5-21-2004

A Fuzzy/Neural Approach to Cost Prediction with Small Data Sets

Holly Danker-McDermot
University of New Orleans

A FUZZY/NEURAL APPROACH TO COST PREDICTION WITH SMALL DATA SETS

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirement for the degree of

Master of Science
in
The Department of Electrical Engineering

by

Holly A. Danker-McDermot

B.S., University of New Orleans, 2002

May 2004

ACKNOWLEDGEMENTS

I would like to thank Dr. Bourgeois for all of her help and assistance to me throughout the duration of this project. I would also like to thank my family starting with my father, Robert Danker, for his ongoing love, support and encouragement during all my years of higher education; my grandparents, Vi and Carl Clements, my “second set of parents” who have always been there for me, and Henry and Emily Danker for their loving generosity and support, my stepfather Jeffrey Sandler, who gives great advice and always makes me laugh, and especially my mother, Debra, for being my source of inspiration on attaining my masters degree. Most importantly, I want to thank my husband, Kyle, whose constant love and support is vital to my success.

ABSTRACT

The project objective in this work is to create an accurate cost estimate for NASA engine tests at the John C. Stennis Space Center testing facilities using various combinations of fuzzy and neural systems. The data set available for this cost prediction problem consists of variables such as test duration, thrust, and many other similar quantities, unfortunately it is small and incomplete. The first method implemented to perform this cost estimate uses the locally linear embedding (LLE) algorithm for a nonlinear reduction method that is then put through an adaptive network based fuzzy inference system (ANFIS). The second method is a two stage system that uses various ANFIS with either single or multiple inputs for a cost estimate whose outputs are then put through a backpropagation trained neural network for the final cost prediction. Finally, method 3 uses a radial basis function network (RBFN) to predict the engine test cost.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Cost estimation is a vital aspect of any project. Some form of cost estimation must be performed to determine if a design is feasible to actually realize or, as in this case, to present an estimate to a customer. Even small adjustments to already designed systems require some sort of cost projection analysis to determine the viability of the proposed improvements. The project objective in this work is to create an accurate rough order of magnitude cost estimate for engine tests at NASA's John C. Stennis Space Center testing facilities or in other words create an accurate nonlinear mapping between the data available to the total cost of the test. NASA also specified the use of fuzzy networks to create this cost prediction system.

This research stems from the development of HACEM (Highly Accurate Cost Estimating Model) [1-2]. The HACEM system used a small set of data attained from project requirement documents to predict the total cost of component and engine tests. This model used adaptive network based fuzzy inference systems (ANFIS), realized in Matlab, that were trained by the given data to predict these test costs. Different realizations of these ANFIS were tested including a simple ANFIS system, an ANFIS system using principal component analysis, and a cascaded ANFIS system. While the results obtained from the HACEM system were good, the study presented in this paper attempts to attain better results. This presents many complications because the only available data sets are small and incomplete.

The data sets utilized were extremely small because much of the information was eliminated due to incomplete data. Even the remaining small data sets contained incomplete data in some variable columns. The various data variables included in each set consisted of quantities like engine thrust, duration of tests in days, and other quantities that will be discussed in the following sections. Finally, the number of variables in each data set was much larger than the size of the items itself, which creates numerous problems when attempting to train fuzzy systems and neural networks. Three different methods were applied to the problem of predicting engine test costs despite the incomplete data; method 1 using locally linear embedding; method 2 using both ANFIS and neural networks; and method 3 using a radial basis function network.

The dimensionality of the data set needs to be reduced because typically for neural networks to train accurately, the dimension of the input data should be much smaller than the actual number of training examples. In method 1, the locally linear embedding (LLE) algorithm [17-18] is used to reduce the dimensionality of the data set which is then used to train an ANFIS to predict the cost of engine tests. One of the systems developed in HACEM, PCA-ANFIS, used principal component analysis (PCA) to reduce the dimensionality of the data set. PCA is a linear operation, however, and this system is highly nonlinear. LLE is a nonlinear method of reducing the dimensionality of the data set and was expected to produce better results.

Method 2 is a more heuristic approach using several ANFIS with a small number of inputs, respectively, and then combining the resulting ANFIS predictions with a neural network to attain a more accurate cost prediction. Several different sizes of ANFIS systems and neural networks were tried to see which produced better results. Method 2A refers to the system developed with each ANFIS taking a single variable as an input, respectively, and the resulting ANFIS outputs are then combined in a neural network. Method 2B is similar, except instead of each ANFIS taking only one variable, each ANFIS takes multiple variables whose outputs are then combined with a neural network to produce the final cost estimate.

Method 3 uses a Radial Basis Function Network (RBFN) to predict the output costs. This method contains no ANFIS part, however, the RBFN is similar to an ANFIS in that it uses membership functions to classify the data. As with method 2, several different sizes of RBFN are used. Also, two different implementations of the RBFN were realized. The first was to use the entire data set of eleven for training, due to the difficulty of clustering such a small data set. This is not a fair way to develop and test a network, however these results will be a good comparison to the correct way of implementing the RBFN because it shows how the results are being affected by the small size of the data set. Because of the inherent inaccuracies of this approach, these results were then compared to the correct way of developing and testing the RBFN by dividing the data set into seven training examples and four testing items.

The organization of the report is presented below. Section 2, past work, contains a brief summary of some of the research performed on this topic, including the HACEM project. Section 3, data and processing issues, discusses in depth the data itself and the choice of data variables used to test each method. Section 4 contains details on ANFIS, specifically the ANFIS toolbox in Matlab. Section 5 contains a brief discussion of the locally linear embedding algorithm and a description of how this algorithm was used to predict engine test costs in method 1. Section 6 discusses the fuzzy/neural approach used in method 2 for the cost prediction. Section 7 includes a discussion of the radial basis function network used in method 3 for cost prediction. Section 8 contains a lengthy presentation, discussion, and comparison of the results attained from using methods 1, 2, and 3. Section 9 contains concluding remarks and suggestions for future work. Finally, the appendices contain the actual data used, the codes used to represent information about the data, and the Matlab code used to develop and implement these methods.

