functions are generally compositions of a linear and a sigmoidal function [2]. The back-propagation learning procedure refers to the way gradient information is accumulated by propagating error information back through a network and is used (in conjunction with an optimization scheme) for adapting modifiable parameters. Back-propagation is essentially a gradient descent method [3]. Gradient descent methods minimize an error E by adjusting network parameters so that these parameters produce a desired input-outupt mapping. Before attempting to use the gradient method, a gradient vector must be obtained. By definition, the gradient vector is the derivative of the error measure with respect to all associated parameters [3]. In other words, the error E is described by small changes in network parameters and is minimized by changes in the parameters in the direction of negative gradient or "downhill" on an error surface.

In order to correctly apply the back-propagation learning algorithm, the objective function must be properly represented in terms of all of the parameters of the network being trained. A systematic way of doing so is to find the network's error propagation model which involve reversing node links and summing error signals which are described by the appropriate partial derivatives. Though systematic, it is tedious. For a detailed description of the back-propagation learning algorithm, see [6].

4.1.1 Local Minima

Since gradient search methods search in the direction of negative gradient, these methods provide no "uphill" mobility. Minimizing the error corresponds to finding the

lowest point, the *global minimum*, on an error surface. All other minima are considered *local*. Since such methods only search in a "downhill" direction, the first minimum encountered, whether truly global or local, is viewed as global, and the minimization procedure terminates, thus yielding a solution which may not be optimal [6]. This can lead to a failure of the gradient search.

4.2 Summary

This chapter has covered the general characteristics of adaptive systems from which the ANFIS is developed. Common adaptive network architectures are mentioned and the back-propagation learning algorithm is briefly discussed. Architectural nodes are defined and entrapment at local minima, a problem associated with back-propagation, is discussed.

CHAPTER 5

Adaptive Neural Fuzzy Inference System Architecture

As previously stated, the fuzzy if-then rules of the rulebase can be developed through human expert knowledge of a system or by inference. The ANFIS architecture is essentially a Sugeno fuzzy model equipped with learning capabilities of a neural network. It identifies membership function parameters and the rulebase that produce a particular input-output mapping. For example, we have chosen the Gaussian as our membership function. The function has several parameters known as *premise* parameters that determine its shape. To identify the appropriate input-output mapping, the ANFIS determines what premise parameter set(s) will produce the desired mapping [3].

A typical ANFIS consists of five layers as shown in Figure 5.1.

For simplicity, we use a two input, two rule first-order Sugeno fuzzy model. We again define adaptive and fixed nodes. An adaptive node is a node which has modifiable parameters while a fixed node has fixed parameters. In the figure, adaptive nodes are represented by squares and fixed nodes are represented by circles. The first layer is an adaptive node with the node output as

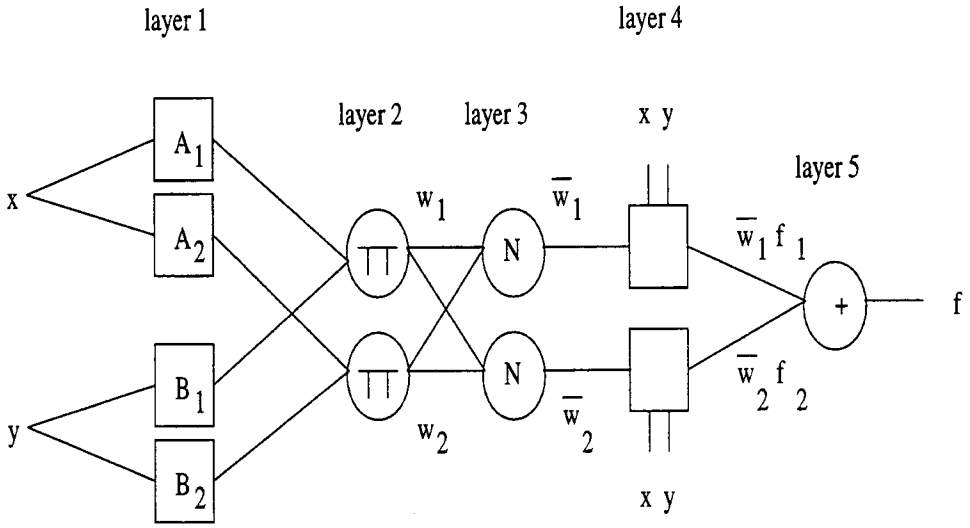


Figure 5.1: ANFIS Architecture

$$O_{1,i} = \mu_{A_i}(x), \quad for\ i = 1,2\ \ or\ O_{1,i} = \mu_{B_{(i-2)}}(y), \quad for\ i = 3,4 \tag{5.1}$$

where x or y are node inputs and A (or B) is the membership function associated with the node. These outputs are essentially the membership values for the premise portion.

Layer 2 consist of fixed nodes which multiply pairs of outputs of layer 1 as shown in the architecture. Each node output in this layer represents the firing strength of a rule. Essentially,

$$O_{2,i} = w_i = \mu_{A_i}(x)\mu_{B_i}(y), \quad for\ i = 1,2. \tag{5.2}$$

Layer 3 also consists of fixed nodes. The i-th node calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths. This ratio is known as the normalized firing strength and is simply:

$$O_{3,i} = \overline{w_i} = \frac{w_i}{w_1 + w_2}, \quad for\ i = 1,2. \tag{5.3}$$

Layer 4 has adaptive nodes with corresponding node functions

$$O_{4,i} = \overline{w_i}(p_i x + q_i y) = \overline{w_i} f_i$$

(5.4)

where the set $\{p_i, q_i\}$ is referred to as the *consequent* parameter set and is determined by means discussed in Chapter 3.

The last layer computes the total output weight contributions of each incoming node output from the previous layer or,

$$O_{5,i} = \sum_{i=1}^{2} \overline{w_i} f_i \tag{5.5}$$

For this architecture, the 2-rule system is of the format:

$$Rule\ 1: \quad if\ x\ is\ A_1\ and\ y\ is\ B_1, then\ f_1 = p_1 x\ +\ q_1 y.$$

$$Rule\ 2: \quad if\ x\ is\ A_2\ and\ y\ is\ B_2, then\ f_2 = p_2 x\ +\ q_2 y.$$

The objective of an ANFIS is to approximate a desired input-output mapping relationship by suitable selection of the adaptive parameters. This objective will be formulated as an optimization problem over the adaptive parameter set, which will be solved using a genetic optimization algorithm.

5.1 The Hybrid Learning Rule

For the hybrid learning procedure, the ANFIS is given a complete set of training data. For batch mode learning as described by Jang in [2], a forward and backward pass through the ANFIS architecture is required. In the forward pass, the architecture is given the set of input vectors (training data) and each node output is calculated. This is repeated until each row of A and b is obtained where A corresponds to the ANFIS inputs and b to the outputs. Jang points out in [2] that if an adaptive network has an output that is linear in its network parameters, then these parameters can be

identified by linear least-squares methods. The problem of interest is the well-known formulation,

$$Ax = b \tag{5.6}$$

For our consideration, b may represent more than one output in which case x is also a matrix. We are interested in finding the best solution x which maps A to b and minimizes the error $|| Ax - b ||$, where the norm is for vectors and is the matrix norm induced by the Euclidean distance norms on the underlying vector spaces for matrices. A traditional means of accomplishing this is by using the projection matrix

$$x = (A^T A)^{-1} A^T b \tag{5.7}$$

A better way is using the singular value decomposition (SVD) of A and constructing its pseudoinverse. One reason for this is the above projection matrix can be shown to be numerically unstable under certain conditions and requires that $(A^T A)^{-1}$ be invertible [11]. At this stage, error measurements can be obtained for the training data set. This method of parameter identification can be used in conjunction with a form of the gradient descent method called back propagation, which propagates the error back from the output to the input. This backwards pass is done for the set of training data and the parameters in x are updated. More specifically, node output calculations continue until layer 4, and the consequent parameters will be identified by a least-squares method. The error will then be propagated backwards through the

network and the premise parameters will be updated.

The parameter identification procedure outlined above minimizes the error by propagating error information back through the network. While the least squares portion of the procedure will be retained, due to the disadvantages of back-propagation previously outlined, we choose to use a Genetic Algorithm instead. This optimization scheme effectively minimizes error with, quite possibly, fewer iterations than simulated annealing. Due to the nature of the search, convergence to local minima can be avoided, unlike gradient search methods which search "downhill" only.

5.2 Summary

This chapter discusses the architectural details of the ANFIS. A layer-by-layer description of the ANFIS is given. A hybrid learning algorithm is presented along with a detailed discussion on how the premise parameters can be adapted and how the consequent parameters are generated.

CHAPTER 6

Optimization: Minimization of Functions

The term "optimization" can be summarized as follows: Given a function f, which may be of one or more variables, find the value of each variable such that f attains it minimum (or maximum) value. This is the main objective of optimization.

There are several types of optimization schemes available, both constrained (imposed restrictions as to where a solution may be located) and unconstrained, to accomplish this. In particular, we are concerned with minimizing error measures. Since restrictions on the solution may prevent obtaining one that is minimal, it is unconstrained optimization which holds our interest. In the following sections, we discuss methods which is suitable for solving both linear and nonlinear problems.

6.1 Simulated Annealing

A more reliable, though computationally costly, way of escaping local minima than gradient search is simulated annealing [7]. With this method, all adaptations, or movements on the error surface, have associated probabilities of occurence. Thus, uphill mobility is possible which implies escaping local minima is possible. By "freezing" the system, that is, by decreasing the probability of adaptations that cause an increased value of the objective, the process converges to the global minimum, although this

convergence is not always a guarantee. Though effective, this method is more costly due to the substantial number of iterations required relative to the requirements for iterative improvement methods like back-propagation [8]. In addition, for a large number of parameters and depending on the nature of the optimization problem, a "good" solution may not be obtained at all.

6.2 Genetic Algorithms

Genetic algorithms have been used to solve difficult problems whose objective functions are not necessarily continuous, differentiable, or smooth and that are not guaranteed a single minimum. By simulating natural selection or "survival of the fittest", genetic algorithms "breed" new solutions from a population or selection of the most "fit" solutions. With each generation of solutions, the genetic algorithm successively improves the solution. Genetic algorithms have been shown to solve both linear and nonlinear problems by exploiting such properties as crossover, mutation, and selection operations applied to individuals in the population [18].

Genetic algorithms have several advantages over standard search methods, namely the ability to avoid local minima. In addition, genetic algorithms do not require gradient or surface curvature information. Usually, standard search methods require that an objective function be explicitly expressed. Generally, the objective function for the genetic algorithm can be a simple expression that does not have to be expressed in terms of the parameter variables to be optimized. Though potentially expensive com-

putationally, there are ways to increase the speed of the search depending on the nature of the objective function (which is influenced by the associated model) and how the parameters to be optimized are genetically represented.

One serious shortcoming of traditional optimization methods is that the objective function is predefined. Such methods are generally adopted because of quick convergence properties. Thus, the objective function may not appropriately describe the inherent properties of the data.

The use of the genetic algorithm requires discussion of six fundamental issues: chromosome representation, selection function, genetic operators (crossover and mutation), initial population, termination criteria, and the evaluation function.

6.2.1 Chromosome Representation

All genetic algorithms (GAs) require that each individual of the population be genetically represented. The representation scheme determines how the problem is structured in the GA and also determines the genetic operators that are used. Individuals or chromosomes are made up of a sequence of genes from a certain alphabet. An alphabet may be composed of binary digits, floating point numbers, integers, matrices, etc. Each gene represents a scalar value. Each chromosome represents a point in the solution space. Chromosomes are represented in either binary or floating point form. For the binary representation, each gene of a chromosome is represented as a string of bits. The precision of the representation depends on the number of bits used. The length of the binary form chromosome depends on the desired precision of the solution and the number of

parameters to optimize. The length of the floating point implementation is the same length as the search parameters.

6.2.2 Selection Functions

Since GAs "breed" new solutions from the most fit individuals, the selection of individuals is extremely important. The selection method has a great impact of the rate of convergence and the success of the search in general. Selection is the process of choosing two parents for crossing to create new individuals. A probabilistic selection is performed based on an individual's fitness such that the best individuals have an increased chance of being selected. Individuals can be choosen more that once so the the "fittest" individuals are retained and contribute to the next generation. There are several schemes for the selection process: roulette wheel selection with scaling techniques, tournament and probabilistic ranking methods [18].

6.2.3 Selection Schemes

As stated, selection plays an extremely important role in the success of the GA. Solutions which produce good fitness values are solutions which correspond to optimal or near optimal parameter sets. A "good" fitness depends on the optimization objective. One selection method which is based on the fitness of the individual is roulette selection. "The principle of roulette wheel selection is a linear search through a roulette wheel with the slots of the wheel weighted in proportion to the individual's fitness value" [14]. A random proportion of individual fitness and the sum of the fitness values

in the population yields the probability that a particular individual is choosen. This type of selection does not guarantee that the best individual will be choosen. In addition, roulette wheel selection only allows for maximization of functions unless some type of scaling technique is incorporated [14].

Ranking methods such as normalized geometric ranking, assign probabilities of selection to individuals based on the rank of solutions when all solutions are sorted. Tournament selection, a ranking method, chooses individuals randomly instead of according to assigned probabilities, and replaces old individuals with new individuals.

6.2.4 Genetic Operators

The genetic operators are the heart of the GA. The operators are used to create new solutions based on existing solutions in the population. These operators are mutation and crossover. Crossoever takes two individuals or solutions and produces two new individuals. Mutation alters one individual to produce a single new solution. Both operations (which may differ according to genetic representation) can be used with a binary or floating point representation of the chromosomes. For more details, see [14].

6.2.5 Population Initialization, Termination, and Evaluation Functions

The GA must be given an initial population. It is common to randomly generate solutions for the entire population. However, genetic algorithms can iteratively improve known solutions. It is possible for the population to contain several good solutions while the other solution can deviate randomly from the known solutions or be totally random.

By seeding the population with known solutions, the time needed for the GA to find an optimal or near-optimal solution can be greatly reduced. This could also permit smaller population sizes and reduce the amount of search time required. However, care should be taken not to produce a population which does not contain a sufficient amount of diversity.

The GA continues to breed new "offspring" solutions from generation to generation until some termination criterion is met. A common method of termination is specifying a maximum number of generations. Generally, a GA will cause the population to converge to a single solution, so population convergence is a possible criterion. However, this convergence may be premature, resulting in a suboptimal solution. When individuals cease to deviate significantly from one another, the algorithm could terminate. If after many successions the GA can not improve the solution, the algorithm could be terminated.

The evaluation or fitness function can take on many forms for use in the GA. It is a description that is appropriate for the objective function being optimized. This description is subjective and problem-dependent. Fortunately, the evaluation function does not have to be expressed in terms of the parameters to be optimized.

6.2.6 Chromosome Representation Scheme Comparison

How individuals are represented greatly impacts the effectiveness of the search. Binary representation encodes individuals as strings of bits where each gene of the string has the same number of bits. Genetic operators are then applied to these strings of

bits. In the floating point implementation scheme, a chromosome is represented as a vector of numbers of the same length as the search parameters. In general, the precision of the floating point scheme is better than the binary representation itself. The binary representation scheme sometimes suffers from premature convergence of the entire population to a single local solution. To alleviate this problem, we introduce a noise assignment scheme to the parameters to be optimized [14]. First, an individual is decoded and noise is added to that individual. If the individual is "fitter", the modified individual is returned to the population; otherwise, the original individual is placed back into the population. This scheme enables the binary GA to achieve high precision and fine tune the solution locally. A small amount of noise does not impair the GA's search ability. However, large amounts of noise tend to degrade the search.

CHAPTER 7

Interaction of the Simulation Model and the GA Optimizer

Figure 7.1 provides a flowchart showing the interaction of the simulation model and GA optimizer. An individual of the GA possesses the parameter information of the ANFIS predictor, such as membership functions and fuzzy rules. The performance of the predictor with a simulation model is gauged based on the fitness of the trial individual which takes the form of the sum-of-squared error. Various parameter sets are generated from the optimizer and the ANFIS/simulation model is the objective function module for calculating the fitness of each trial individual.

An individual from a GA represents one trial set of fuzzy membership functions. The ANFIS predictor involves 6 inputs, each input having at least 2 membership functions. The membership function of choice is the Gaussian, which is represented by two parameters. For this scenario, there are a total of 24 parameters required to specify the membership functions of the predictor. In some cases, the consequent parameters could be included in the trial individual description. However, it would be senseless to use the GA to optimize parameters which are linearly related. Since the local linear models of the ANFIS will be retained, standard LSE methods are sufficient for finding the consequence parameter set. The floating point representation scheme is used to represent an individual.

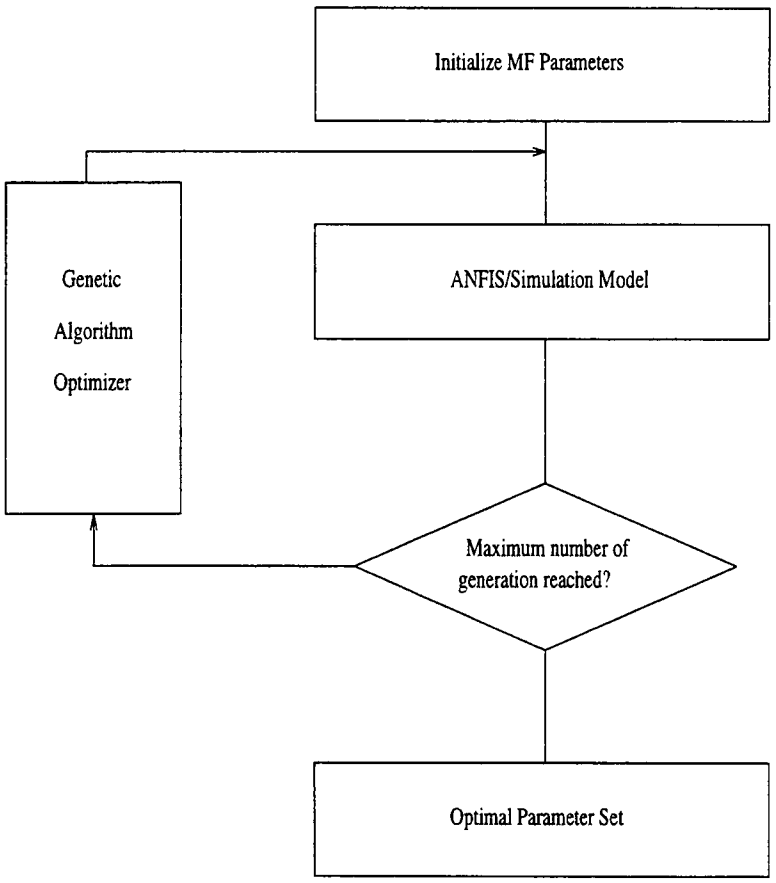


Figure 7.1: Interaction of GA and ANFIS/Simulation Model

When the simulation begins, the initial membership function parameters and fuzzy rules and generated. These initial parameters (membership function parameters) make up the first individual. The particular GA used requires that a bound for each parameter be specified. These bounds are simply an extension of the initial parameter set in the upper and lower directions. The population is then seeded with individuals who are variations of the initial parameter set. We used an elitist selection policy, where the best individual is continually pasted from generation to generation unaltered. As the simulation progresses, the GA continually breeds new solutions which minimize the sum-of-squared error until the maximum number of generations is reached. The GA implementation we used was designed for maximization, so the fitness value (SSE) was made negative.

It should be noted that the ANFIS predictor only allows for the prediction of one output at a time. So, three separate predictors were used to predict the three object parameters. Each predictor followed the same procedure for initialization and optimization with the appropriate simulation model.

7.1 Training Data Selection

For the simulations, the first half of the data points was used for training the ANFIS, while the second half was used for validating the identified model. This particular selection method was done for the sake of simplicity and resulted in acceptable predictions. It should be noted that typically the training set should contain information

representative of the dynamics of interest. Inappropriate data selection could produce less than reasonable results. However, for the data sets chosen, the first half of the data was sufficient and resulted in reasonable prediction.

Simulation indicated that for the range prediction, knowledge about the current object only was sufficient for perfect range prediction. However, to ensure that the ANFIS captured the dynamics of the training set for the height and midpoint predictions, the data was selected in the following format:

$$[x(t-5), x(t-4), \ldots, x(t);\ y(t)],$$

where $x(t-5), \ldots, x(t-1)$ and $x(t)$ are the past five and current inputs respectively, and $y(t)$ is the desired output. The training data matrices for each ANFIS only contains information about the parameter being predicted.