2. PAST WORK

2.1. HACEM

HACEM or Highly Accurate Cost Estimating Model is a system using several Adaptive Network-Based Fuzzy Inference Systems (ANFIS) developed to predict the cost of engine and component tests for the testing facilities of NASA's John C. Stennis Space Center (SSC). This system was needed because the current method of cost prediction is more of a heuristic method. Three different systems were developed as part of the HACEM project relating to engine testing. The first, ANFIS-E, was a simple ANFIS developed through trial and error. The second, PCA-ANFIS-E, was also a simple ANFIS, however principal component analysis was used to reduce the data dimensionality. The third ANFIS scheme developed for estimation was the Parallel ANFIS-E. This method consists of several parallel ANFIS handling particular inputs whose outputs are then combined in the final stage consisting of a single ANFIS [1-2].

These methods attained various degrees of accuracy. The ANFIS-E performed the best with a total root mean squared error of 8%, even with one article being estimated with an error of about 40%. Likewise, the PCA-ANFIS-E had the same article estimated with a 40% error. Finally, the Parallel-ANFIS-E system also had trouble estimating that outlying article, while it estimated the other articles closely [1].

2.2. Nonlinear Dimensionality Reduction

When dealing with fuzzy systems or neural networks it is always preferable to have more data sets than the dimensionality of the data because inputting too much information for each data set into the system will make the results less accurate. This was a large problem for the NASA engine test data because there were only eleven viable data sets with a dimensionality of nineteen. One method to ensure the system attains more accuracy is to somehow reduce this data set so that only the most important information is given to the network, while all other data is eliminated so as not to confuse the system. In HACEM, principal component analysis was used to reduce the dimensionality of the data set [1]. The only problem with using PCA in this situation was that it is a linear transformation and the data has a highly nonlinear relationship between individual data components. This is why using a nonlinear dimensionality reduction method is a favorable alternative to using PCA. Unfortunately, there does not seem to be a large amount of research in this field. Most of the research found on this topic was usually related to image processing, which does not suffer from the problem of small data sets as in the case of the engine test data.

The Isomap (isometric feature mapping) method, developed by Tenenbaum, Silva, and Langford [3], is one of the nonlinear dimensionality reduction methods that have been applied to image processing. This algorithm attempts to use classical multidimensional scaling (MDS) to map data points from a high dimensional input space into low dimensional coordinates of a nonlinear manifold [4]. This method works by first finding the neighborhood of each point. This neighborhood can be calculated using the k nearest neighbors or the set of points within a certain radius. These neighborhoods are represented as a weighted graph over the data points with edges weighted by the Euclidean distance between neighbors. Next, the approximated geodesic distance between two points is found by computing the sum of the arc lengths along the shortest path connecting both points [5]. Finally, the classical metric MDS method is applied to the approximated geodesic distances. MDS is used to compute the largest eigenvectors, which give the coordinates of the data points in the lower-dimensional space [4-5].

The Isomap method is relies heavily on the nearest neighbor algorithm, which is not viable for use in extremely small data sets because almost any point could be considered a neighbor. Most of the nonlinear dimensionality reduction methods, however, required some sort of nearest neighbor processing [4-7]. There is simply not enough data to make a good neighborhood grouping. This is especially true of the meager eleven data sets for engine testing.

2.3. Dealing with Incomplete Data Sets

Another problem with the engine test data is that frequently there is information missing in each data set. Ideally, if this information cannot be found it would be best to eliminate these data variables entirely. This is not a viable option in this situation, however, since almost all of the data variables have at least one quantity missing. Further, even if a certain variable has no missing data, it may not be predictive of the cost. As with the nonlinear dimensionality reduction problem, there does not seem to be a large amount of research in dealing with incomplete data sets.

Much of the research dealing with incomplete data sets involve neural classification systems. Ishibuchi *et al.* [8] proposed a method for dealing with incomplete data by using an interval representation of incomplete data with missing inputs. After a network is trained using learning algorithms for interval training data, a new sample consisting of the missing inputs, is presented along with an interval vector. The output from the neural network is also an interval vector. This output is then classified using four definitions of inequality between intervals [8]. This method is more theoretical in its implementation than is desirable for the cost prediction problem.

Granger *et al*. [9] proposed using a fuzzy ARTMAP neural network to deal with incomplete data for a classification problem. This approach presented the fuzzy ARTMAP with an indicator vector that described whether a data component was present or not. Unlike replacement methods, the weight vector is modified as well as the input vector in response to missing components [9]. A future implementation of this method might perform well with the engine test cost prediction problem.

Another method to deal with incomplete data is using the normal information diffusion model, which divides an observation into many parts according to a normal function [10]. This technique attempts to find a suitable membership function to represent a fuzzy group that represents the incomplete data. This fuzzy group is then used to derive more data samples [10]. Unfortunately, this method can be computationally intensive.

Finally, some other methods viable for the engine data test sets are mean and multiple imputation. Mean imputation is simply replacing the missing data with the mean value of the sample. This method can cause misleading results because the changed data cannot reflect the uncertainty due to the missing data. Multiple imputation is another method that is similar to mean imputation, however, the missing data is replaced by a set of possible values from their predictive distribution. This set reflects the uncertainty of the values predicted from the observed ones [11]. This method yields much better results than mean imputation, however, it can be computationally intensive. A variation on mean imputation was used in methods 2 and 3.

3. DATA AND PROCESSING ISSUES

The primary challenge with predicting the engine test cost is due to the small amount of data available for training and testing. The goal of this project is to create an accurate nonlinear mapping between the data available to the total cost of the test. If the quality of the data is poor or the amount of data is small, an accurate nonlinear mapping is very difficult to attain. The project requirement documents (PRD) attained from NASA provided all of the data. This data totaled an original amount of 38 data sets of which only 32 were determined to be viable because of completeness. Then, it was further reduced since only about one third of this data was from engine tests, the rest was provided from component tests [2]. The component tests involve testing individual engine components while the engine tests involve testing the entire engine system. The engine test was the only quantity used in this work. This reduced the total number of examples available for use for predicting engine test cost to 11. This set then had to be divided into training and testing data, which will be discussed later in this section. The reduced data set used for engine cost prediction is contained in the Appendix 12.1. An article number denotes the actual data sets. The article numbers that were used are 1, 2, 5, 6, 14, 15, 28, 29, 30, 34, and 35. However, for convenience and greater clarity the article numbers used in this paper are different than the actual NASA article numbers. The correspondence between article numbers is illustrated in Appendix 12.1.

In Figure 3.1, the engine test articles costs are shown. The article labeled 6 in the plot is clearly much more costly than any of the others in the set of 11. These are the costs that are to be predicted using the methods presented in later sections (5 and 6) of this paper.

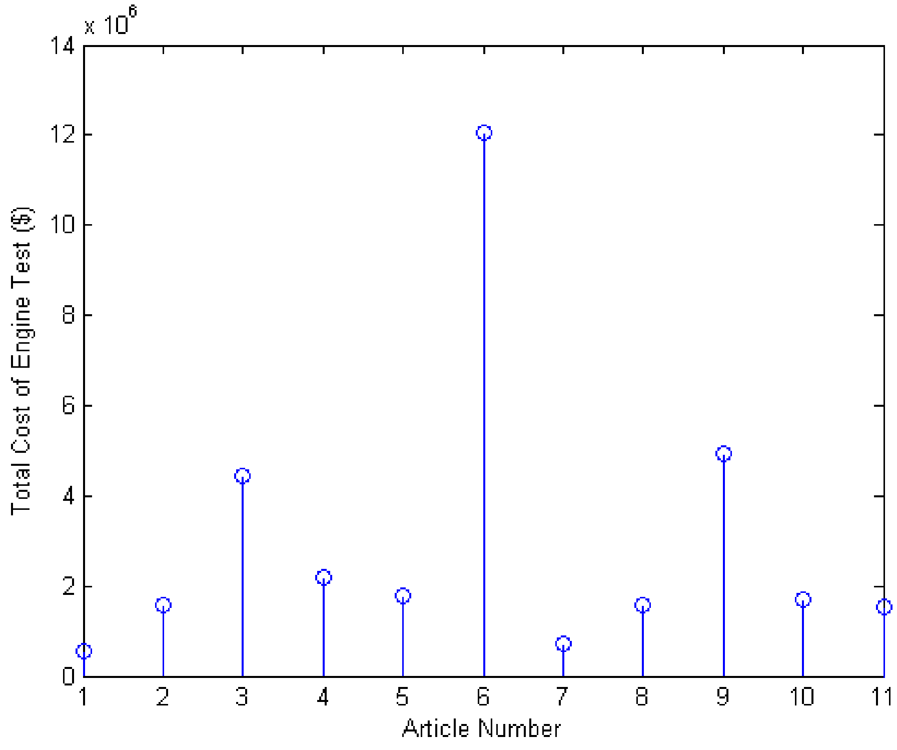


Figure 3.1: Total Cost of Engine Tests

In Table 3.1, the data variables are listed and described. An X in the last column denotes that the variable that was considered for use in the cost estimating system. These variables were picked for various reasons, however, they were not all used. Some were eliminated after performing exhaustive and sequential searches, which are described later in this section. A (b) in the third column denotes a quantity that was made a Boolean value, 1 indicating yes and 0 indicating no. Article number 1 has the following data: 9, 2, 70, 4250, 3, 550000, which corresponds to variables 1, 2, 3, 7, 17, and 19. This means that article number 1 has an estimated duration of 9 days, with 2 tests to be performed, each test lasting a maximum of 70 seconds, with a thrust of 4250, performed on test stand E3, with a total cost of testing equal to $550,000.

Table 3.1: Data Variables for Engine Tests

	Variable	Description	Use
1	DuratDd	Duration of tests in days	X
2	NoTest	Number of tests	X
3	TestDurMax	Maximum duration of test	X
4	Fuel	Fuel code	X
5	Pressurant	Pressurant code	X
6	Oxidizer	Oxidizer code	X
7	Thrust	Thrust	X
8	FuelFlow	Fuel flow rate	X
9	FuelPressfl	Fuel pressurant flow rate	
10	OxidizerFL	Oxidizer flow rate	X
11	PressuraPr	Pressurant pressure	X
12	ThrustMeas	Thrust measurement (b)	
13	Cooling?	Cooling system (b)	
14	GimbalAxes	Number of gimballing axes	
15	Safety?	Special safety requirement (b)	
16	Handling?	Special handling requirement (b)	
17	TestStand	Test stand code	X
18	FacilitMod	Level of facility modifications	
19	TotalCost	Total cost of tests	X

3.1. Testing and Training Sets Selection

When developing an ANFIS or neural network a set of data for training and a separate set of data for testing must be chosen. Theoretically, as long as the entire data set is large, the set could be divided randomly into training and testing. However, in practice it is important to make sure that the training set chosen is representative of all the data, including the current testing set and all future data that will enter the system. This is exceedingly difficult for the small set of 11 articles used for this project. Since the set is so small, the outcome is heavily dependent on which articles are chosen. Therefore, the strategy that was used was to choose the training set having the largest range of cost, from the largest to the smallest. This is not necessarily the optimal choice, however, taking too much care to manipulate the choice of the training and testing sets can lead to misleading results.

The other important concern when choosing training and testing sets is to determine the actual size of each, respectively. Normally, neural networks perform better when the training set is large. This is not possible with the small data set used here. Different sizes of training and testing sets were tried. The majority of the work was done using a training set of six or seven and a testing set of five or four.