7.2 Time Averaged Prediction Error

In following sections, the ANFIS is used to make future object predictions based on input object data. Though a SSE defintion is used for training the ANFIS, this is inappropriate for validating the model. To gauge the performance of the predictor, we define an error measure which assumes that the object error, that is, the difference between the predicted object and the actual object, follows a uniform distribution.

As shown in Figure 7.2, at each time interval, each object has associated parameters x^h and x^l which define the height and width of the object. Assuming that both the actual and predicted objects follow a uniform distribution, the probability density

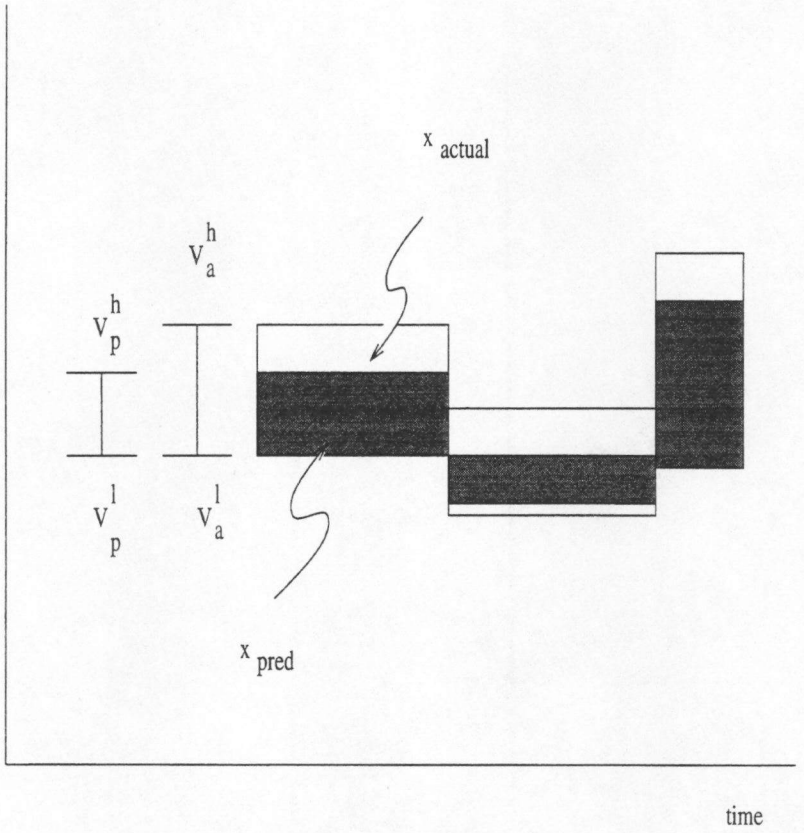


Figure 7.2: Actual and Predicted Object Sequence.

functions are given by

$$p(x_{act}) = \begin{cases} \frac{1}{v^h_{act} - v^l_{act}} & x_{act} \in [v^l_{act}, v^h_{act}] \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

and

$$p(x_{pred}) = \begin{cases} \frac{1}{v^h_{pred} - v^l_{pred}} & x_{pred} \in [v^l_{pred}, v^h_{pred}] \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

The plots of the probability densities functions for the actual and predicted signals are shown in Figures 7.3 and 7.4. It is assumed that both x^h_{act} and x^l_{pred} are independent. Thus, the joint distribution is product of the distributions for x^h_{act} and x^l_{pred} or

$$p(x_{act}, x_{pred}) = p(x_{act})p(x_{pred}) = \begin{cases} \frac{1}{v^h_a - v^l_a}\frac{1}{v^h_p - v^l_p} & [v^l_a, v^h_a] \times [v^l_p, v^h_p] \\ 0 & \text{otherwise} \end{cases} \tag{7.3}$$

The 2-norm prediction error is given by $(x_{act} - x_{pred})^2$. It follows that the 2-norm error is

$$E[(x_{act} - x_{pred})^2] = \int_t \int_{x_{act}} \int_{x_{pred}} (x_{act} - x_{pred})^2 p(x_{act}, x_{pred}) dx_{act} dx_{pred} \tag{7.4}$$

$$= \int_t \int_{v^l_p}^{v^h_p} \int_{v^l_t}^{v^h_t} (x_{act} - x_{pred})^2 p(x_{act}, x_{pred}) dx_{act} dx_{pred} \tag{7.5}$$

$$\tag{7.6}$$

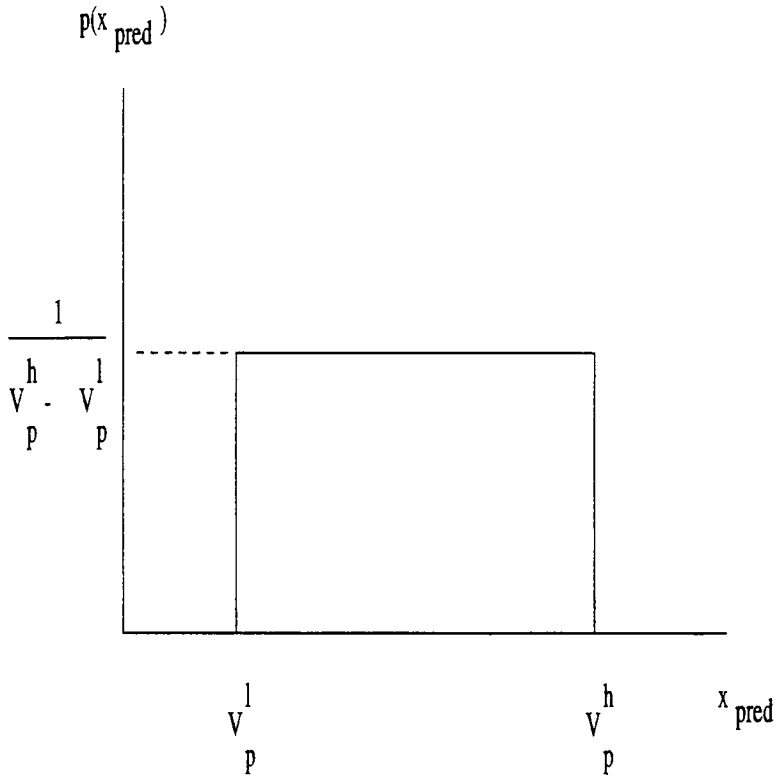


Figure 7.3: Probability Density Function for x_{pred}

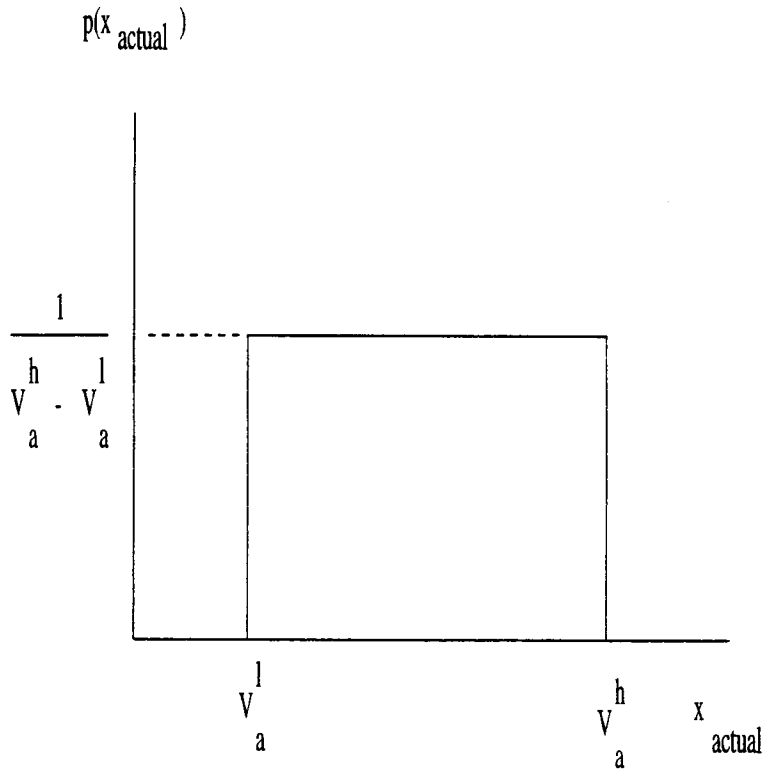


Figure 7.4: Probability Density Function for x_{actual}

$$E[(x_{act}-x_{pred})^2] = \left((t_f-t_o)\frac{1}{v_t^h-v_t^l}\right)\left(\frac{1}{v_p^h-v_p^l}\right)\left\{\left(\frac{(v_t^h)^3-(v_t^l)^3}{3}\right)(v_p^h-v_p^l)+\left(\frac{(v_p^h)^3-(v_p^l)^3}{3}\right)(v_t^h-v_t^l)-\left(\frac{(v_t^h)^2-(v_t^l)^2}{2}\right)((v_p^h)^2-(v_p^l)^2)\right\} \quad (7.7)$$

Equation 7.6 simplifies to 7.7. The time averaged error can now be represented as

$$E=\frac{1}{T}\int_t\left(\int_{x_{true}}\int_{x_{pred}}(x_{act}-x_{pred})^2 p(x_{act},x_{pred})dx_{pred}dx_{act}\right)dt \quad (7.8)$$

where T is the duration of the predicted object sequence.

Since both the actual object and its corresponding predicted object are on the same interval for each interval of the sequence, we can integrate each actual object and its corresponding predicted object over the same interval of time.

It should be noted that the error measure above is used only for comparison with other research associated with this project.

7.3 Simulation Results

In this section, the ANFIS is used for predicting a sequence of output objects given input object data. Using the training data format above, simulations with various object-oriented signals were done. The number of rule generated is $(nmf)^{ninput}$ where nmf is the number of fuzzy rules and $ninput$ is the number of inputs. For several of the simulations, the genetic algorithm reduced the SSE by a small amount, while for

other simulations, there was a drastic reduction in the SSE. In all cases, the initial results are indeed intuitively reasonable, and results are produced within a reasonable amount of time.

7.4 Object-Oriented Signal Modeling

We begin our simulations by first demonstrating the ability of the ANFIS to model an object-oriented, non-periodic signal. We use the same training data format as given above. Sixty-four fuzzy rules are used to model the signal and 105 data points are used for training.

Figure 7.5 shows the raw signal to be scrubbed and used for training and prediction. The ANFIS is trained only using the first half of the signal.

Figures 7.6 and 7.7 show the initial membership functions. After training, the change in the height SSE was noticable as shown in Figure 7.8. There is also a noticable reduction in the midpoint SSE as shown in Figure 7.9. The final membership functions are shown in Figures 7.10 and 7.11. Figure 7.12 shows the result.

After relatively little training, the ANFIS predicts the second half of the signal with a fair degree of accuracy. The RMSE is 8.1547.

We now consider a second signal which the ANFIS reasonably models. As with the first demonstration, 64 fuzzy rules are generated and 205 data points are used for training and for validating the model found.