3.2. Data Analysis

As mentioned earlier in this section, only certain data variables were determined to have the most predictive power for the engine test cost. These were primarily chosen from variables that were considered to be the most predictive by using exhaustive and sequential searches. Both searches were performed in Matlab with the commands *exhsrch* and *seqsrch* to determine which input variables have the most predictive power for ANFIS modeling. The exhaustive and sequential searches operate by searching for the minimum training error for different permutations of inputs to the ANFIS. The exhaustive search calculates all possible permutations of inputs, creating an ANFIS for all input combinations. The sequential search observes the ANFIS results for each input candidate independently. These searches are important because the dimensionality of our data set was too large compared to the size of the data set. However, it is important to only eliminate the variables that were the least predictive for the total engine test cost. The exhaustive search yielded the best results, however, was considerably more time consuming than the sequential search, and cannot deal with more than four variables simultaneously.

The entire data set, number of membership functions, and number of variables that were used in the ANFIS were inputs to these search functions. The accuracy of these search functions was probably somewhat compromised due to the small size of our data set. However, these searches were the best estimate we could obtain to the predictive power of variables without attaining more data sets.

Tables 3.2 through 3.5 contain all of the results from the searches. Different numbers of inputs into the ANFIS were chosen, varying from one input to four inputs. The single input searches have four variables listed because after each search was performed, the most predictive variable was removed from the set, then another search was performed. The inputs varying from 2 to 4 were simply searched once for each membership function number. A search for membership function sizes varying from 2 to 5 was performed in almost all cases. The only case where a membership function of size 5 was not performed was for the case with four inputs. The searches for this case became too lengthy and would not complete. For the single input variable case, both the exhaustive and sequential searches produced the same results (Table 3.2). However, for the multiple variable cases this is not so. In fact, the sequential search also has a tendency to choose redundant variables, as seen in Table 3.3-5 with Thrust and various other variables. For the single input ANFIS, all of the variables found to be relevant by these searches were used. For the multiple input ANFIS, some of the variables found to be predictive were used, however, not all were. Mostly the single input variables found to be predictive were used throughout the various methods.

Table 3.2: Variable Importance for a Single Input

Exhaustive Search				
MF #	**1st**	**2nd**	**3rd**	**4th**
2	Thrust	OxidizerFL	FuelFlow	TestStand
3	TestStand	TestDurMax	PressuraPa	Pressurant
4	TestStand	TestDurMax	DuratDd	Pressurant
5	TestStand	OxidizerFL	TestDurMax	DuratDd
Sequential Search				
MF #	**1st**	**2nd**	**3rd**	**4th**
2	Thrust	OxidizerFL	FuelFlow	TestStand
3	TestStand	TestDurMax	PressuraPa	Pressurant
4	TestStand	TestDurMax	DuratDd	Pressurant
5	TestStand	OxidizerFL	TestDurMax	DuratDd

Table 3.3: Variable Importance for 2 Inputs

Exhaustive Search		
MF #	**1**	**2**
2	DuratDd	TestDurMax
3	TestDurMax	Pressurant
4	TestDurMax	FuelPresFL
5	TestDurMax	Pressurant
Sequential Search		
MF #	**1**	**2**
2	Thrust	Thrust
3	TestDurMax	TestStand
4	TestDurMax	TestStand
5	TestDurMax	TestStand

Table 3.4: Variable Importance for 3 Inputs

Exhaustive Search			
MF #	1	2	3
2	DuradDd	Thrustmeas	Cooling?
3	NoTest	Fuel	TestStand
4	Fuel	Pressurant	TestStand
5	Fuel	Pressurant	TestStand
Sequential Search			
MF #	**1**	**2**	**3**
2	Thrust	Thrust	Thrust
3	TestDurMax	Fuel	TestStand
4	TestDurMax	GimbalAxes	TestStand
5	TestDurMax	TestStand	FacilitMod

Table 3.5: Variable Importance for 4 Inputs

Exhaustive Search				
MF #	1	2	3	4
2	NoTest	Pressurant	Oxydizer	Cooling?
3	Fuel	Pressurant	Oxydizer	TestStand
4	Fuel	Pressurant	Oxydizer	TestStand
Sequential Search				
MF #	**1**	**2**	**3**	**4**
2	Thrust	Thrust	Thrust	Thrust
3	TestDurMax	Fuel	Fuel	TestStand
4	TestDurMax	GimbalAxes	GimbalAxes	TestStand

3.3. Normalization of Data

All of the data was normalized before it was used in the ANFIS and neural network systems. The maximum of each respective data variable was found and increased by 10% ($x_{\max}$). The minimum of each respective data variable was also found and reduced by 10% ($x_{\min}$). These percentages were used to accommodate any future data outside the range already presented into the training set. These values were then used to normalize each data variable including the total cost using the following formula (3.1).

$$\bar{x}_{norm} = \frac{x_{orig} - x_{\min}}{x_{\max} - x_{\min}} \qquad (3.1)$$

This normalization caused all of the values of the normalized $\bar{x}_{norm}$ to be between zero and one. To attain the un-normalized predicted cost the inverse of equation 3.1 is used. The only exception to this method of normalization was any of the variables that included codes, for example TestStand. These variables were normalized according to the possible values to account for any instances that were not included in the testing data set.

4. ANFIS

The adaptive network-based fuzzy inference systems (ANFIS) were first developed by Jang and Sun [12-13], to take advantage of the best attributes from neural networks and fuzzy systems. ANFIS is a fuzzy inference system (FIS) that uses neural network algorithms to adapt itself in order to achieve better results. The direct advantage that it has over neural networks is that it can also accept linguistic information and adapt itself using numerical data [12].

The basic structure of any fuzzy inference system includes a fuzzification interface, a rule base, a database that defines the membership functions used in the rules, a fuzzy reasoning method that performs the inference procedure, and a de-fuzzification interface. A membership function provides a measure of an input's similarity to the fuzzy set [14]. For example, in Figure 4.1 these two membership functions could represent a fuzzy model for determining how cold or hot the temperature is outside. If the temperature is 80° F then it has a degree of membership of 0.3 cold and 0.5 hot. This is how a membership function defines inputs. An ANFIS uses neural network training techniques to adjust the membership functions. The rule base is usually in the form of fuzzy if-then rules. Using this example, a possible fuzzy rule could be if the input has a higher degree of hotness, then turn on the air conditioner.

The Sugeno fuzzy inference system, proposed by Takagi and Sugeno [15-16] is a type of fuzzy system where only the input set is considered fuzzy, i.e., the output of a Sugeno fuzzy inference system is not fuzzy, instead it is crisp. In Matlab, for a Sugeno system, the final output membership function is either constant or linear. In the problem dealt with in this research, a linear output membership function was always used.

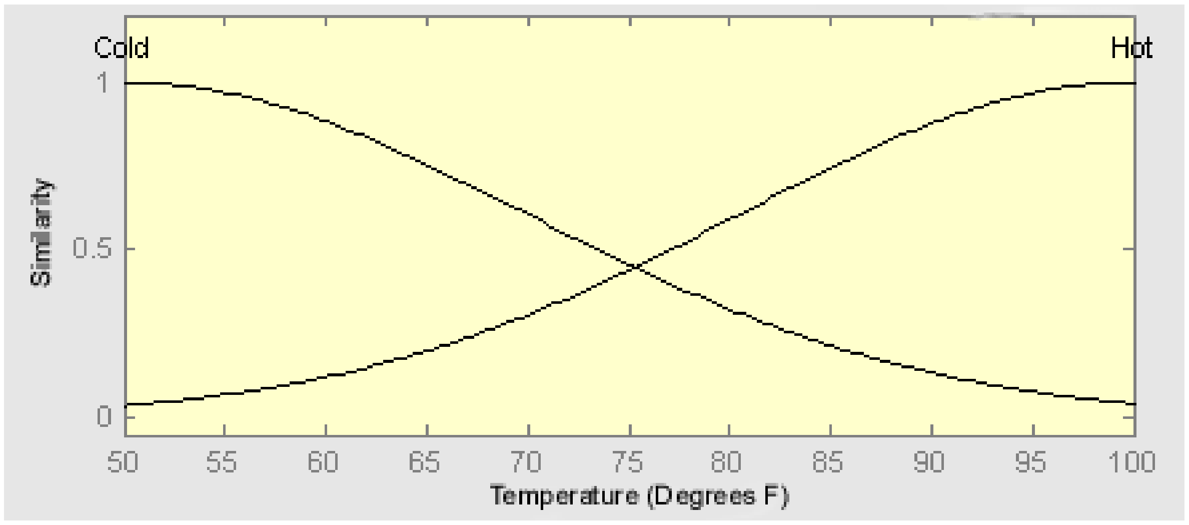
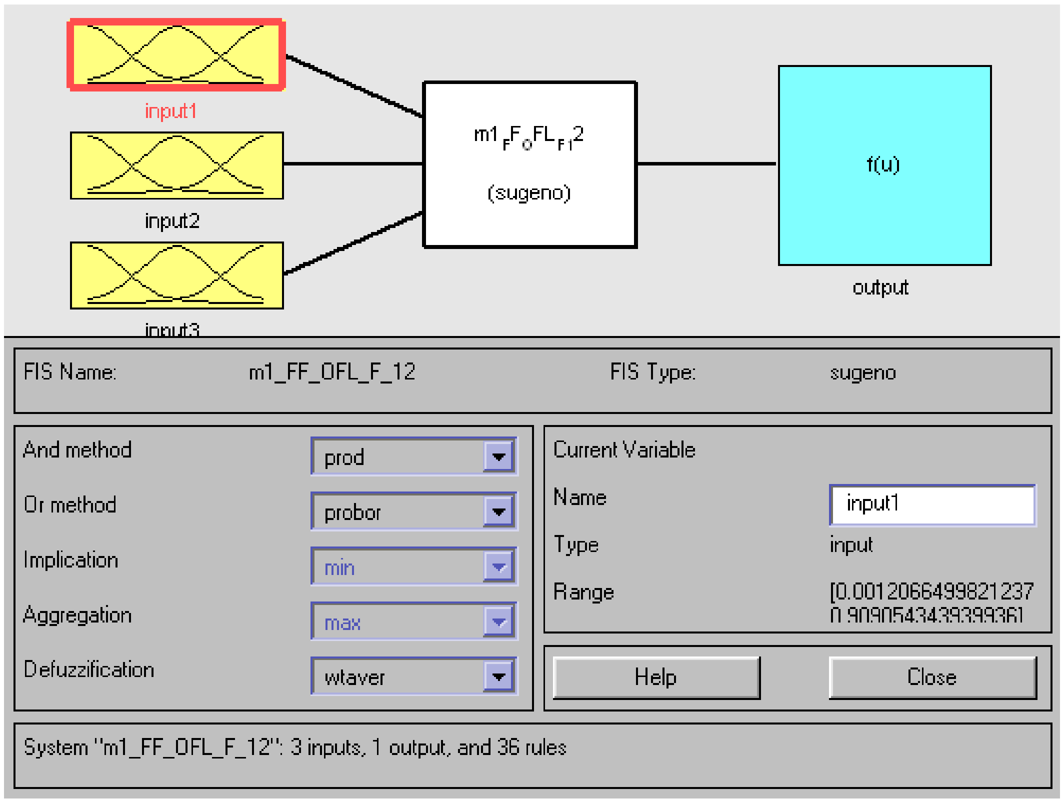


Figure 4.1: Example of Membership Functions in a Temperature System

Matlab's implementation of ANFIS uses a Sugeno-type fuzzy inference system whose parameters are trained using an adaptive neural network technique. The membership function parameters of the FIS are adjusted by a combination of a back propagation gradient descent algorithm and least squares method. The input training data consists of a matrix of the training data with the last column consisting of the target output data. Test input data is entered in the same way. The user must manually provide the number of inputs. While the fuzzy toolbox provides many membership function types, the Gaussian type membership functions always performed the best for the engine test data, namely *gaussmf* and *gauss2mf*. Figure 4.2 illustrates the block diagram of one FIS used in the engine test cost estimation in method 2B. The three inputs enter the ANFIS and are then processed to predict the cost.