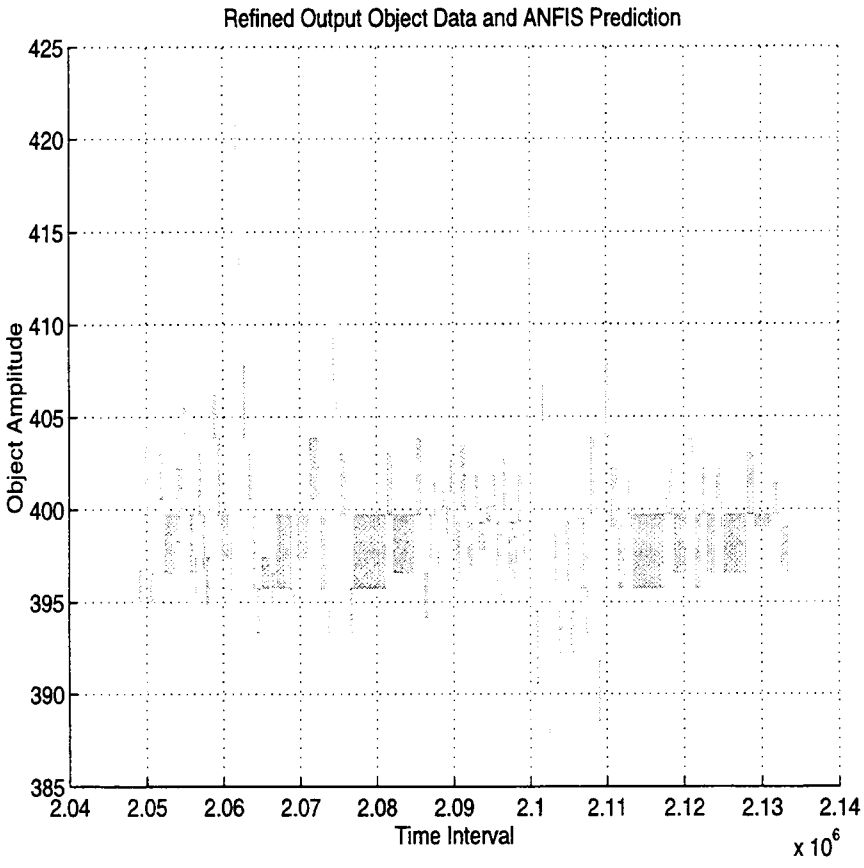


Figure 7.5: Raw Signal

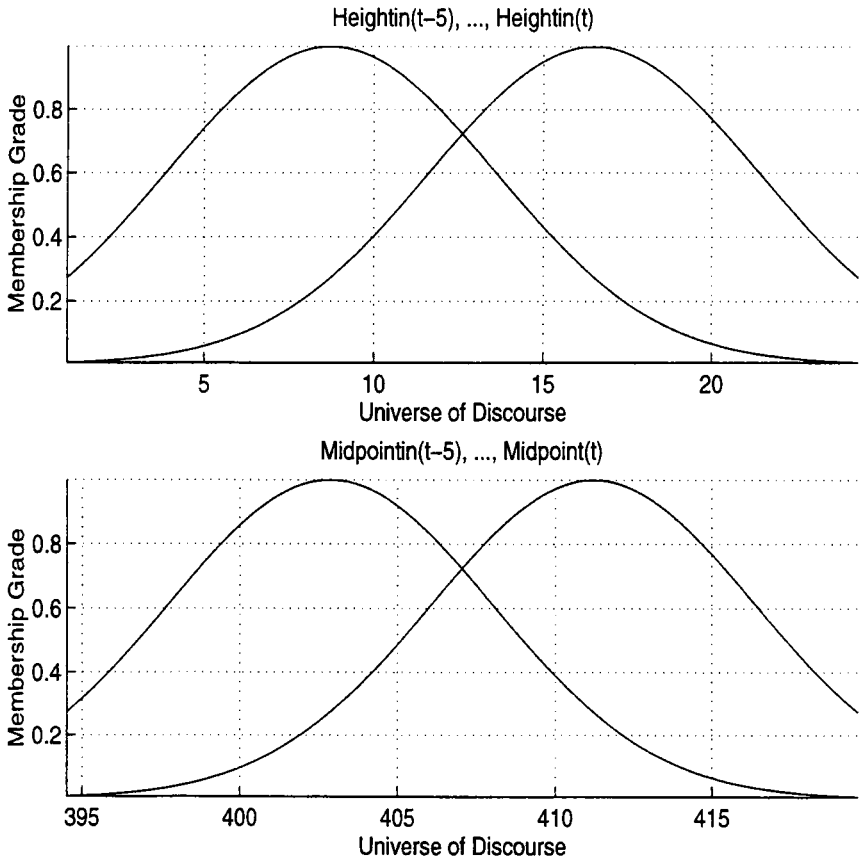


Figure 7.6: Initial Membership Functions for the Height and Midpoint

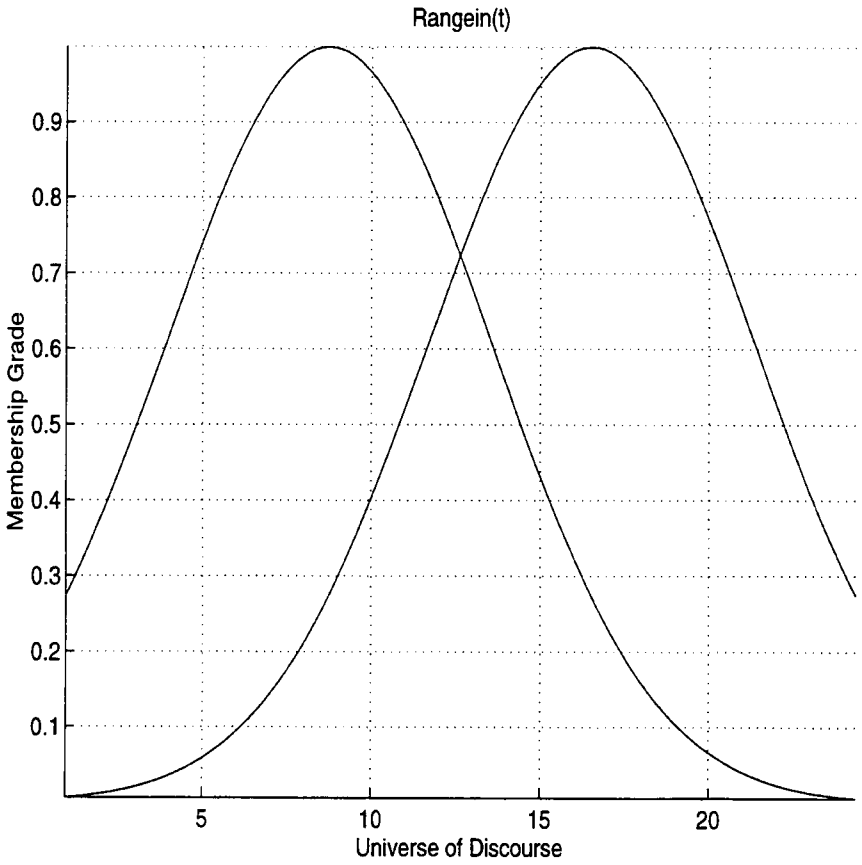


Figure 7.7: Initial Membership Functions for the Range

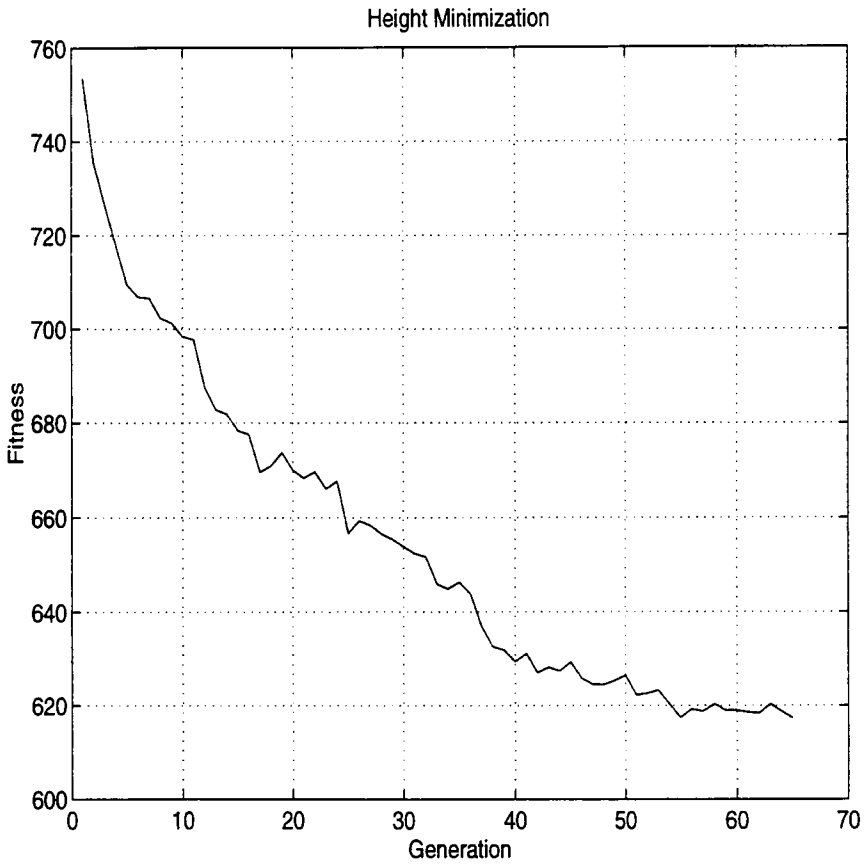


Figure 7.8: Height Cost Function Minimization

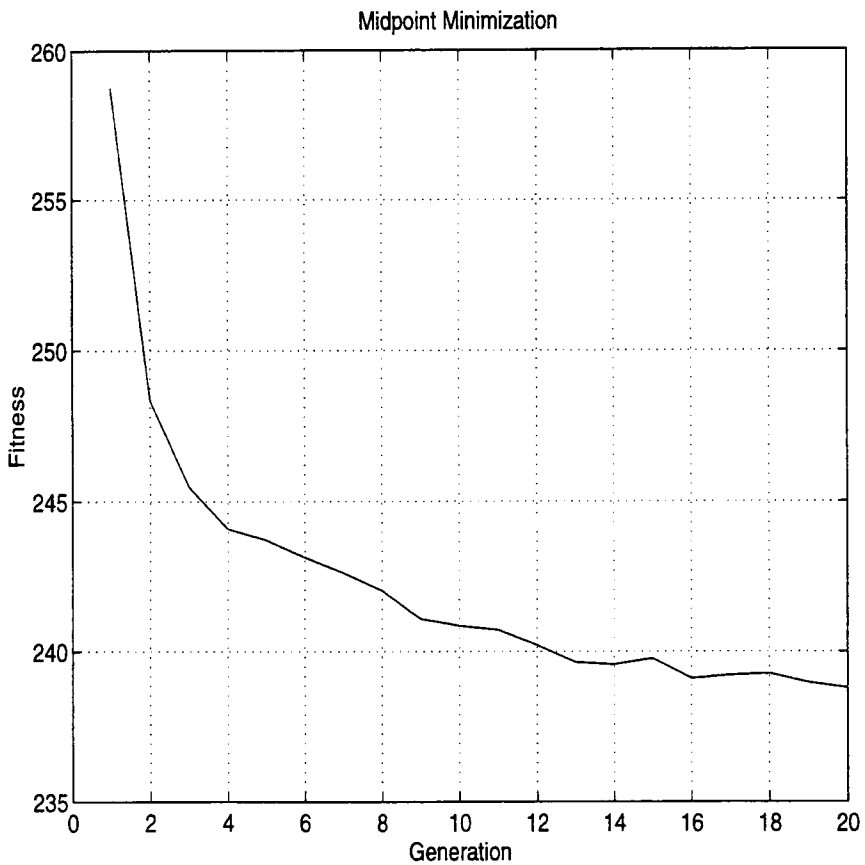


Figure 7.9: Midpoint Cost Function Minimization

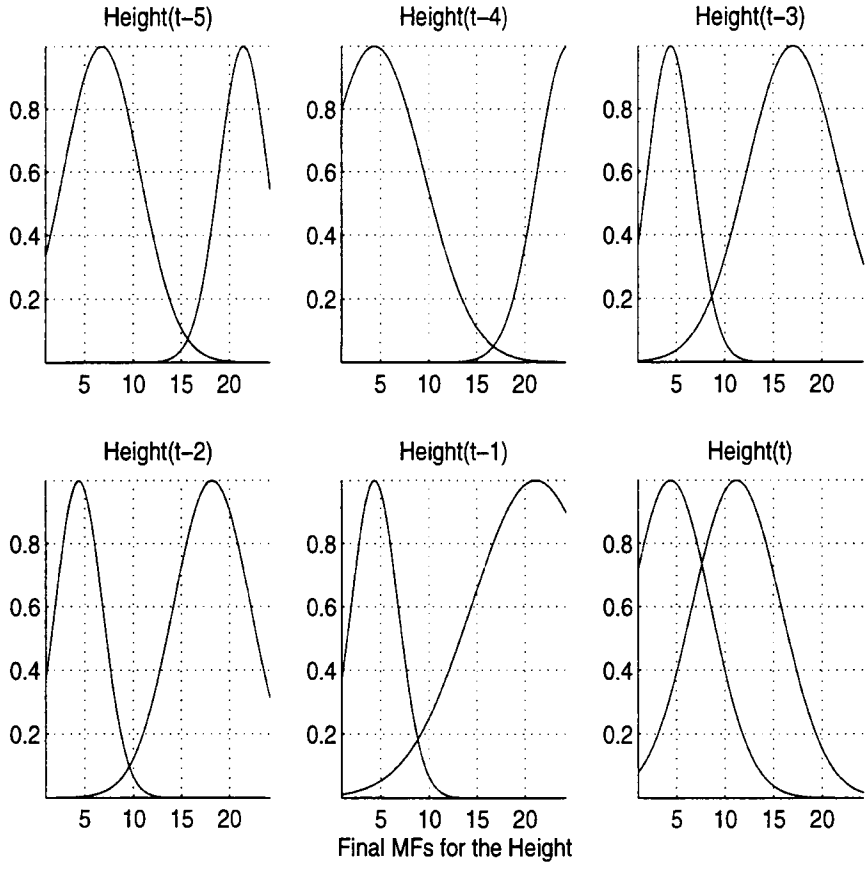


Figure 7.10: Final Membership Functions for the Height

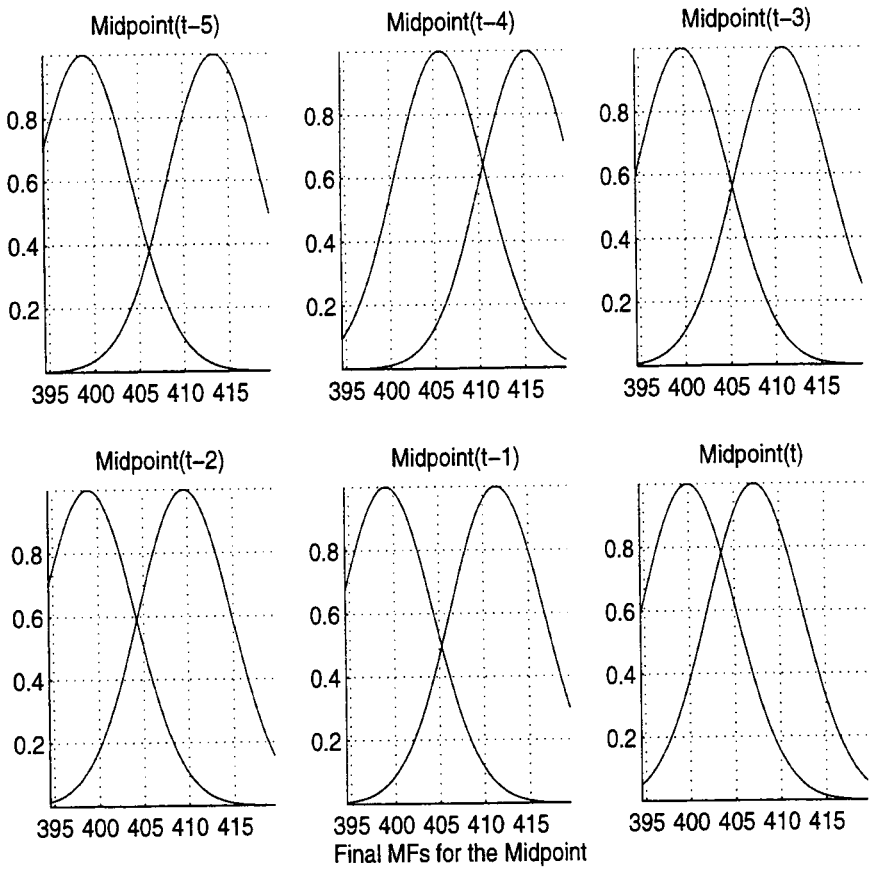


Figure 7.11: Final Membership Functions for the Height

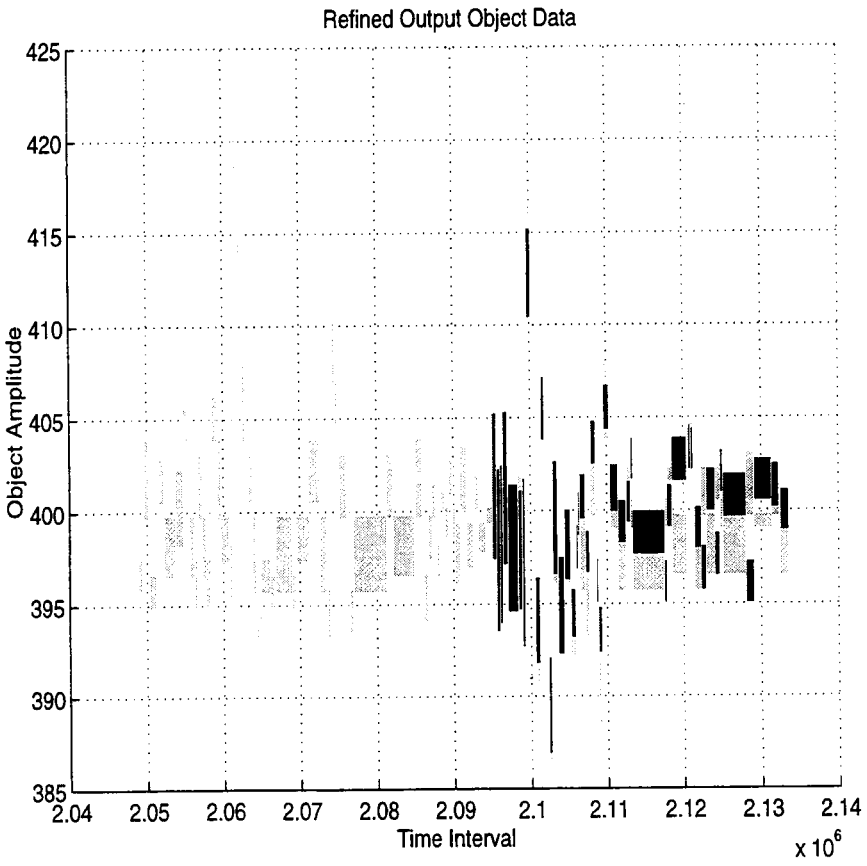


Figure 7.12: ANFIS Prediction

Again, only the first half of the signal is used for training. The signal we wish to model is shown in Figure 7.13. The initial membership functions are shown in Figures 7.14 and 7.15. Figure 7.16 shows the midpoint cost function minimizations. The height cost function minimization is shown in Figure 7.17. Notice that even though the ANFIS has no explicit information about the predicted half of the signal, it is still able to reasonably infer what the second half of the signal should be. This is proof that the ANFIS-generated rulebase has captured the dynamics of the signal. The final membership functions are shown in Figures 7.18 and 7.19. After a brief training session, the prediction results are shown in Figure 7.20. The RMSE is 8.254.

We now demonstrate the potential of the ANFIS for relationship extraction for two object-oriented signals.