FIS Name: m1_FF_OFL_F_12 FIS Type: sugeno

And method: prod
Or method: probor
Implication: min
Aggregation: max
Defuzzification: wtaver

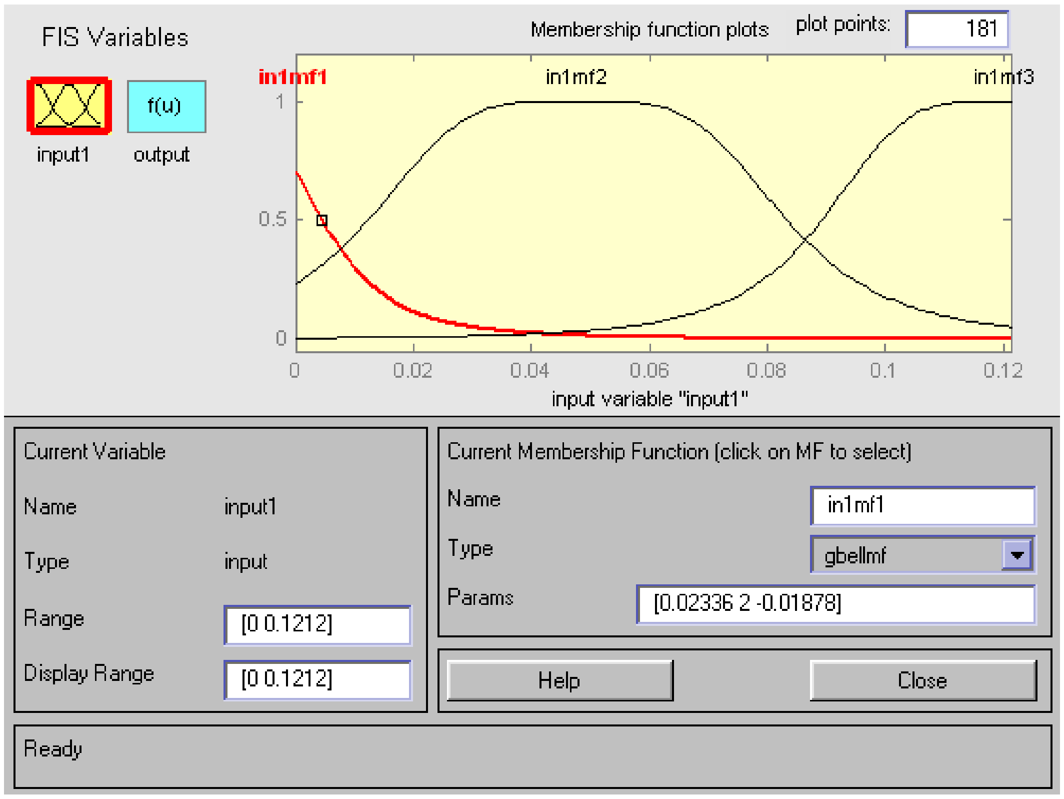
Current Variable
Name: input1
Type: input
Range: [0.0012066499821237 0.909054343939936]

System "m1_FF_OFL_F_12": 3 inputs, 1 output, and 36 rules

Figure 4.2: Block Diagram of a Method 2 Fuzzy Inference System

4.1. Membership Functions

Membership functions were developed for each data input. The number, shape, and overall range interval of membership functions were chosen according to the number of inputs. The most desirable number of membership functions was determined heuristically for each individual system. Given the small number of inputs, no more than five and no less than two membership functions per input were used. The best shape was also determined heuristically for each system with the best results always attained from the Gaussian types (*gaussmf* and *gauss2mf*). The range interval value was determined from the range of data plus or minus ten percent, as discussed in the normalization section 3.3. Figure 4.3 shows the membership functions of one FIS developed for the PressuraPr variable used to predict the engine test cost in method 2A. This FIS has a single input variable with three Gaussian membership functions.



Current Variable
Name input1
Type input
Range [0 0.1212]
Display Range [0 0.1212]