7.4.1 Relationship Extraction: Case I

Sets of Training Data: 425

Sets of Validation Data: 425

Linear Parameters: 448

Nonlinear Parameters: 36

Total Number of Parameters: 484

Total Number of Rules: 64

RMSE: 9.8568

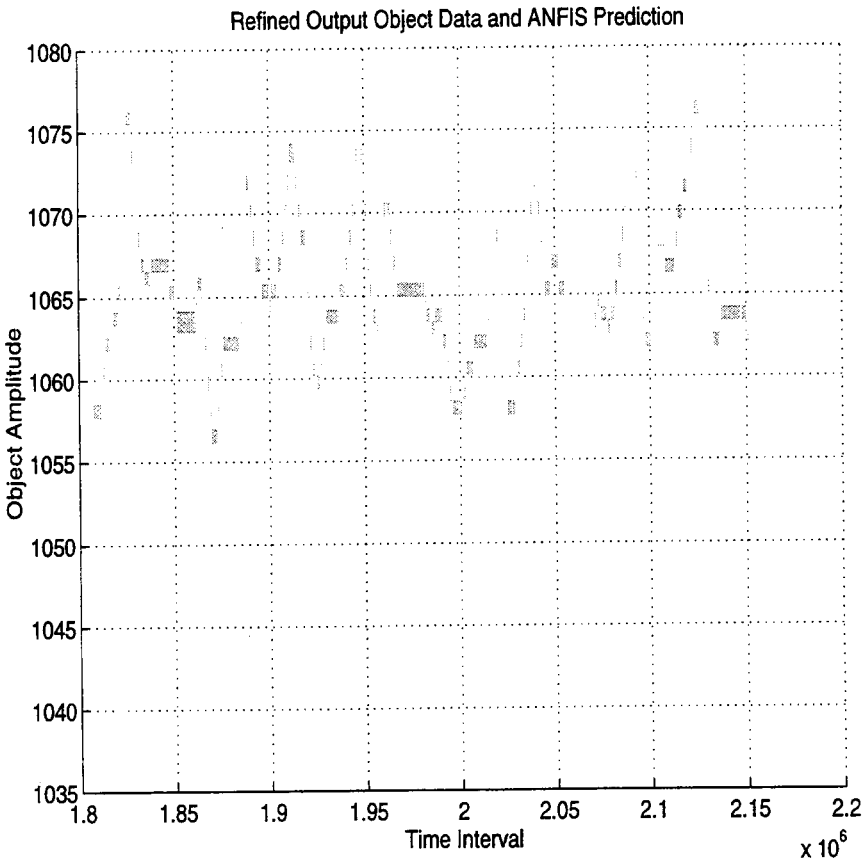


Figure 7.13: Object-Oriented Signal

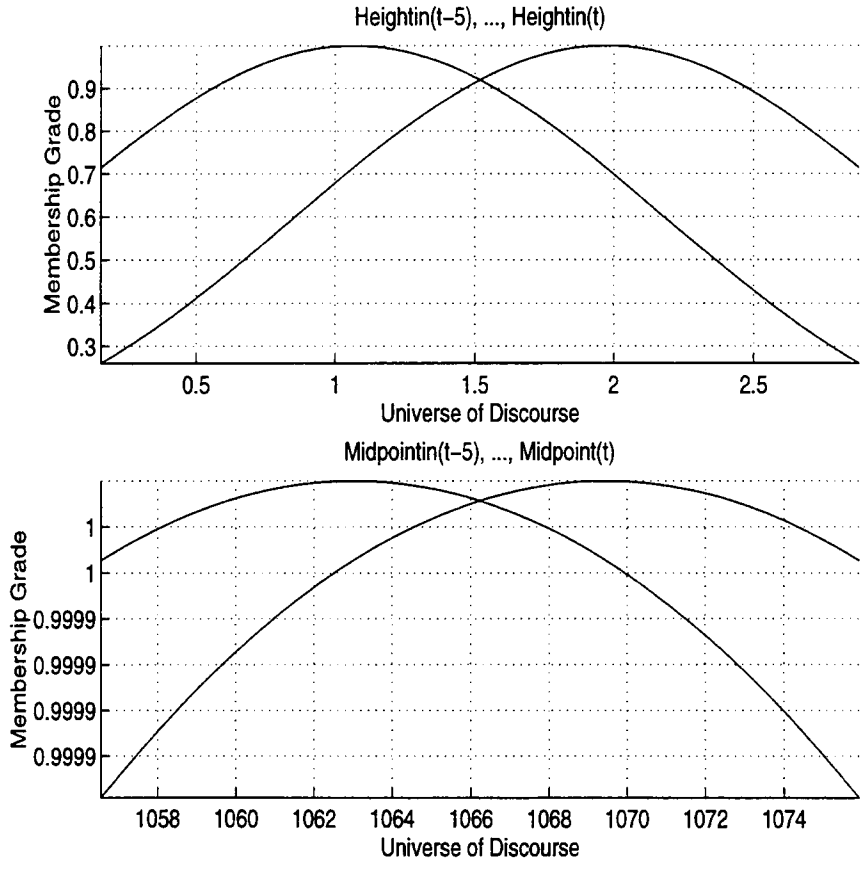


Figure 7.14: Initial Membership Functions for the Height and Midpoint

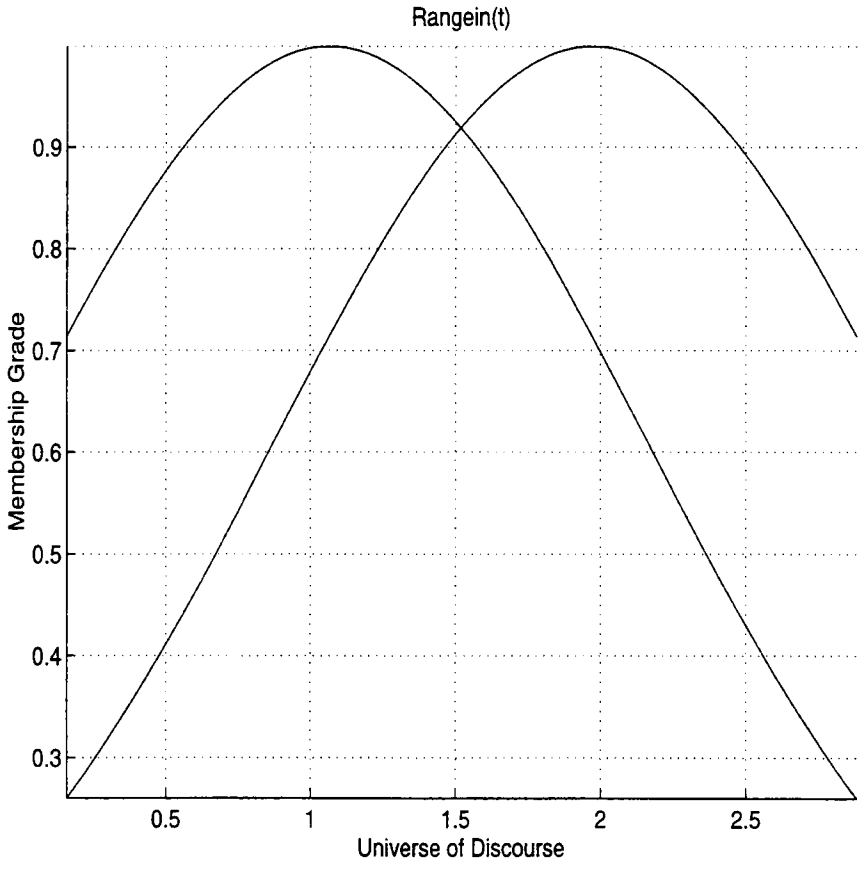


Figure 7.15: Initial Membership Functions for the Range

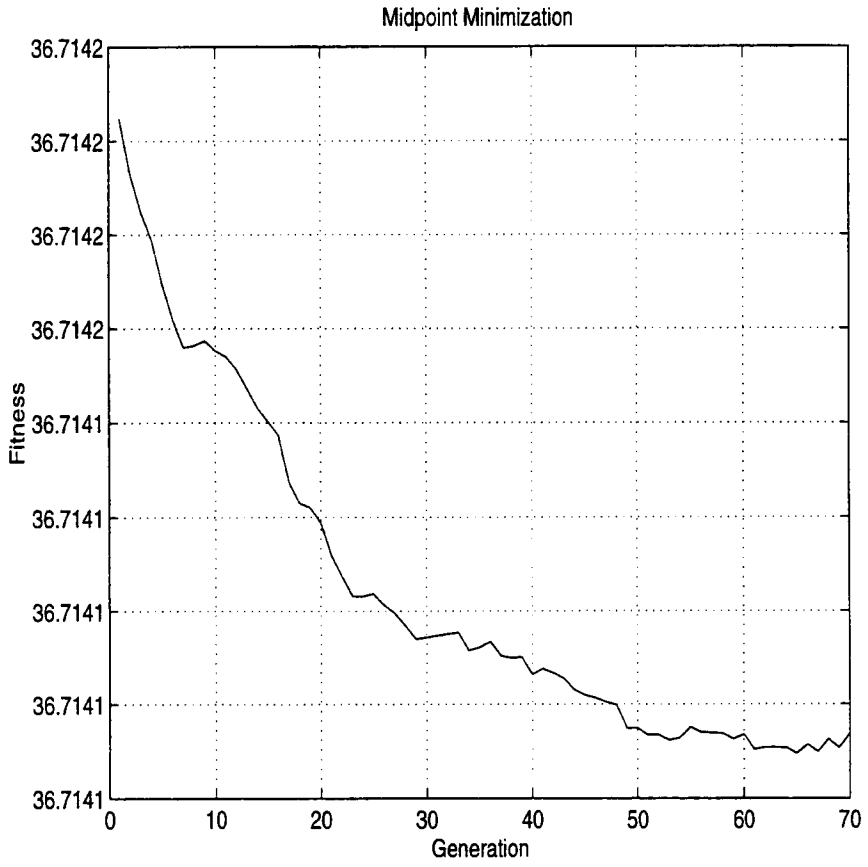


Figure 7.16: Midpoint Cost Function Minimization

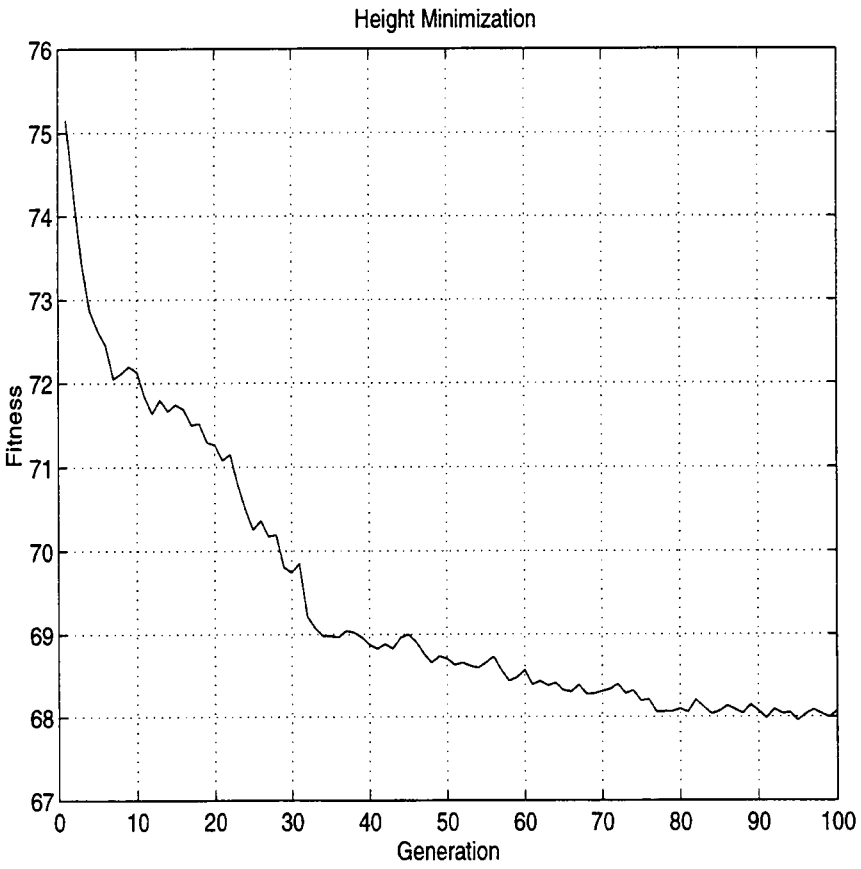


Figure 7.17: Height Cost Function Minimization

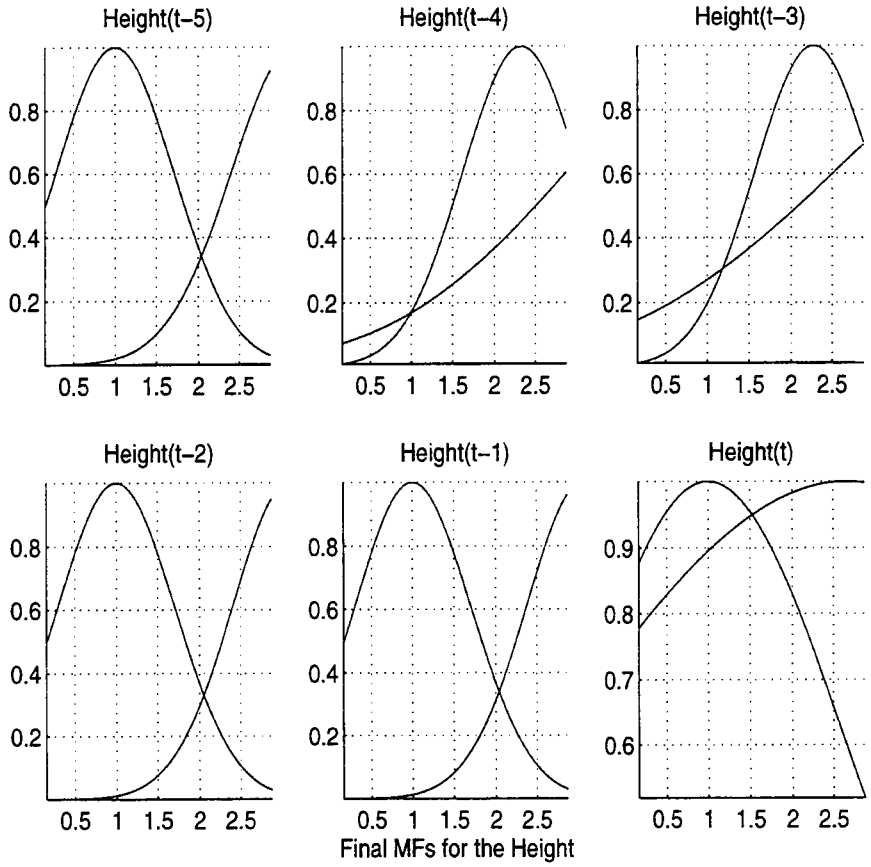


Figure 7.18: Final Membership Functions for the Height

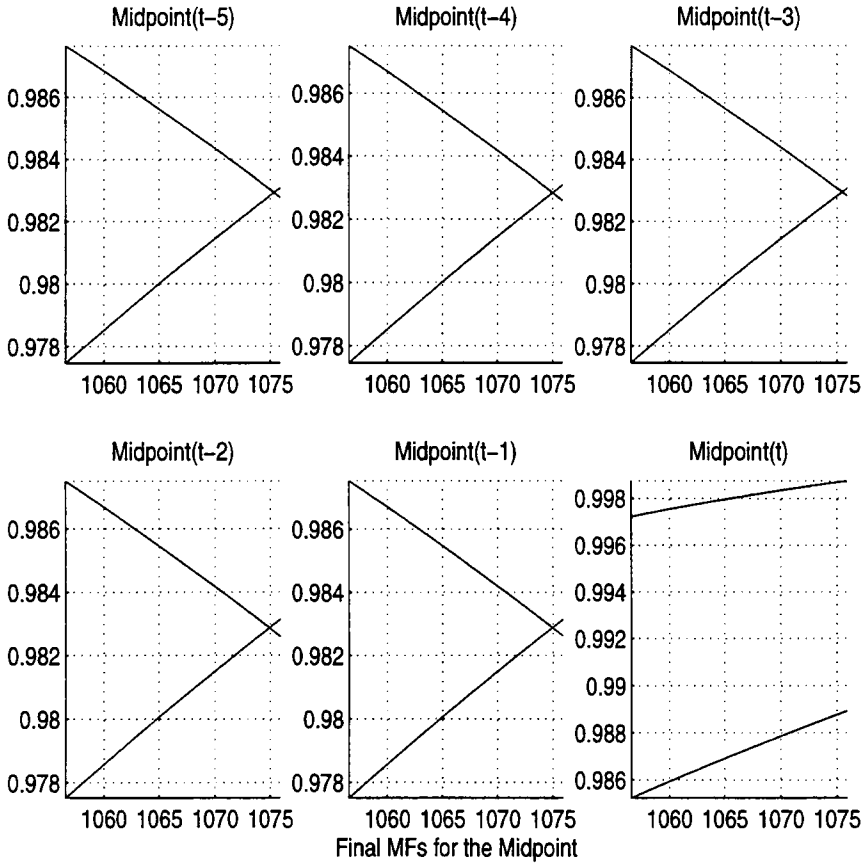


Figure 7.19: Final Membership Functions for the Midpoint

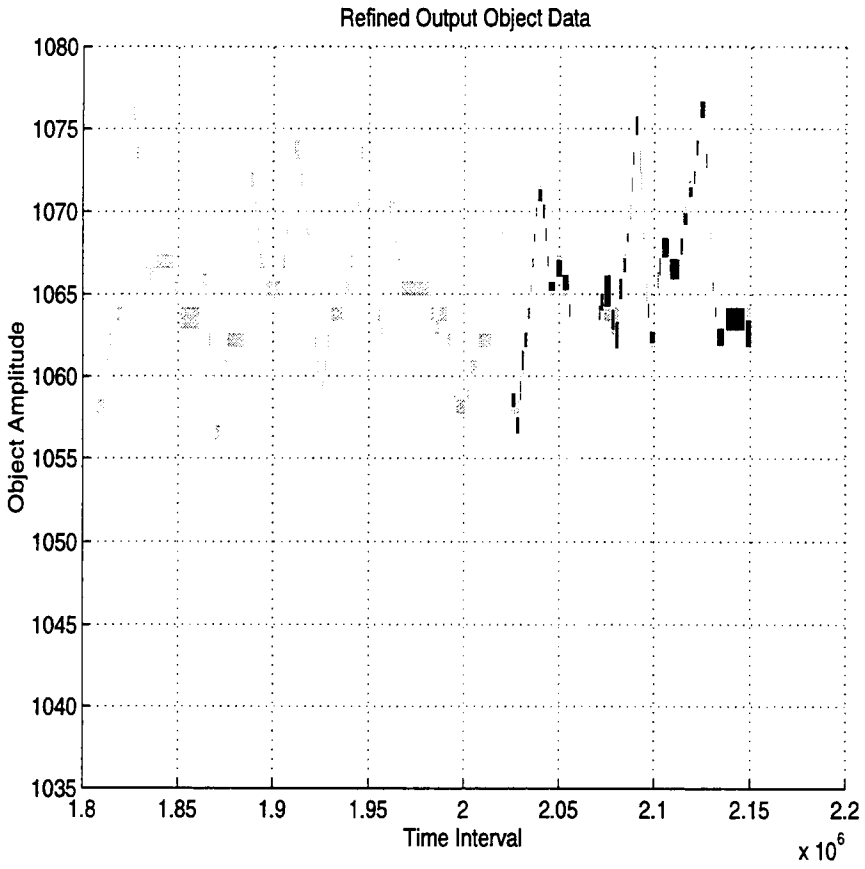


Figure 7.20: ANFIS Prediction

Figure 7.21 shows the unrefined object-oriented signals while Figures 7.22 shows the refined input and output signals.

Figure 7.23 and Figure 7.24 show the initial network membership functions. Figure 7.25 shows the progress of the height parameter minimization. For this particular set of signals, the genetic algorithm significantly reduces the midpoint SSE as shown in Figure 7.26. Figure 7.27 shows the final membership functions for the height. Figure 7.28 shows the final memberhsip functions for the midpoint. Figure 7.29 shows the model prediction.

The ANFIS model seems to have had more difficulty predicting the future sequence of output objects for this particular simulation. It is not fully understood why the fuzzy rulebase more easily predicts some signals than others and why some relationships are more easily identified than others. We only used the first half of the object signal sequences for training; so perhaps essential dynamics were not contained in the training set and as a result, were not captured by the ANFIS and consequently, not reflected in the model prediction. In addition, the amount of training data may be insufficient. Generally, modeling methods of this kind require more training points than what was available for this simulation. The input and output object sequences were arbitrarily selected. So, no more information about the signals is known other than what is used to develop the fuzzy model.

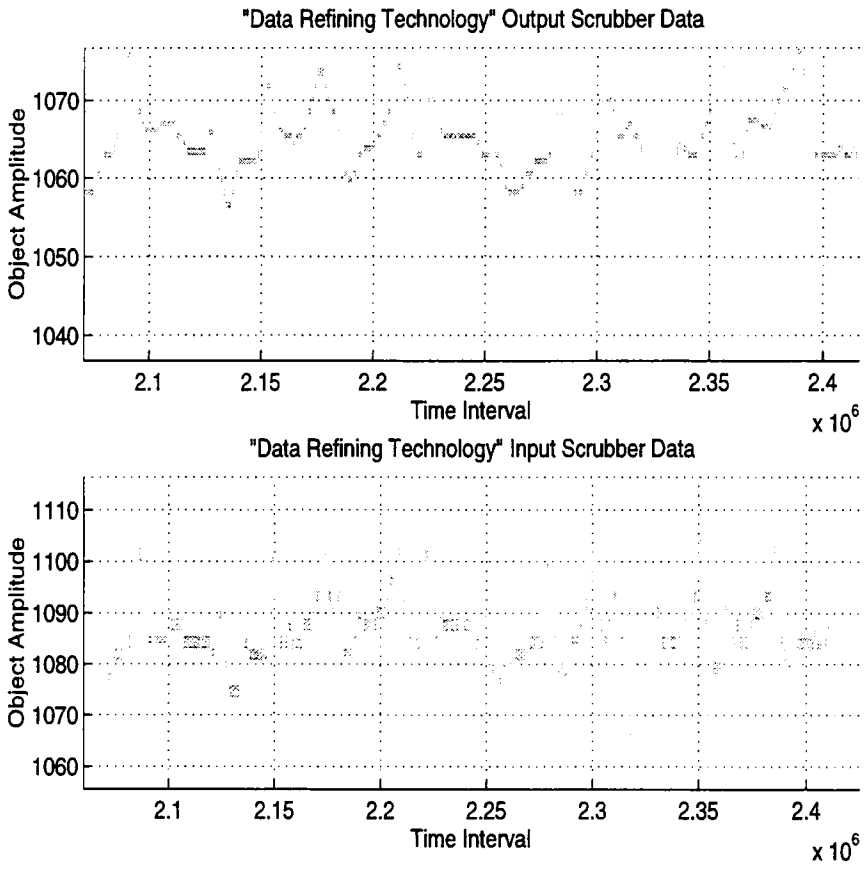


Figure 7.21: Unrefined Object-Oriented Signals for Case I.

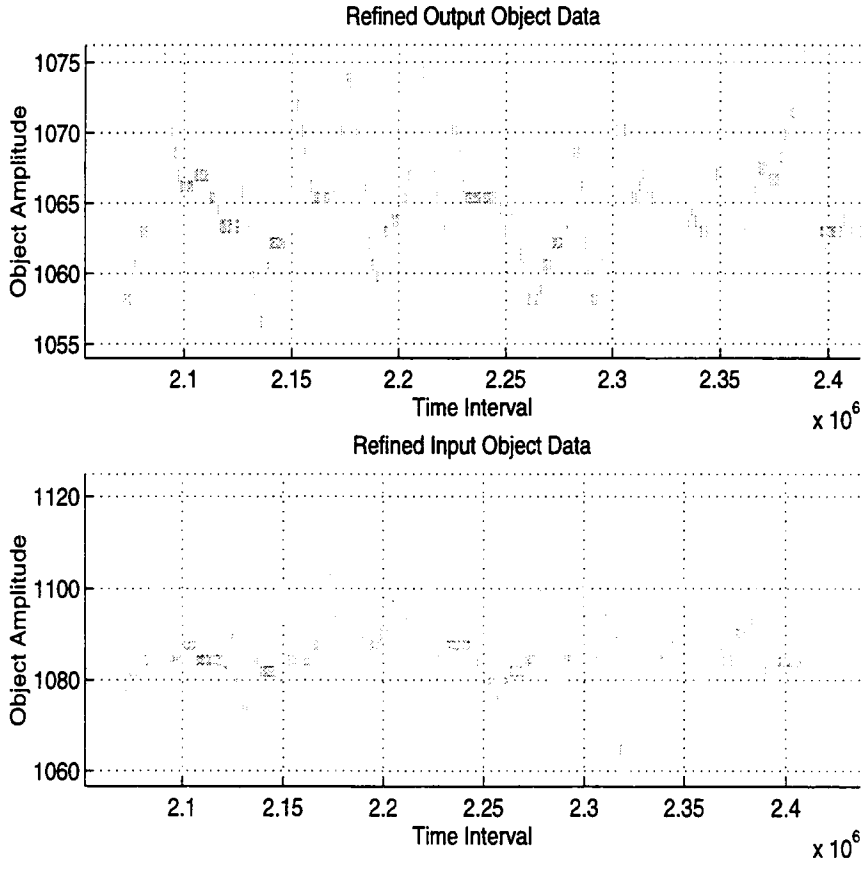


Figure 7.22: Refined Object-Oriented Signals for Case I.

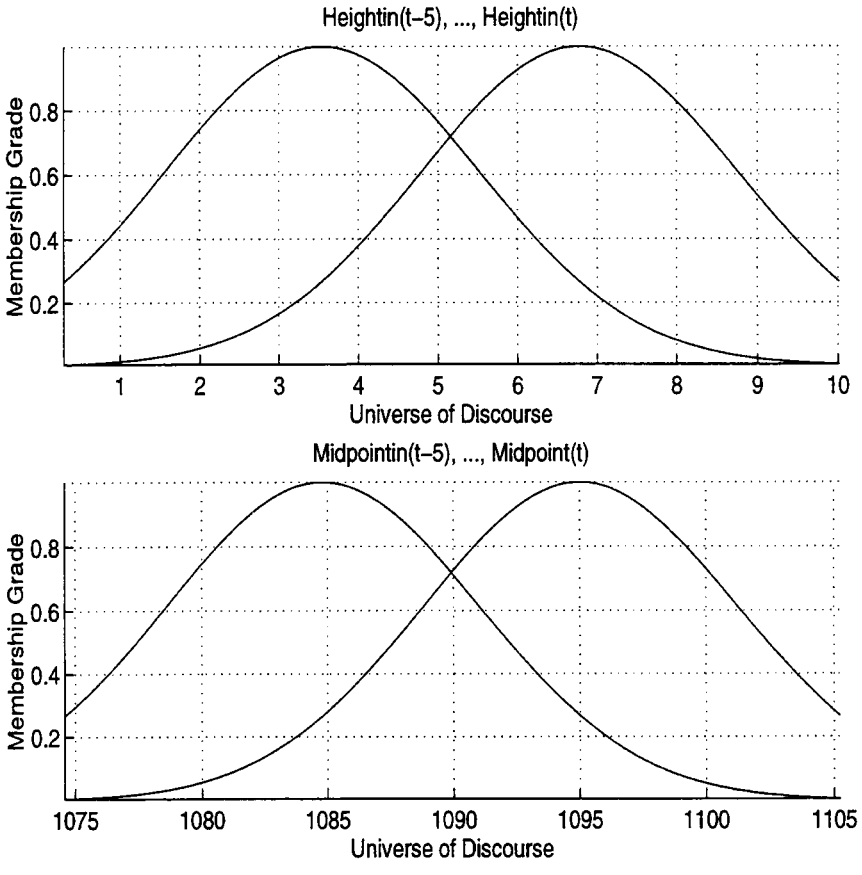


Figure 7.23: Initial Membership Functions for the Height and Midpoint.

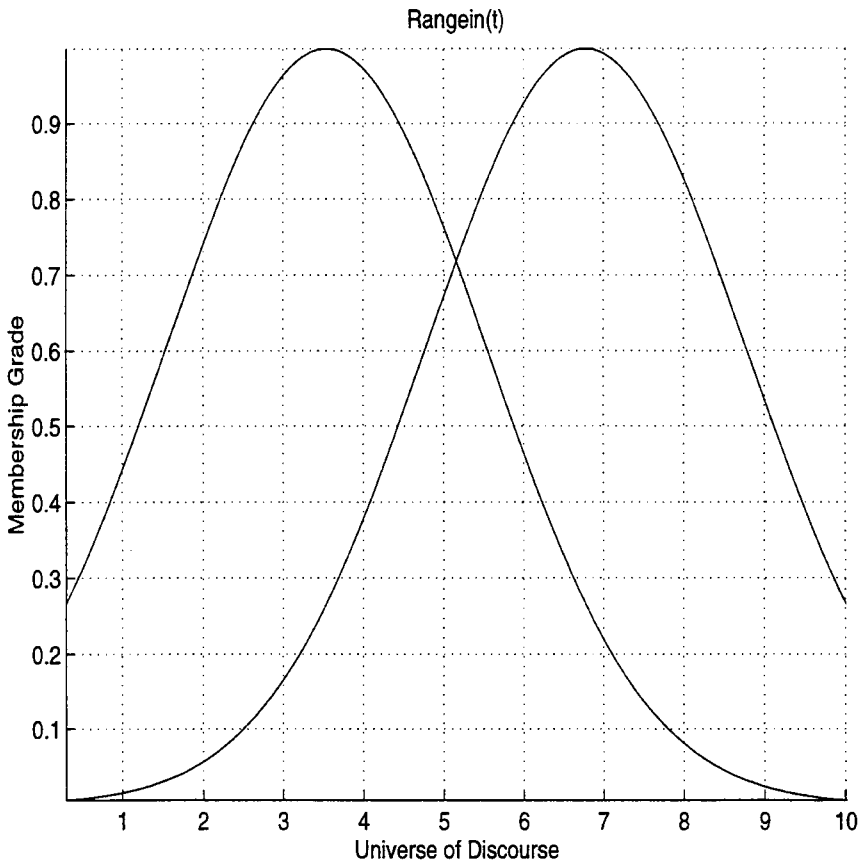


Figure 7.24: Initial Membership Functions for the Range

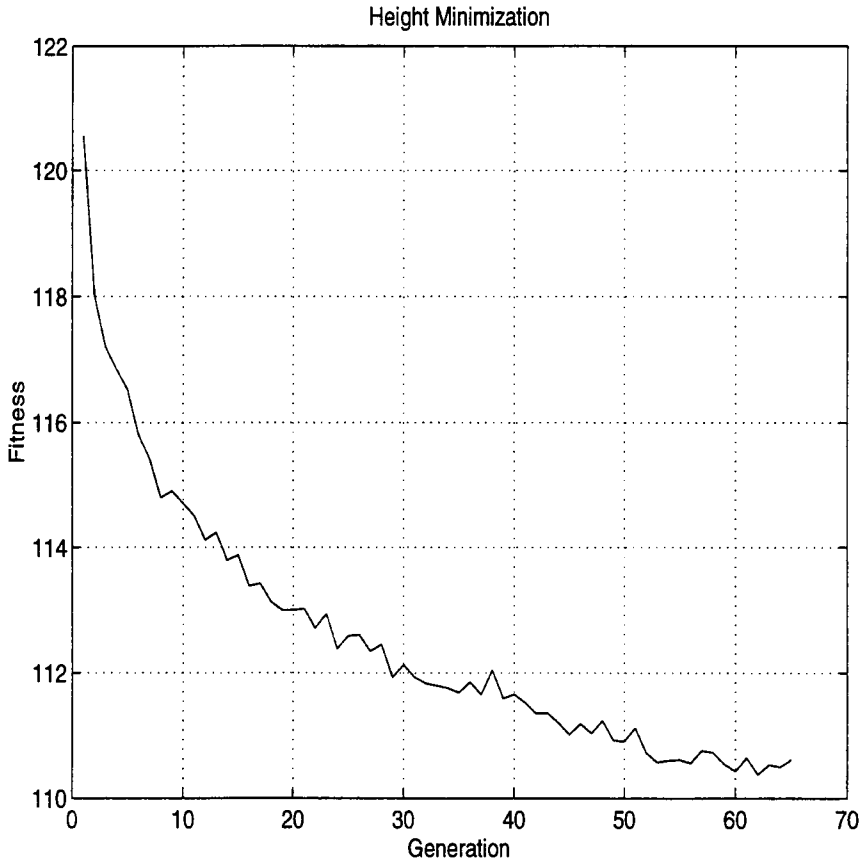


Figure 7.25: Height Cost Function Minimization

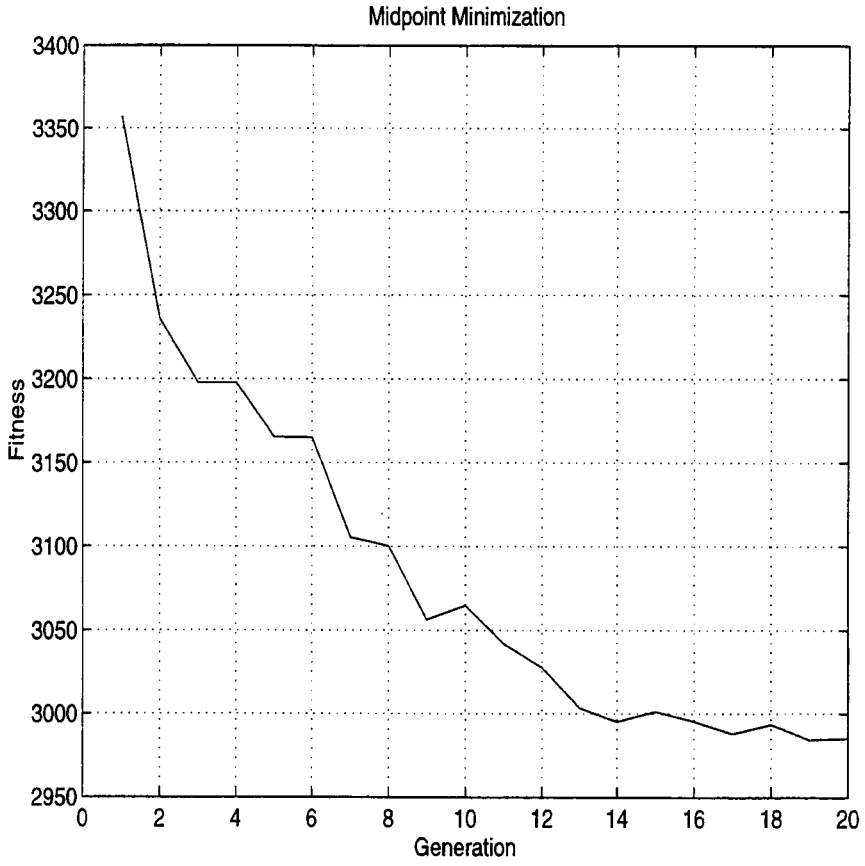


Figure 7.26: Midpoint Cost Function Minimization

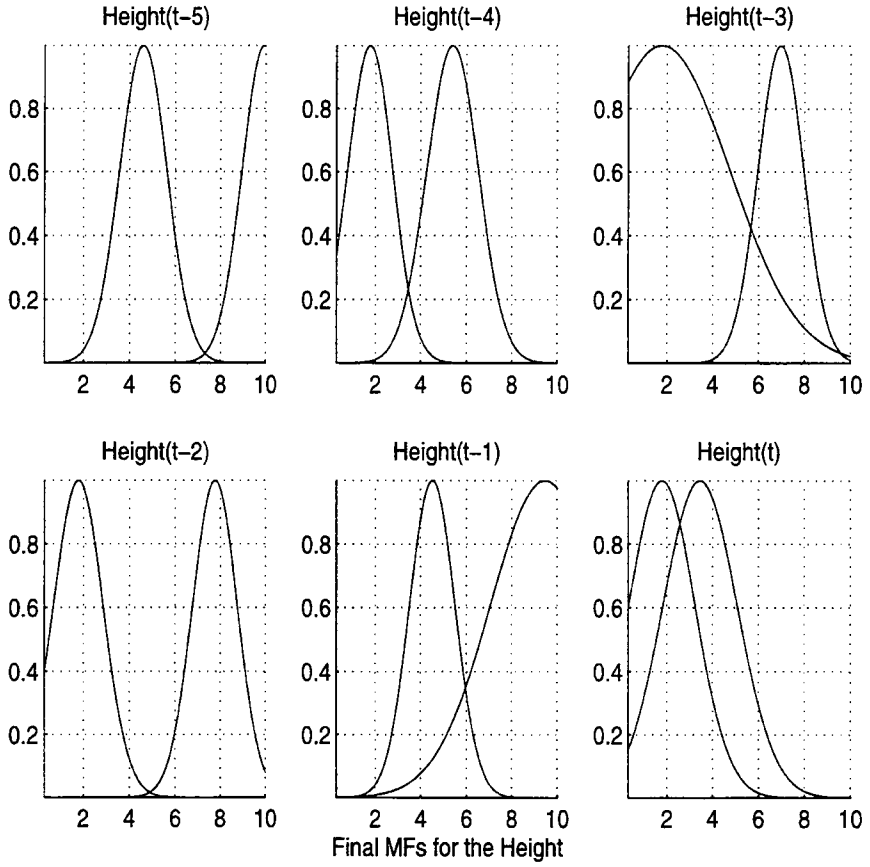


Figure 7.27: Final Height Membership Functions

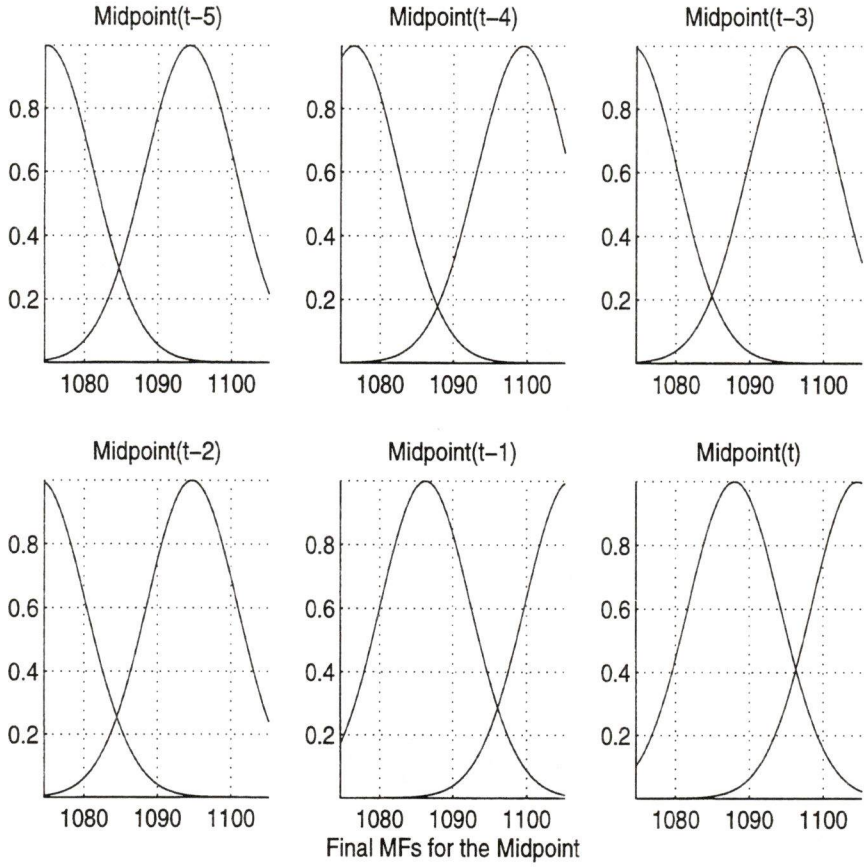


Figure 7.28: Final Midpoint Membership Functions

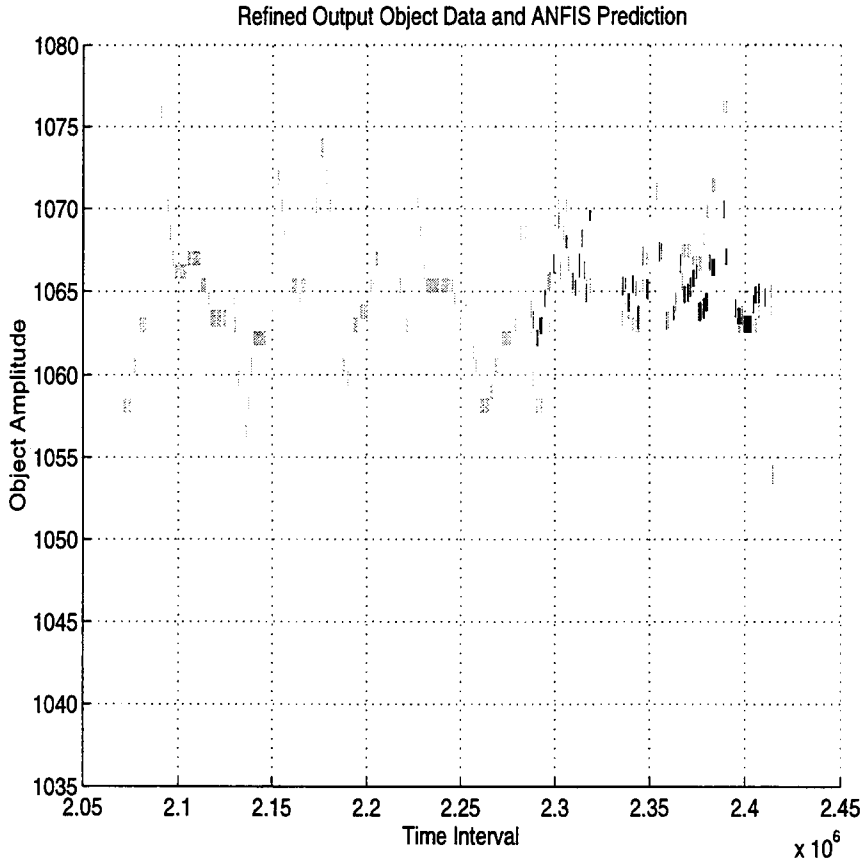


Figure 7.29: ANFIS Prediction

7.4.2 Relationship Extraction: Case II

Sets of Training Data: 195

Sets of Validation Data: 195

Linear Parameters: 448

Nonlinear Parameters: 36

Total Number of Parameters: 484

Total Number of Rules: 64

RMSE: 6.3872

Figure 7.30 shows the unrefined object-oriented signals while Figures 7.31 shows the refined input and output signals.

Figure 7.32 and Figure 7.33 show the initial network membership functions. For this set of signals, the genetic algorithm significantly reduces the height and midpoint SSE as shown in Figure 7.34 and Figure 7.35. Figure 7.36 shows the final membership functions for the height. Figure 7.37 shows the final memberhsip functions for the midpoint. Figure 7.38 shows the model prediction.

It should also be noted that none of the simulations took more the 2.5 hours to run.

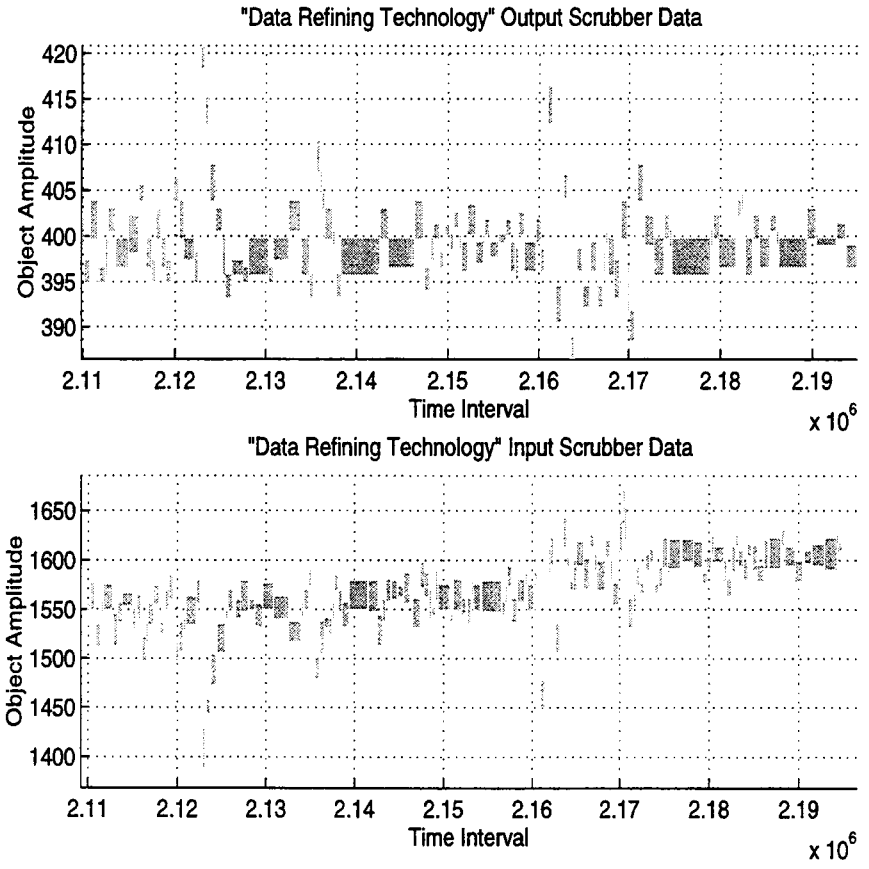


Figure 7.30: Unrefined Object-Oriented Signals for Case II.

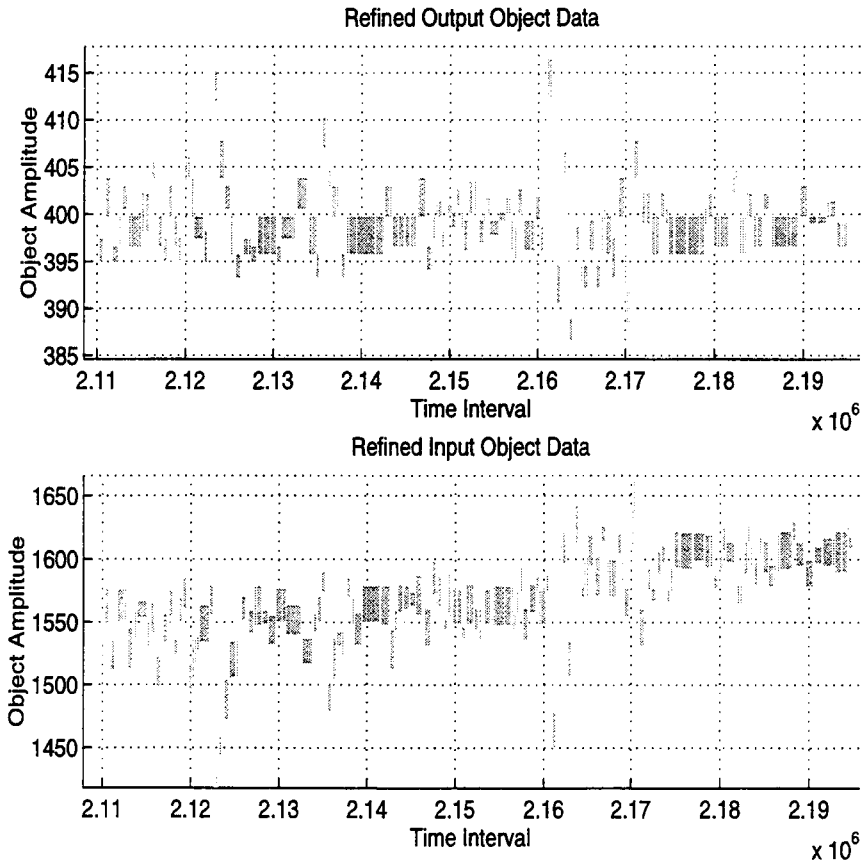


Figure 7.31: Refined Object-Oriented Signals for Case II.