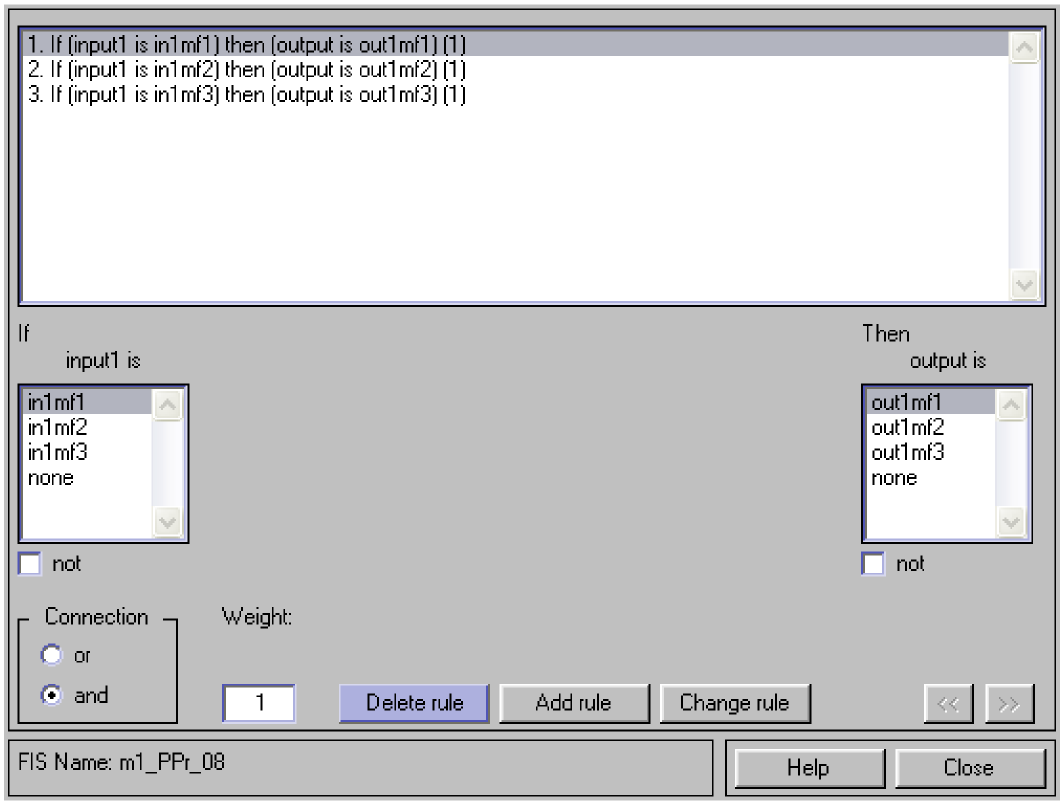
Current Membership Function (click on MF to select)
Name in1mf1
Type gbellmf
Params [0.02336 2 -0.01878]

Ready

Figure 4.3: Membership Functions for PressuraPr Variable (Method 2)

4.2. ANFIS Rules

The ANFIS rules were obtained from Matlab's Fuzzy Toolbox. While in the case of most fuzzy systems it is simple to derive logical rules directly, this ceases to be practical when dealing with an ANFIS. The neural network derives rules that are often not logical, however, optimal results are still obtained. In Figure 4.4 an example of ANFIS rules from the same PressuraPr FIS used in Figure 4.3 is shown. This is a relatively uncomplicated rule set because there is only one input. The top box shows the actual rules. The bottom boxes allow the user to individually alter each rule. A more visual representation of these rules is given in Figure 4.5. The figures labeled 1, 2, and 3 represent each rule. The red line represents a given input. Its effect on each rule is shown in yellow. The output column represents the output of each rule. The bottom right output plot demonstrates how the output of each rule is combined and defuzzified to form an output value.



```
1. If (input1 is in1mf1) then (output is out1mf1) (1)
2. If (input1 is in1mf2) then (output is out1mf2) (1)
3. If (input1 is in1mf3) then (output is out1mf3) (1)
```

If input1 is: [✓ in1mf1] in1mf2 in1mf3 none; not [ ]

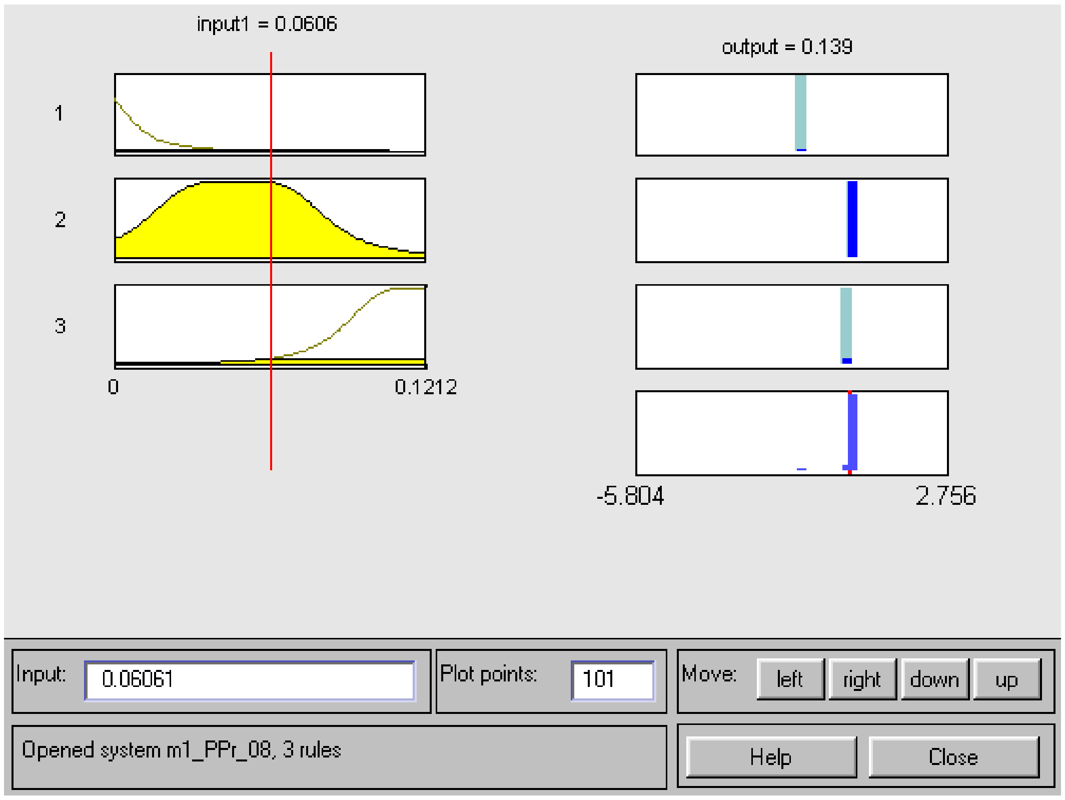
Then output is: [✓ out1mf1] out1mf2 out1mf3 none; not [ ]

Connection: or [✓ and]

Weight: 1

FIS Name: m1_PPr_08

Figure 4.4: ANFIS Rules for PressuraPr Variable (Method 2)



Input: 0.06061

Plot points: 101

Opened system m1_PPr_08, 3 rules

Figure 4.5: Graphical ANFIS Rules for PressuraPr Variable (Method 2)

4.3. ANFIS Structure

The ANFIS structure for the above PressuraPr variable example is show in Figure 4.6. The input goes into three membership functions where the ANFIS rules that were developed during training are applied. These rules produce three outputs that are then combined and defuzzified to produce a single output, the estimated engine test cost.