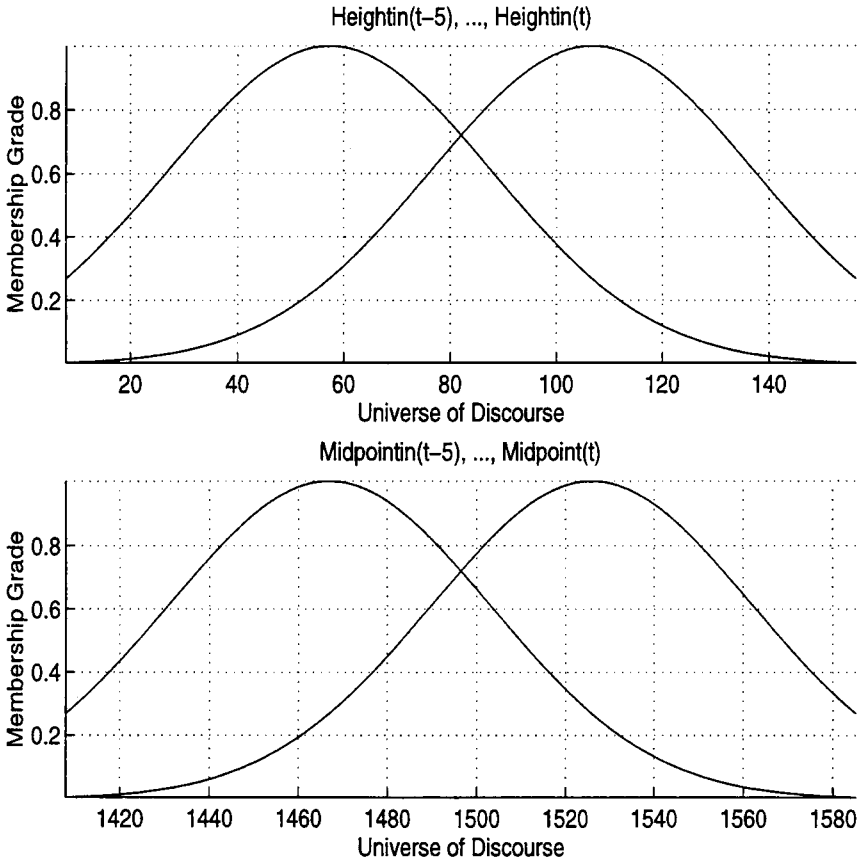


Figure 7.32: Initial Membership Functions for the Height and Midpoint

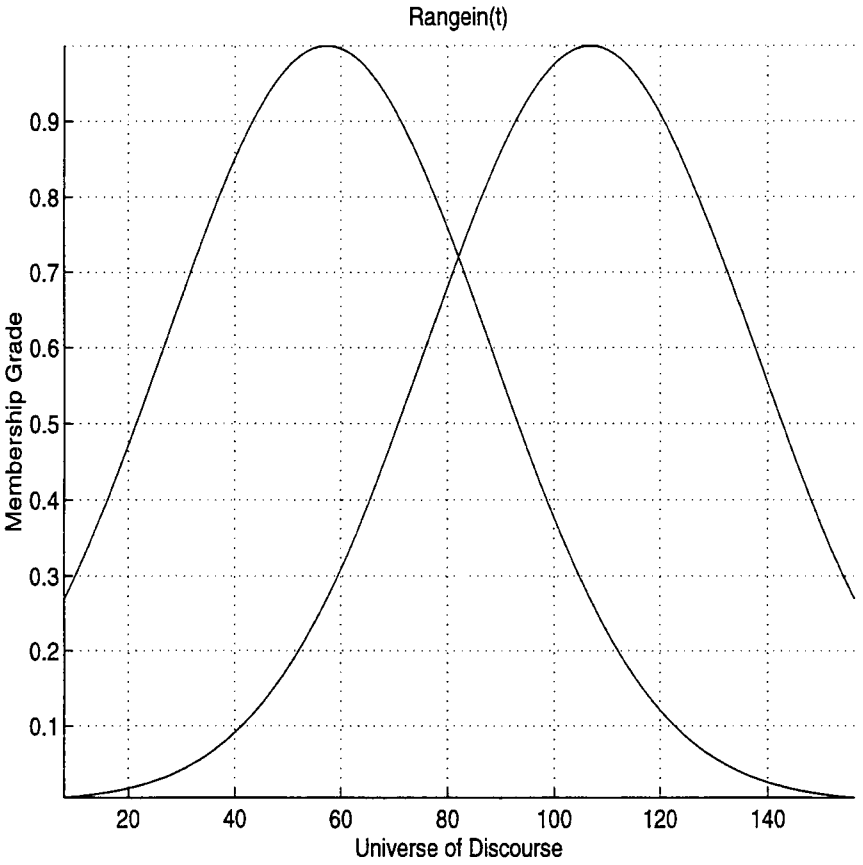


Figure 7.33: Initial Membership Functions for the Range

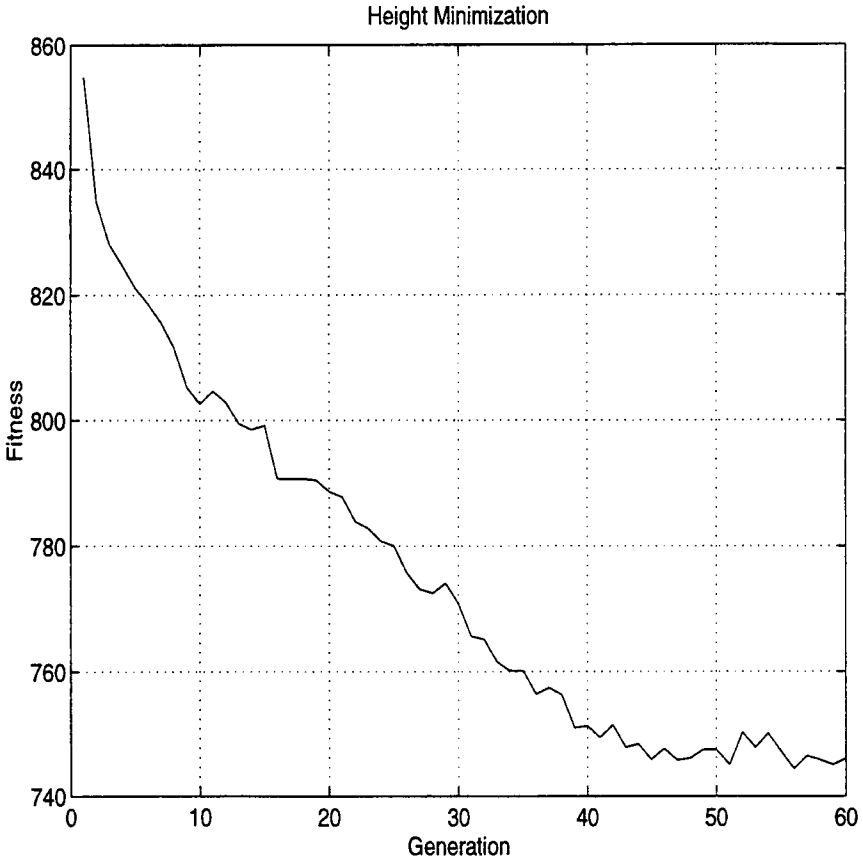


Figure 7.34: Height Cost Function Minimization

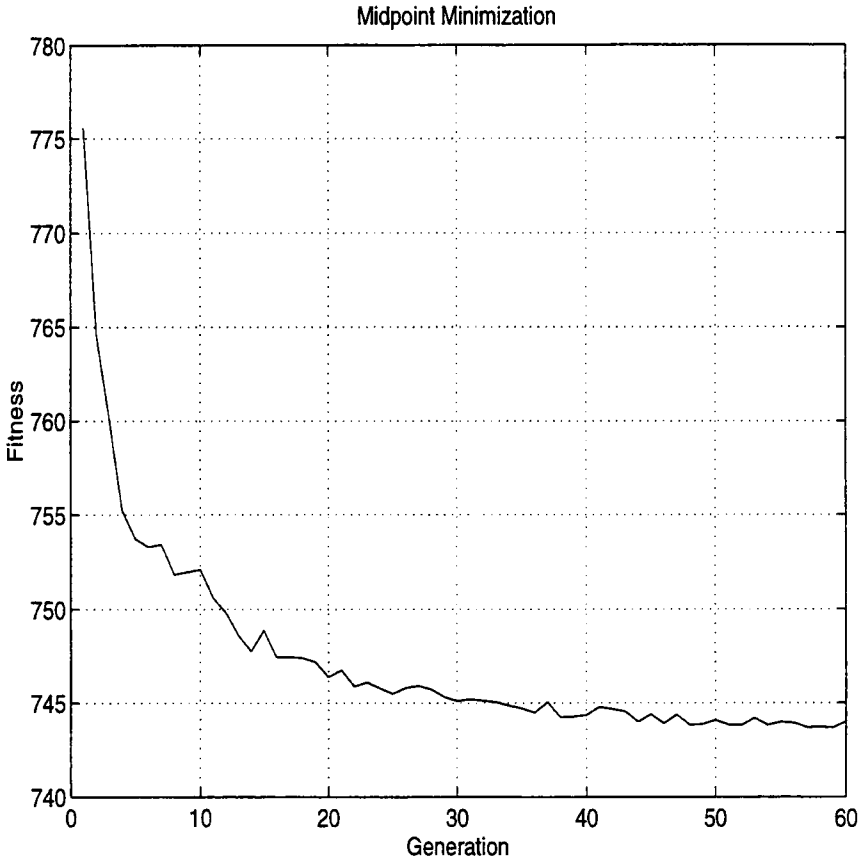


Figure 7.35: Midpoint Cost Function Minimization

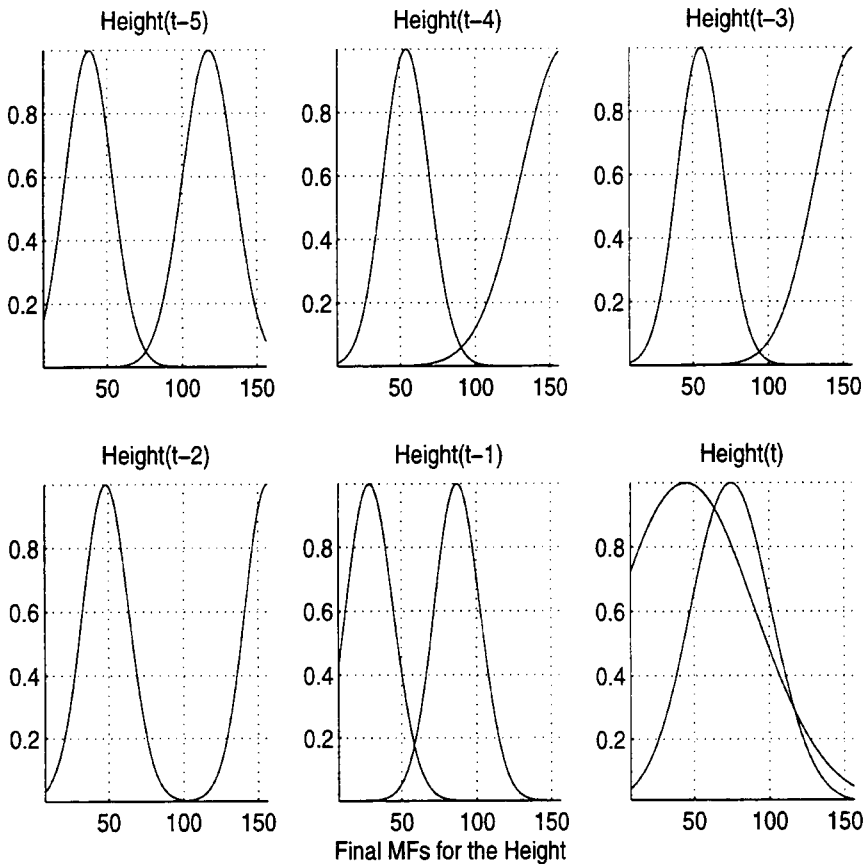


Figure 7.36: Final Height Membership Functions

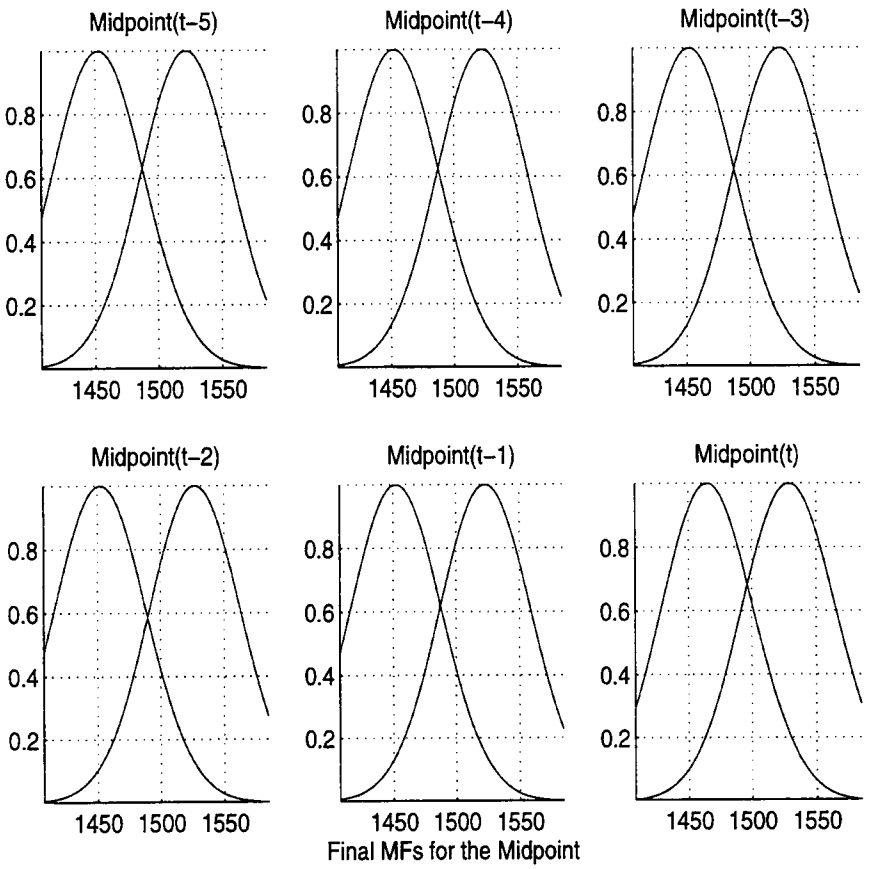


Figure 7.37: Final Midpoint Membership Functions

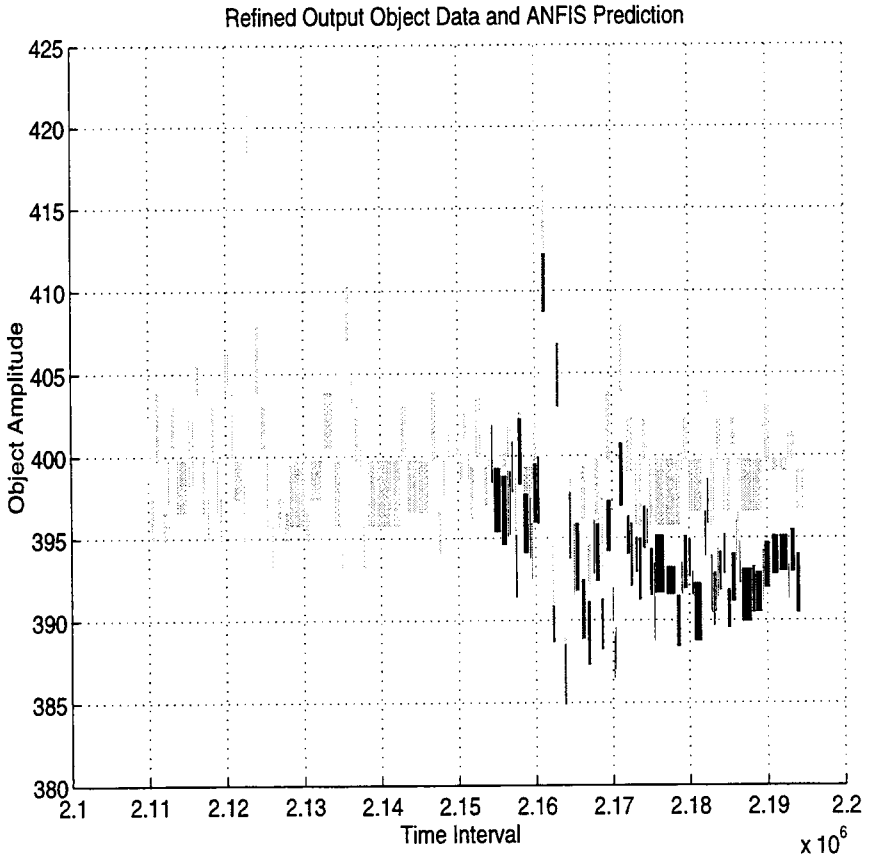


Figure 7.38: ANFIS Prediction

CHAPTER 8

Summary and Future Research Direction

This research explored using fuzzy rules for capturing the dynamics of object-oriented signals and for relationship extraction between pairs of signals. Sets of fuzzy rules are generated using an Adaptive Neuro-Fuzzy Inference System or ANFIS. The simulations demonstrate the potential of using fuzzy rules for the modeling and prediction of highly nonlinear, somewhat chaotic, object signals; signals which would otherwise be very difficult to model using traditional modeling techniques. The fuzzy rule sets generated by the ANFIS are able to make predictions based on a sequence of past input events; events which can easily be represented in a training data set. Such modeling and prediction capabilities would prove extremely useful for preventive maintenance purposes and quality control.

8.1 Discussion

The simulations presented show the potential of using fuzzy rules for modeling complex object-oriented signals; however, the limited amount of available data hampers the ability to draw firm conclusions. The ANFIS appears to be an effective way of generating sets of fuzzy rules which capture signal dynamics. The genetic algorithm is an effective means for optimizing relatively large parameter sets in a short amount of

time.

The genetic algorithm used in the simulations requires that bounds be specified for each parameter to be optimized. It was discovered that bounds which were to liberal caused the genetic algorithm to zero the membership function grades. Of course this produced a SSE of zero, but no dynamics were captured. Bounds which were to conservative restricted the search to only a section of the search space. Thus, the bounds were carefully set above and below the parameter values for each parameter. Given the complexity of the signals predicted, and the limited knowledge about the signal(s) of interest, the predictions were reasonable.

For the second signal modeling simulation, the genetic algorithm could not significantly reduce the SSE. For that particular run, there was a great degree of overlapping of the membership functions for the midpoint parameter. However, for the simulations with a lesser degree of overlap, a noticable reduction of the midpoint SSE resulted. In [16], Johensan suggests that for the TSK fuzzy model, the penalty on mismatch between the desired response and the actual response is greatest when the degree of overlap for membership functions is largest. He goes on the say that this causes rules of the fuzzy model to conflict. Given a set of membership functions with a modest degree of overlap, the TSK fuzzy model minimizes this controversy, thus minimizing the overall model error. Therefore, parameter bounds selection should be done with these concepts in mind.

8.2 Research Summary

A fuzzy rule set approach for complex signal modeling has been presented. The model is developed from a set of input and output object-oriented signal data, produced using condensation technology, arranged as a sequence of past events.

The fuzzy rule set is generated using an Adaptive Neuro-Fuzzy Inference System (ANFIS) and the architectural details of the ANFIS are presented; along with necessary background information related to the ANFIS. A genetic algorithm is used to train the ANFIS and the model produced is validated using highly nonlinear exothermic reaction process data.

8.3 Future Research Direction

A fuzzy rule set proves to be an effective way of modeling complex signals. Past sequence of events can be easily incorporated in the training data set. However, as the number of past events increases, the number of rule generated by the ANFIS goes up considerably. Some signals may be so complex, that more information about the signal is needed; that is, a greater number of past events should be included in the training set. Future reseach could consider the optimization of the rulebase. A method which determines which rules are most significant in capturing the dynamics of a signal could be incorporated into the ANFIS and the rulebase could be generated accordingly. This would not only reduce the number of rules, but also the computational time that is usually associated with generating a large rulebase.

In addition, the use of a fuzzy rule base for the control of object-oriented signal is quite reasonable since fuzzy control is well documented as an excellent control approach.

An ANFIS-generated rulebase appears to be an effective means of modeling extremely complex dynamics. It provides a way to predict future events of a process with a good degree of accuracy.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Birdwell, J. D., and B. C. Moore, "Condensation of information from signals for process modeling and control", *Hybrid Systems II*, Springer, pp. 45–63, 1995.

[2] Jang, J.-S. Roger, and Chuen-Tsai Sun, "Neuro-Fuzzy Modeling and Control", *IEEE Transaction of Neural Networks*, March 1995.

[3] Jang, J.-S. Roger, "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation", *IEEE Transaction on Neural Networks*, September 1992.

[4] Jang, J.-S. Roger and Chuen-Tsai Sun, "Predicting Chaotic Time Series with fuzzy if-then rules", *IEEE International Conferences on Fuzzy Systems*, March 1993.

[5] Kosko, Bart, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, 1992.

[6] Braspenning, P. J., F. Thuijsman, and A. J. M. M. Weijters, "Artificial Neural Networks: An Introduction of ANN Theory and Practice", *Springer-Verlag Berlin Heidelberg*, 1995.

[7] Aarts, E. H. L. and P. J. M. van Laarhoven, "Simulated Annealing: Theory and Applications", *Kluwer Academic Publishers*, 1989.

[8] Chen, Oscal T.-C., and Bing J. Sheu, "Optimization Schemes for Neural Network Training", *IEEE International Conference on Neural Networks*, 1994.

[9] Flannery, Brian P., William H. Press, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes in FORTRAN*, Press Syndicate of the University of Cambridge, 1992.

[10] Mohamed, J. L., and J. E. Walsh, *Numerical Algorithms*, Clarendon Press Oxford, 1986.

[11] Strang, G., *Linear Algebra and Its Application*, Harcourt Brace Jovanovich, 1988.

[12] The Math Works Inc., *Matlab Reference Guide*, Van Nostrand Reinhold Co., 1992.

[13] Luenberger, David G., *Linear and Nonlinear Programming*, Addison-Wesley Publishing Company, 1984.

[14] Chambers, Lance, *Practical Handbook of Genetic Algorithms: Applications*, CRC Press, Inc.,1995.

[15] Kim, J. W., Moon, Y. K. and Zeigler, B. P., *Designing Fuzzy Neural Net Controllers using GA Optimization*, Proc. of International Joint Symposium IEEE/IFAC on Computer-Aided Control System Design. Tucson, AZ, 1994.

[16] Johansen, Tor A., "On the Optimality of the Takagi-Sugeno-Kang Fuzzy Inference Mechanism", *IEEE International Conference on Fuzzy Systems*, 1995.

[17] Ong, Sze Wei, "Finite Impulse Response Hybrid System Modeling", MS Thesis, Dept. of Electrical and Computer Engineering, University of Tennessee, Knoxville, TN., August 1997.

[18] Houck, Christopher R., Joines, Jeffery A., and Kay, Micheal G., "A Genetic Algorithm for Function Optimization: A Matlab Implementation", http://www.eos.ncsu.edu/eos/info/ie589k$_i nfo/GAOT/$, *May* 1995.

APPENDICES

```
% [lpred,hpred,rpred] = mainfile(nmf,ninput,input_file,output_file)

% Program MAINFILE.M Arthur P. Heath, Jr.
% Program reads data from input/output scubber file.  This
% data is then used as training and checking data for
% prediction using an ANFIS system.  Parameters nmf and
% ninput are the number of membership functions and inputs,
% respectively.  Function returns predicted object data.
%
% The function accepts the following parameters:
%
%   nmf:  the number of membership function per input
%      ninput:  the number of inputs
%  input_file:  input scrubber file
% output_file:  output scrubber file
%     popsize:  population size for Genetic Algorithm
%      bounds:  upper and lower boundaries of parameter set for
%         use with Genetic Algorithm
%      genmax:  maximum number of generations

clear;
clear global;
format short
global lowin lowout highin highout N K
global n m M E datao datai
global time rangeout rangein
global rflowin rfhighin rfrangein rflowout
global rfhighout rfrangeout lastin lastout
global firstin firstout chkt0 t0 tf nmf npoint

% Add GAOT to MATLAB's directory search path
path(path,'/peach/heath/GAOT');

% Start real-time clock; get start time, date
time = fix(clock);

% Input Data
% Open 'testresi.dat' and read its contents (scrubber data)
fid = fopen('testresi.dat');
[datai,count] = fscanf(fid,'%d %d/%d/%d %d:%d:%d %d/%d/%d ...
%d:%d:%d %f %f',[15 inf]);
fclose(fid);
```

```
% Output Data
% Open 'testreso.dat' and read its contents (scrubber data)
fid = fopen('testreso.dat');
[datao,count] = fscanf(fid,'%d %d/%d/%d %d:%d:%d %d/%d/%d ...
%d:%d:%d %f %f',[15 inf]);
fclose(fid);

p = [0 0 1 0 0 0;1 0 0 0 0 0;0 1 0 0 0 0;zeros(3,3) eye(3)];
datetime = time*p;

% Output Data Object Parameters
lowout = zeros(size(datao,2),1);
highout = zeros(size(datao,2),1);
rangeout = zeros(size(datao,2),1);

% Input Data Object Parameters
lowin = zeros(size(datai,2),1);
highin = zeros(size(datai,2),1);
rangein = zeros(size(datai,2),1);

% Get object parameter for refining and use
% for training and checking data

for k = 1:size(datao,2)

% Output Object Information
  lowout(k,1) = datao(14,k);
  highout(k,1) = datao(15,k);
  bgnout(:,k) = p*datao([2:7],k);
  fnshout(:,k) = p*datao([8:13],k);
  rangeout(k,1) = etime(fnshout(:,k)',bgnout(:,k)');
end

for k = 1:size(datai,2)

% Input Object Information
  lowin(k,1) = datai(14,k);
  highin(k,1) = datai(15,k);
  bgnin(:,k) = p*datai([2:7],k);
  fnshin(:,k) = p*datai([8:13],k);
  rangein(k,1) = etime(fnshin(:,k)',bgnin(:,k)');
end
```

```
% Plot Output Object Data

a = 4;
firstout = zeros(size(datao,2),1);
lastout = zeros(size(datao,2),1);
subplot(2,1,1);
for k = 1:size(datao,2);
  firstout(k) = convert2(bgnout(:,k)',a);
  lastout(k) = convert2(fnshout(:,k)',a);
  patch([firstout(k) firstout(k) lastout(k) lastout(k)], ...
     [lowout(k) highout(k) highout(k) lowout(k)],'y');
end
grid;
title('"Data Refining Technology" Output Scrubber Data');
ylabel('Object Amplitude');
xlabel('Time Interval');
axis([min(firstout) max(lastout) min(lowout) max(highout)]);

% Plot Input Object Data

firstin = zeros(size(datai,2),1);
lastin = zeros(size(datai,2),1);
subplot(2,1,2);
for k = 1:size(datai,2);
  firstin(k) = convert2(bgnin(:,k)',a);
  lastin(k) = convert2(fnshin(:,k)',a);
  patch([firstin(k) firstin(k) lastin(k) lastin(k)], ...
        [lowin(k) highin(k) highin(k) lowin(k)], 'y');
end
grid;
title('"Data Refining Technology" Input Scrubber Data');
ylabel('Object Amplitude');
xlabel('Time Interval');


% Refine data.

disp('  ');
disp('Refining Object Time Intervals ...         ');
[rflowin,rfhighin,rfrangein,rflowout,rfhighout,rfrangeout] = ...
        refine(lowin,highin,rangein,lowout,highout,rangeout);
```

```
% Plot Refined Data

% CONVERT RFRANGEIN TO MATCH PLOTS ABOVE.
chkt0 = t0;
figure(3); subplot(2,1,2);
title('Refined Input Object Data');grid
ylabel('Object Amplitude');
xlabel('Time Interval');
for k = 1:length(rflowin)
  patch([chkt0 chkt0 (chkt0+rfrangein(k)) (chkt0+rfrangein(k))], ...
        [rflowin(k) rfhighin(k) rfhighin(k) rflowin(k)], ...
         'y');
  chkt0 = chkt0 + rfrangein(k);
end

chkt0 = t0;
subplot(2,1,1);title('Refined Output Object Data');grid
ylabel('Object Amplitude');
xlabel('Time Interval');
for k = 1:length(rflowout)
  patch([chkt0 chkt0 (chkt0+rfrangeout(k)) (chkt0+rfrangeout(k))], ...
        [rflowout(k) rfhighout(k) rfhighout(k) rflowout(k)], ...
         'y');
  chkt0 = chkt0 + rfrangeout(k);
end

chkt0 = t0;
figure;title('Refined Output Object Data and ANFIS Prediction');grid
ylabel('Object Amplitude');
xlabel('Time Interval');
for k = 1:length(rflowout)
  patch([chkt0 chkt0 (chkt0+rfrangeout(k)) (chkt0+rfrangeout(k))], ...
        [rflowout(k) rfhighout(k) rfhighout(k) rflowout(k)], ...
         'y');
  chkt0 = chkt0 + rfrangeout(k);
end

% ANFIS Training and Prediction

disp(' ');
disp('Training ANFIS ... ');
```

```
nmf = 2;  % Number of membership functions per input
ninput = 6;
rulebase = nmf^ninput;

% ANFIS parameters

%  Randomly select Training Data
l = length(rflowin);
l = 2*floor(l/2);

% First half of data used for training
start = 1;
indtrn = [start:start+floor(l/2)-6];
start = start + 1;
indtrn2 = [start:start+floor(l/2)-6];
start = start + 1;
indtrn3 = [start:start+floor(l/2)-6];
start = start + 1;
indtrn4 = [start:start+floor(l/2)-6];
start = start + 1;
indtrn5 = [start:start+floor(l/2)-6];
start = start + 1;
indtrn6 = [start:start+floor(l/2)-6];
start = start + 1;
indtrn7 = [start:start+floor(l/2)-6];
npoint = length(indtrn);


% Second half of data used for checking
start = ceil(l/2);
indchk = [start:start+floor(l/2)-6];
start = start + 1;
indchk2 = [start:start+floor(l/2)-6];
start = start + 1;
indchk3 = [start:start+floor(l/2)-6];
start = start + 1;
indchk4 = [start:start+floor(l/2)-6];
start = start + 1;
indchk5 = [start:start+floor(l/2)-6];
start = start + 1;
indchk6 = [start:start+floor(l/2)-6];
start = start + 1;
```

```
indchk7 = [start:start+floor(l/2)-6];
chkpt = length(indchk);

% Define parameter set height, midpoint, and range

heightin = rfhighin - rflowin;
midpointin = (rfhighin + rflowin)./2;
heightout = rfhighout - rflowout;
midpointout = (rfhighout + rflowout)./2;

% Preparing Training Data

% Training Data
TrainDataheight = [heightin(indtrn) heightin(indtrn2) ...
   heightin(indtrn3) heightin(indtrn4) ...
   heightin(indtrn5) heightin(indtrn6) ...
                  heightout(indtrn7)];
TrainDatamidpoint = [midpointin(indtrn) midpointin(indtrn2) ...
     midpointin(indtrn3) midpointin(indtrn4) ...
     midpointin(indtrn5) midpointin(indtrn6) ...
     midpointout(indtrn7)];
TrainDatarange = [heightin(indtrn) midpointin(indtrn) ...
  rfrangein(indtrn) rfrangeout(indtrn)];

% Checking Data
ChkDatahin = [heightin(indchk) heightin(indchk2) ...
      heightin(indchk3) heightin(indchk4) ...
      heightin(indchk5) heightin(indchk6)];
ChkDatamin = [midpointin(indchk) midpointin(indchk2) ...
      midpointin(indchk3) midpointin(indchk4) ...
      midpointin(indchk5) midpointin(indchk6)];
ChkDatainr = [heightin(indchk7) midpointin(indchk7) ...
      rfrangein(indchk7)];

ChkDataout = [heightout(indchk7) midpointout(indchk7) ...
      rfrangeout(indchk7)];

figure;

% Initial MFs
subplot(2,1,1);
[t1,mftrnin1] = genmsf(TrainDataheight(:,1),nmf);
title('Heightin(t-6), ..., Heightin(t)');
subplot(2,1,2);
[t1,mftrnin2] = genmsf(TrainDatamidpoint(:,1),nmf);
```

```
title('Midpointin(t-6), ..., Midpoint(t)');
figure
subplot(1,1,1);
[t1,mftrnin3] = genmsf(TrainDatarange(:,1),nmf);
title('Rangein(t)');

xx = sprintf('Training Data:  %d', npoint);
yy = sprintf('Checking Data:  %d', chkpt);
disp(xx);disp(yy);

x = sprintf('Linear Parameters:  %d', (rulebase*7));
y = sprintf('Nonlinear Parameters:  %d', (2*nmf*ninput));
z = sprintf('Total Number of Parameters:  %d', (7*rulebase ...
            + 2*nmf*ninput));
w = sprintf('Total Number of Rules:  %d', (rulebase));
disp(x);disp(y);disp(z);disp(w);
disp(' ');


[pset1h] = accparam(TrainDataheight(:,1),nmf);
[pset2h] = accparam(TrainDataheight(:,2),nmf);
[pset3h] = accparam(TrainDataheight(:,3),nmf);
[pset4h] = accparam(TrainDataheight(:,4),nmf);
[pset5h] = accparam(TrainDataheight(:,5),nmf);
[pset6h] = accparam(TrainDataheight(:,6),nmf);
paramsetheight = [pset1h; pset2h; pset3h; pset4h; pset5h; pset6h];

[pset1m] = accparam(TrainDatamidpoint(:,1),nmf);
[pset2m] = accparam(TrainDatamidpoint(:,2),nmf);
[pset3m] = accparam(TrainDatamidpoint(:,3),nmf);
[pset4m] = accparam(TrainDatamidpoint(:,4),nmf);
[pset5m] = accparam(TrainDatamidpoint(:,5),nmf);
[pset6m] = accparam(TrainDatamidpoint(:,6),nmf);
paramsetmidpoint = [pset1m; pset2m; pset3m; pset4m; pset5m; pset6m];

[pset1r] = accparam(TrainDatarange(:,1),nmf);
[pset2r] = accparam(TrainDatarange(:,2),nmf);
[pset3r] = accparam(TrainDatarange(:,3),nmf);
paramsetrange = [pset1r; pset2r; pset3r];

[errheight] = anftrn(TrainDataheight,ninput,nmf,paramsetheight,0.0002);
clear global TData
[errmidpoint] = anftrn(TrainDatamidpoint,ninput,nmf, ...
```

```
      paramsetmidpoint,0.0002);
clear global TData
[errrange,conrange] = anftrnr(TrainDatarange,3,nmf,paramsetrange);
clear global TData

ww = sprintf('Height SSE:  %6.4e', errheight);
disp(' ');disp(ww);
ww = sprintf('Midpoint SSE:  %6.4e', errmidpoint);
disp(' ');disp(ww);
ww = sprintf('Range SSE:  %6.4e', errrange);
disp(' ');disp(ww);

% Optimize nonlinear parameters beginning with
% random solution choice

disp(' ');
disp('Optimizing Nonlinear Parameters ...');
disp(' ');

disp('Optimizing ANFIS for "Height" prediction');
hbounds = [paramsetheight+0.5*paramsetheight  ...
           paramsetheight-0.5*paramsetheight];
mbounds = [paramsetmidpoint+0.01*paramsetmidpoint ...
           paramsetmidpoint-0.01*paramsetmidpoint];
[newparamsetheight,conheight] = optimize(TrainDataheight,hbounds, ...
                        ninput,nmf,paramsetheight,errheight, ...
                        2,15,0.0002);
clear global TrainData tolerance eparamset;
disp('Optimizing ANFIS for "Midpoint" prediction');
[newparamsetmidpoint,conmidpoint] = optimize(TrainDatamidpoint, ...
      mbounds,ninput,nmf,paramsetmidpoint, ...
        errmidpoint,2,15,0.0002);
clear global TrainData tolerance eparamset;
if errrange > 0.9
  disp('Optimizing ANFIS for "Range" prediction');
  [newparamsetrange,conrange] = optimize(TrainDatarange,3,nmf, ...
           paramsetrange,errrange);
 else
  newparamsetrange = paramsetrange;
end

% Evaluate ANFIS
clear global TrainData ChkData paramset
[heightpred] = evalf(TrainDataheight,ChkDatahin,ninput,nmf, ...
```

```
       conheight,newparamsetheight);
clear global TrainData ChkData paramset
[mpred] = evalf(TrainDatamidpoint,ChkDatamin,ninput,nmf, ...
  conmidpoint,newparamsetmidpoint);
clear global TrainData ChkData paramset
[rpred] = evalfr(TrainDatarange,ChkDatainr,3,nmf, ...
          conrange,newparamsetrange);

lpred = (2*mpred - heightpred)./2;
hpred = (heightpred + 2*mpred)./2;

% Plot Predicted Objects
figure(3);
hold on;
chkt0 = t0 + sum(rfrangein([1:1/2+5]));
for k = 1:length(rpred)
  patch([chkt0 chkt0 (chkt0+rpred(k)) (chkt0+rpred(k))], ...
        [lpred(k) hpred(k) hpred(k) lpred(k)],'r');
  chkt0 = chkt0 + rpred(k);
end
hold off;

% RMS Error
[rms_error] = rmse(heightpred,mpred,rpred,ChkDataout)

% Plot new membership functions

clear global paramset xdat
genfinp(newparamsetheight,TrainDataheight(:,1),nmf);
clear global paramset xdat
genfinp(newparamsetmidpoint,TrainDatamidpoint(:,1),nmf);
```

```
% REFINE - Program to refine object data by adjusting
% time interval of objects resulting in an equal number
% of input and output objects.

function [rflowin,rfhighin,rfrangein, ...
          rflowout,rfhighout,rfrangeout] = ...
          refine(lowin,highin,rangein, ...
          lowout,highout,rangeout);

global lowin highin rangein datai N
global lowout highout rangeout datao E
global firstin lastin firstout lastout
global rflowin rfhighin rfrangein rflowout
global rfhighout rfrangeout firstin firstout
global lastin lastout t0 tf

% Determine new object sequences starting and
% ending points.

t0 = max(min(firstin),min(firstout));
tf = min(max(lastin),max(lastout));

% Determine which objects are between t0 and tf.

startobjseqin = min(find(t0 <= lastin & lastin <= tf));
endobjseqin = max(find(t0 <= lastin & lastin <= tf));
startobjseqout = min(find(t0 <= lastout & lastout <= tf));
endobjseqout = max(find(t0 <= lastout & lastout <= tf));

% Define where new but unrefined object sequences
% begin and end.

firstin(startobjseqin) = t0; firstout(startobjseqout) = t0;
lastin(endobjseqin) = tf; lastout(endobjseqout) = tf;

% Setup new but unrefined object table for reference

objin = zeros(endobjseqin-startobjseqin+1,4);
objout = zeros(endobjseqout-startobjseqout+1,4);

ri = startobjseqin;
ro = startobjseqout;
for k = 1:endobjseqin-startobjseqin+1
```

```
  objin(k,:) = combine(lowin(k+ri-1),highin(k+ri-1), ...
               firstin(k+ri-1),lastin(k+ri-1))';
end
for k = 1:endobjseqout-startobjseqout+1
  objout(k,:) = combine(lowout(k+ro-1),highout(k+ro-1), ...
        firstout(k+ro-1),lastout(k+ro-1))';
end


numobjin = length(find(t0 < lastin < tf));
numobjout = length(find(t0 < lastout < tf));

% Determine the number of refined objects and
% arrange time intervals in ascending order

intervaldata = zeros(length(objin(:,3))+length(objout(:,4)),1);
intervaldata = [objin(:,3);objout(:,4)];
intervaldata = sort(intervaldata);

% Remove repeated start and end times from new time interval

diffintervaldata = diff(intervaldata);
r = find(diffintervaldata ~= 0);
rm = size(r,1)+1;
rn = size(intervaldata,1);
newintervaldata = intervaldata(r);
newintervaldata(rm) = intervaldata(rn);
intervaldata = newintervaldata;

% Begin Refining Process for Data by refining time interval

a = 0;
m = 1;
rfobjout = zeros(length(intervaldata)-1,4);
for k = 1:length(intervaldata)
  while k <= length(intervaldata) & (a+m+1) <= length(intervaldata)
    rfobjout(m+a,:) = combine(0,0, ...
                      intervaldata(m+a),intervaldata(m+a+1))';
    if (m+a+2) <= length(intervaldata)
      rfobjout(m+a+1,:) = combine(0,0, ...
                          intervaldata(m+a+1),intervaldata(m+a+2))';
    end
    a = a + 1;
```

```
    m = m + 1;
  end
end
rfobjin = rfobjout;

% Assign appropriate output object highs and lows

wo = zeros(length(intervaldata),1);
wi = zeros(length(intervaldata),1);
for k = 1:length(intervaldata)-1
  qo = find(rfobjout(k,3) == objout(:,3));
  if ~isempty(qo)
    wo(k) = qo;
    rfobjout(k,1) = objout(qo,1);
    rfobjout(k,2) = objout(qo,2);
   else
    rfobjout(k,1) = objout(max(wo),1);
    rfobjout(k,2) = objout(max(wo),2);
  end

% Assign appropriate input object highs and lows

  qi = find(rfobjin(k,3) == objin(:,3));
  if ~isempty(qi)
    wi(k) = qi;
    rfobjin(k,1) = objin(qi,1);
    rfobjin(k,2) = objin(qi,2);
   else
    rfobjin(k,1) = objin(max(wi),1);
    rfobjin(k,2) = objin(max(wi),2);
  end
end
end

% Refined input objects

rflowin = rfobjin(:,1);
rfhighin = rfobjin(:,2);
rfrangein = rfobjin(:,4) - rfobjin(:,3);

% Refined output objects
```

```
rflowout = rfobjout(:,1);
rfhighout = rfobjout(:,2);
rfrangeout = rfobjout(:,4) - rfobjout(:,3);
return;

f = combine(rfobjout(:,1),rfobjout(:,2),rfobjout(:,3),rfobjout(:,4))
```

```
function [bestsol,pbest] = optimize(TrainData,bounds,ninput,nmf,...
        paramset3,errout,genmax,populationSize,tolerance);

% OPTIMIZE - Program to iteratively update ANFIS
% membership function parameters and linear parameters
% using a Genetic Algorithm and Least Squares
% Estimation.  Function returns parameter set
% which corresponds to minimum SSE error.

global TrainData rulebase nmf npoint ninput tolerance

% Optimize parameters using GA
% Initialize GA parameters

% Run Binary Genetic Algorithm with Noise Assignment to parameter set
tstart = clock;
eparamset = [paramset3; -errout]';
[initialpop] =  initialize(populationSize,bounds,'evalfun',...
[],[1e-6 1]);
[bestsol,endPopulation,bestpop,trace] = ga(bounds,'evalfun',...
   [],initialpop,[1e-6 1 1], ...
'maxGenTerm',genmax,'tournSelect',ceil(0.5*populationSize));
figure;
plot(trace(:,1),(abs(trace(:,3))));
xlabel('Generation'); ylabel('Fitness');
bestsol = real(bestsol(1:length(paramset3)))';
[minerr,pbest] = anftrn(TrainData,ninput,nmf,bestsol,tolerance);
disp(' ');
xx = sprintf('Required Optimizing Time:  %5.4e hrs',...
     etime(clock,tstart)/3600);
disp(xx);
return
```

```
function [err,conseq] = anftrn(TData,ninput,nmf, ...
        paramset,tolerance);

% ANFTRN  Program that uses a least squares method and
% a Genetic Algorithm for identifying the linear and
% nonlinear parameters for the fuzzy inference system.
%
% Parameters:
%
%          nmf:  number of membership functions
%            ninput:  number of inputs
%          rulebase:  number of rules in rulebase
%     mf(parameter):  tabular list of mf values

global nmf 1
global ninput npoint

[rules] = numconv(nmf,ninput);
rulebase = nmf^ninput;

% Layer 1  Fuzzification of Universe of Discourse
%    for input and output parameters (training set)

% Generate MFs using appropriate parameter set

psetmf = zeros(2*nmf,ninput);

aa = 1;
for k = 1:ninput
  aaa = 2*nmf*k;
  psetmf(:,k) = paramset([aa:aaa]);
  aa = aaa + 1;
end

[t1,mflin] = genfin(psetmf(:,1),TData(:,1),nmf);
[t1,mfhin] = genfin(psetmf(:,2),TData(:,2),nmf);
[t1,mfrangein] = genfin(psetmf(:,3),TData(:,3),nmf);
[t1,mftrn4] = genfin(psetmf(:,4),TData(:,4),nmf);
[t1,mftrn5] = genfin(psetmf(:,5),TData(:,5),nmf);
[t1,mftrn6] = genfin(psetmf(:,6),TData(:,6),nmf);
wnormin = zeros(size(rules,1),size(mflin,2));

% Generate rule structure and calculate firing strengths
```

```
mflin = mflin([2:nmf+1],:);
mfhin = mfhin([2:nmf+1],:);
mfrangein = mfrangein([2:nmf+1],:);
mftrn4 = mftrn4([2:nmf+1],:);
mftrn5 = mftrn5([2:nmf+1],:);
mftrn6 = mftrn6([2:nmf+1],:);
rulebasein = zeros(size(rules,1),size(mflin,2));

for r = 1:size(rules,1)
  rulebasein(r,:) = [mflin(rules(r,1),:).*mfhin(rules(r,2),:).* ...
    mfrangein(rules(r,3),:).*mftrn4(rules(r,4),:).* ...
    mftrn5(rules(r,5),:).* ...
    mftrn5(rules(r,6),:)];
end

% Layer 3  Compute the normalized firing strength

for r = 1:size(rules,1)
  wnormin(r,:) = rulebasein(r,:)./sum(rulebasein);
end

% Layer 4  Determine Consequent Parameters by Perform Least
%          Squares Estimate

xp = zeros(7,rulebase);
totaloutputparam = zeros(size(TData,1),1);

for k = 1:rulebase

  Ahat = [wnormin(k,:)'.*TData(:,1) wnormin(k,:)'.*TData(:,2) ...
          wnormin(k,:)'.*TData(:,3) wnormin(k,:)'.*TData(:,4) ...
          wnormin(k,:)'.*TData(:,5) wnormin(k,:)'.*TData(:,6) ...
          wnormin(k,:)'.*ones(size(TData,1),1)];

  bhat = [wnormin(k,:)'.*TData(:,7)];
  xp(:,k) = pinv(Ahat,tolerance)*bhat;

%  e1 = norm(Ahat*xp(:,k)-bhat);

% Layer 5  Sum of the node outputs of layer 5

  totaloutputparam = totaloutputparam + Ahat*xp(:,k);
```

```
end
conseq = xp;

% Calculate SS Error

[err] = sse(totaloutputparam,TData(:,7));

return
```

```
function [err,conseq] = anftrnr(TData,ninput,nmf,paramset);

% ANFTRN  Program that uses a least squares method and
% a Genetic Algorithm for identifying the linear and
% nonlinear parameters for the fuzzy inference system.
%
% Parameters:
%
%          nmf:  number of membership functions
%            ninput:  number of inputs
%          rulebase:  number of rules in rulebase
%     mf(parameter):  tabular list of mf values

global rules nmf 1
global rulebase ninput npoint

[rules] = numconv(nmf,ninput);
rulebase = nmf^ninput;

% Layer 1  Fuzzification of Universe of Discourse
%    for input and output parameters (training set)

% Generate MFs using appropriate parameter set

psetmf = zeros(2*nmf,ninput);

aa = 1;
for k = 1:3 %ninput
  aaa = 2*nmf*k;
  psetmf(:,k) = paramset([aa:aaa]);
  aa = aaa + 1;
end

[t1,mflin] = genfin(psetmf(:,1),TData(:,1),nmf);
[t1,mfhin] = genfin(psetmf(:,2),TData(:,2),nmf);
[t1,mfrangein] = genfin(psetmf(:,3),TData(:,3),nmf);
wnormin = zeros(size(rules,1),size(mflin,2));

% Generate rule structure and calculate firing strengths

mflin = mflin([2:nmf+1],:);
mfhin = mfhin([2:nmf+1],:);
mfrangein = mfrangein([2:nmf+1],:);
```

```
rulebasein = zeros(size(rules,1),size(mflin,2));

for r = 1:size(rules,1)
  rulebasein(r,:) = [mflin(rules(r,1),:).*mfhin(rules(r,2),:).* ...
    mfrangein(rules(r,3),:)];
end

% Layer 3  Compute the normalized firing strength

for r = 1:size(rules,1)
  wnormin(r,:) = rulebasein(r,:)./sum(rulebasein);
end

% Layer 4  Determine Consequent Parameters by Perform Least
%          Squares Estimate

xp = zeros(4,rulebase);
totaloutputparam = zeros(size(TData,1),1);

for k = 1:rulebase

  Ahat = [wnormin(k,:)'.*TData(:,1) wnormin(k,:)'.*TData(:,2) ...
          wnormin(k,:)'.*TData(:,3) wnormin(k,:)'.* ...
  ones(size(TData(:,1)),1)];

  bhat = [wnormin(k,:)'.*TData(:,4)];
  xp(:,k) = pinv(Ahat,0.0005)*bhat;

  e1 = norm(Ahat*xp(:,k)-bhat);
% Layer 5  Sum of the node outputs of layer 5

  totaloutputparam = totaloutputparam + Ahat*xp(:,k);

end
conseq = xp;

% Calculate SS Error

[err] = sse(totaloutputparam,TData(:,4));

return
```

```
function [pred] = evalf(TrnData,ChkData,ninput,nmf, ...
  conseqp,opparamset)

% EVALF - Evaluation of Adaptive Neural Fuzzy Inference System.
% Computes output using checking data for optimal membership
% function parameter sets.


[rules] = numconv(nmf,ninput);
rulebase = nmf^ninput;
pred = zeros(size(ChkData,1),1);

%  Pass optimal parameter set

psetmf = zeros(2*nmf,ninput);
aa = 1;
for k = 1:ninput
  aaa = 2*nmf*k;
  psetmf(:,k) = opparamset([aa:aaa]);
  aa = aaa + 1;
end

% Layer 1  Fuzzification of Universe of Discourse
%    for input validation set with optimal parameter set

[t1,mflin] = genfin(psetmf(:,1),TrnData(:,1),nmf);
[t2,mfhin] = genfin(psetmf(:,2),TrnData(:,2),nmf);
[t3,mfrangein] = genfin(psetmf(:,3),TrnData(:,3),nmf);
[t1,mftrn4] = genfin(psetmf(:,4),TrnData(:,4),nmf);
[t2,mftrn5] = genfin(psetmf(:,5),TrnData(:,5),nmf);
[t3,mftrn6] = genfin(psetmf(:,6),TrnData(:,6),nmf);

% Generate rule structure and calculate firing strengths

mflin = mflin([2:nmf+1],:);
mfhin = mfhin([2:nmf+1],:);
mfrangein = mfrangein([2:nmf+1],:);
mftrn4 = mftrn4([2:nmf+1],:);
mftrn5 = mftrn5([2:nmf+1],:);
mftrn6 = mftrn6([2:nmf+1],:);
wnormin = zeros(size(rules,1),size(mflin,2));
rulebasein = zeros(size(rules,1),size(mflin,2));
```

```
for r = 1:size(rules,1)
  rulebasein(r,:) = mflin(rules(r,1),:).*mfhin(rules(r,2),:).* ...
    mfrangein(rules(r,3),:).*mftrn4(rules(r,1),:).* ...
    mftrn5(rules(r,2),:).*mftrn6(rules(r,1),:);
end

% Layer 3  Compute the normalized firing strength

for r = 1:size(rules,1)
 wnormin(r,:) = rulebasein(r,:)./sum(rulebasein);
end

% Compute ANFIS output

% Consequent Parameter Set

% Layer 5

x = ChkData(:,1); % Change to checking data
y = ChkData(:,2);
z = ChkData(:,3);
u = ChkData(:,4);
v = ChkData(:,5);
w = ChkData(:,6);


for r = 1:rulebase
  pred = pred + wnormin(r,:)'.*(x*conseqp(1,r) + y*conseqp(2,r) + ...
 z*conseqp(3,r) + u*conseqp(4,r) + v*conseqp(5,r) + ...
         w*conseqp(6,r)+ conseqp(7,r));
end
```

```
function [pred] = evalfr(TrnData,ChkData,ninput,nmf,conseq,opparamset)

% EVALFR - Evaluation of Adaptive Neural Fuzzy Inference System.
% Computes output using checking data for optimal membership
% function parameter sets.

[rules] = numconv(nmf,ninput);
rulebase = nmf^ninput;
pred = zeros(size(ChkData,1),1);

%  Pass optimal parameter set

psetmf = zeros(2*nmf,ninput);
aa = 1;
for k = 1:3 %ninput
  aaa = 2*nmf*k;
  psetmf(:,k) = opparamset([aa:aaa]);
  aa = aaa + 1;
end

% Layer 1  Fuzzification of Universe of Discourse
%    for input validation set with optimal parameter set

[t1,mflin] = genfin(psetmf(:,1),ChkData(:,1),nmf);
[t2,mfhin] = genfin(psetmf(:,2),ChkData(:,2),nmf);
[t3,mfrangein] = genfin(psetmf(:,3),ChkData(:,3),nmf);


% Generate rule structure and calculate firing strengths

mflin = mflin([2:nmf+1],:);
mfhin = mfhin([2:nmf+1],:);
mfrangein = mfrangein([2:nmf+1],:);


wnormin = zeros(size(rules,1),size(mflin,2));
rulebasein = zeros(size(rules,1),size(mflin,2));

for r = 1:size(rules,1)
  rulebasein(r,:) = mflin(rules(r,1),:).*mfhin(rules(r,2),:).* ...
    mfrangein(rules(r,3),:);
end
```

```
% Layer 3  Compute the normalized firing strength

for r = 1:size(rules,1)
 wnormin(r,:) = rulebasein(r,:)./sum(rulebasein);
end

% Compute ANFIS output

% Consequent Parameter Set

% Layer 5

x = ChkData(:,1); % Change to checking data
y = ChkData(:,2);
z = ChkData(:,3);

for r = 1:rulebase
  pred = pred + wnormin(r,:)'.*(x*conseq(1,r) + y*conseq(2,r) + ...
 z*conseq(3,r) + conseq(4,r));
end
```

```
function [parameterset] = accparam(dataset,nmf)

% ACCPARAM - Function calculates initial nonlinear parameters for
% membership functions.  Function accepts input data, 'data' and
% the number of membership function per input, 'nmf'.  Function
% returns 'nmf' parameter sets.  Membership functions are
% Gaussian.


div = size(dataset,1);

% Determine number of individual parameter sets.

xm = dataset/max(dataset);
a = zeros(nmf,1);
c = zeros(nmf,1);
parameterset = zeros(nmf*2,1);

xn = zeros(div,nmf);
xn = linspace(min(xm),max(xm),div)';
center = linspace(min(xm),max(xm),nmf+2);

for k = 1:nmf
  a(k) = std(xn); %2.084*mean(xn)/(nmf+1);
  c(k) = center(k+1);
end

a = a.*max(dataset);
c = c.*max(dataset);
j = 1;
for k = 1:2:(3*nmf)-3
  parameterset(k) = a(j);
  parameterset(k+1) = c(j);
  j = j + 1;
end

clear dataset
return
```

```
function [mfvalue,tabgrade] = genmsf(x,nmf);

% GENMSF - Generate initial Gaussian
% membership function.  Function accepts values for
% fuzzification and generates three parameters that determine the
% shape and position of the curve.  Function returns the
% initial membership functions and tabulated grade values.
%
% The Gaussian MF has the following formula:
%
% gbmf(x,a,c) = exp(-((x-c)/a)^2)
% where
% 'x' is the input values for fuzzification
%  'a' is the width of the mf
%   'c' is the center of the mf
%     'nmf' is the number of membership functions
%

if nargin < 2
  nmf = 2;
end

div = size(x,1);
xm = sort(x)/max(x);

% Curve Parameters

mfvalue = zeros(div,nmf);xn = zeros(div,nmf);

% Generate membership functions

xn = linspace(min(xm),max(xm),div)';
a = zeros(nmf,1);
c = zeros(nmf,1);

center = linspace(min(xm),max(xm),nmf+2);
for k = 1:nmf
  c(k) = center(k+1);
  a(k) = std(xn);  %2.084*mean(xn)/(nmf+1);
  mfvalue(:,k) = exp(-((xn-c(k))./a(k)).^2);
end
```

```
a = a.*max(x);
c = c.*max(x);

% Tabulate MF values for reference

tabgrade = combine((xn*max(x))',mfvalue');
hold on;
for n = 1:nmf
  plot(xn*max(x),mfvalue(:,n));
end
hold off;
axis([min(x) max(x) min(min(mfvalue')) max(max(mfvalue'))]);
grid;
xlabel('Universe of Discourse');
ylabel('Membership Grade');

return
```

```
function [mfvalue,tabgrade] = genfin(paramset,xd,nmf);

% GENFIN- Function accepts values for fuzzification and
% appropriate MF parameters and generates Guassian
% membership function.  The generalized bell is of the form:
%
% gbmf(x,a,c) = exp(-((x-c)/a)^2)
%
% where
%  'x' is the input values for fuzzification
%   'a' is the width of the mf
%    'c' is the center of the mf
%      'nmf' is the number of membership functions
%
% a,b and c are contained in 'paramset'
%
% The fourth input argument is for displaying membership
% functions.

global nmf ninput

if nargin < 2
  nmf = 2;
end

div = size(xd,1);
xm = sort(xd)/max(xd);

% Curve Parameters

mfvalue = zeros(div,nmf);xn = zeros(div,nmf);
p = paramset/max(xd);

% Generate membership functions

xn = linspace(min(xm),max(xm),div)';
w = 1;
for k = 1:2:2*nmf
  mfvalue(:,w) = exp(-((xn-p(k+1))./p(k)).^2);
  w = w + 1;
end
```

```
% Tabulate MF values for reference

tabgrade = combine((xn*max(xd))',mfvalue');

return
```

```
% CONVERT - Mfile to convert real time to days, hours, minutes
% or seconds.
% Date and Time information is given in the following format:
%
% [Year Month Day Hour Minute Second]
%
% Use 'n' to specify whether output is in days, hours, minutes,
% or seconds by n equal to 1, 2, 3, or 4 respectively.


function [a] = convert(b,n);

% Seconds
if n == 4
  a = b(3)*86400 + b(4)*3600 + b(5)*60 + b(6);
end

% Minutes
if n == 3
  a = b(3)*1400 + b(4)*60 + b(5) + b(6)/60;
end

% Hours
if n == 2
  a = b(3)*24 + b(4) + b(5)/60 + b(6)/3600;
end

% Days
if n == 1
  a = b(3) + b(4)/24 + b(5)/1440 + b(6)/86400;
end
```

```
function [fiterr,eparamset] = evalfun(eparamset,options)

% Evaluation Function

global TrainData ninput nmf tolerance paramset1

paramset1 = eparamset([1:size(eparamset,2)-1])';
[erro,p] = anftrn(TrainData,ninput,nmf,paramset1,tolerance);

fiterr = -erro;
eparamset = [paramset1; fiterr]';

return
```

```
function [err] = sse(outputp,TrData);

% SSE - Program to calculate the Sum of Squared error given by
% the following:
%
% err = sum((lhat-l)^2)
%
% where l is the desired object parameter and lhat is the
%       actual object parameter.

lhat = outputp;
l = TrData;

err = sum((lhat-l).^2);
return
```

```
function [err_avg] = rmse(outputh,outputm,outputr,ChkDat);

% RMSE - Program to calculate the RMS error given by
% the following:
%
%  p1 = {-(y1h-y2h)^4 + (y1h-y2l)^4 + (y1l-y2h)^4 - (y1l-y2l)^4}
%
%  p2 = 12*(y1h-y1l)*(y2h-y2l)
%
%  p = p1/p2
%
%  e = [(deltat)*p/(deltaf)]^(1/2), deltat = tfo - t0o
%      deltatf = tf - t0
%
% where y1 is the actual object parameter output and
% y2 is desired object parameter output.  l and h represent
% the object parameters low and high, respectively.
% tf and t0 are the initial and final time of the object
% sequence while tfo and t0o are the initial and final time
% of each object.

global  t0 tf

y1l = outputh;
y1h = outputm;
y2l = ChkDat(:,1);
y2h = ChkDat(:,2);

p1 = (-(y1h-y2h).^4 + (y1h-y2l).^4 + (y1l-y2h).^4 - (y1l-y2l).^4);
p2 = 12*((y1h-y1l).*(y2h-y2l));
p = p1./p2;
deltat = ChkDat(:,3);
deltaf = tf - t0;
err =(deltat.*(p./deltaf));
err_avg = mean((deltat.*(p./deltaf)).^0.5);
%figure;plot(err,deltat);xlabel('time (sec)');
```

```
function [r] = numconv(nmf,ninputs)

% NUMCONV  Number Conversion Program
%
% Converts a base 10 number, num, to base m, where m = 2, 3
% and ... .  Program associated with ANFIS Training Program
% rule selection.  The function is of the form:
%
% [convnum] = numconv(nmf,ninputs)
%
% where 'nmf' and 'ninputs' are the number of membership
% functions and the number of inputs, respectively, and
% 'convnum' is the converted number.

global r rulebase

rulebase = nmf^ninputs;
m = nmf;
r = zeros(1,ceil(rulebase/m));

c = 1;
for a = 0:rulebase-1
  num = a;n = 1;
  while num ~= 0
    r(a+1,n) = rem(num,m);
    num = fix(num/m);
    n = n + 1;
  end
  r(a+1,:) = fliplr(r(a+1,:));
end

r = r(:,[size(r,2)-(n-2):size(r,2)]);

% Since a zero'th index is not permitted in MATLAB

r = ones(size(r,1),size(r,2)) + r;
```

VITA

Arthur Pedro Heath, Jr. was born in Goldsboro, North Carolina on November 6, 1972. He attended schools in the Wayne County Public School System, where he graduated from Goldsboro High School in 1991. He attended North Carolina Agricultural and Technical State University and received a Bachelor of Science in Degree in electrical engineering in 1995. Following graduation, he began working for Sandia National Laboratories, Albuquerque, New Mexico. In August, 1995, he entered the Master's program in electrical engineering at the University of Tennessee, Knoxville. He officially received his Master's Degree in December 1997. Afterwards, he returned to Sandia National Laboratories as a technical staff member.