The ANFIS were also created using grid partitioning rather than clustering. Grid partitioning is superior to the clustering method because the data sets are so small. If the data sets were larger, clustering would be a better option for creating the FIS [2].

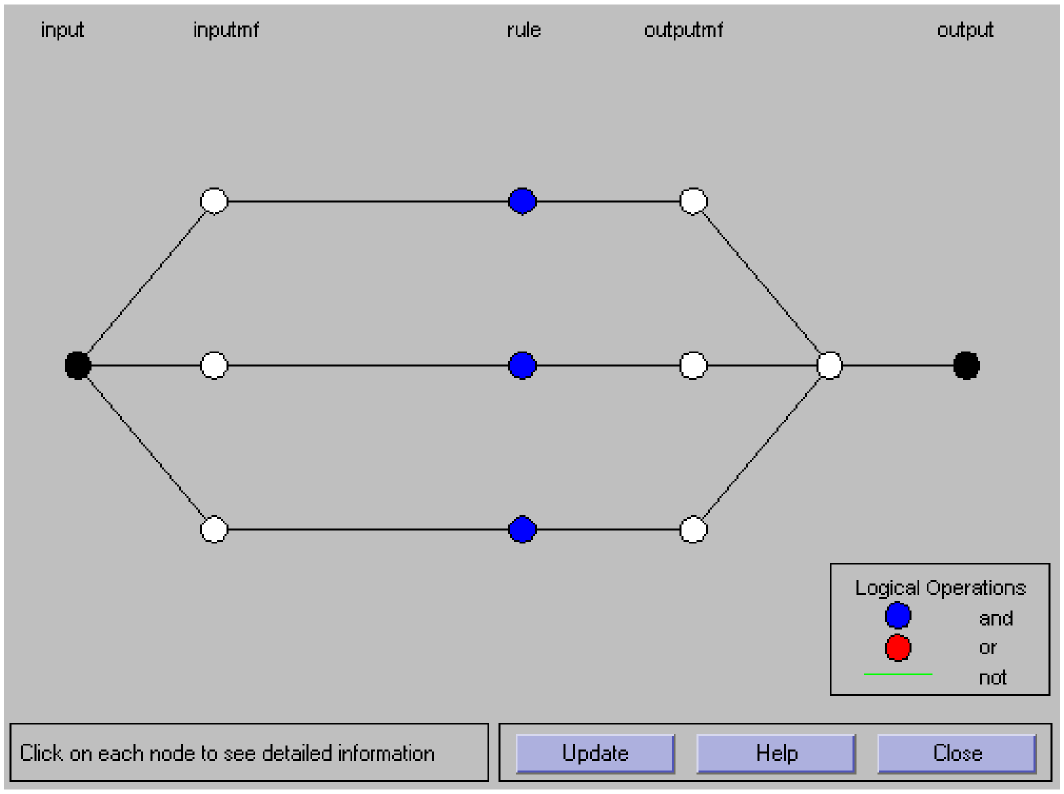


Figure 4.6: ANFIS Structure for PressuraPr Variable (Method 2)

5. METHOD 1: LLE

The first method implemented to solve the engine test cost estimation problem was to use locally linear embedding (LLE) as a nonlinear dimensionality reducer to condition the input data. Then, an ANFIS is used to predict the engine test cost based only on the reduced data. Figure 5.1 is a block diagram that visually represents this method. LLE, developed by Saul and Roweis [17-18], is a nonlinear dimensionality reduction method originally applied to image processing. LLE attempts to map the input data to a lower dimensional global coordinate system that preserves the relationships between neighboring points [4]. Locally linear neighborhoods of the input data are then mapped into a lower dimensional coordinate system.

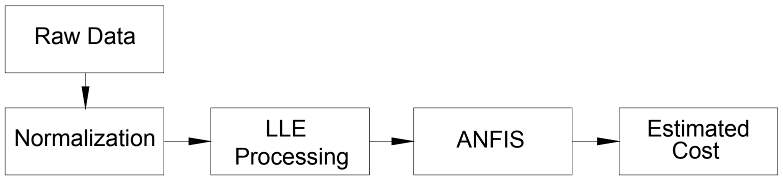


Figure 5.1: Block Diagram of Method 1: LLE

The LLE algorithm is divided into three steps: selecting neighbors; computation of weights that best reconstruct each data point by it's neighbors, and; mapping to embedded coordinates [5][17]. The first step simply involves finding K nearest neighbors for each data point. This can be accomplished using different methods including finding the Euclidean distances between respective points or finding all neighbors within a fixed radius.

The second step involves finding the weights that best reconstruct each data point. The data consists of N real-valued vectors $\vec{X}_i$, each of dimensionality D sampled from an underlying manifold. As long as there are enough sample points, it is expected that each data point lies on or close to a locally linear section on the manifold. The local area is then characterized by linear coefficients that reconstruct each data point from its neighbors. The reconstructed errors are then measured by the following cost function (5.1)

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2 \tag{5.1}$$

This cost function (5.1) adds up the squared distances between all of the data points and their reconstructions. The weights W_{ij} represent the contribution of the jth data point to the reconstruction of the ith data point. The weights are computed by minimizing the cost function on two conditions: 1) that each data point $\vec{X}_i$ is reconstructed only from its neighbors or, in other words, $W_{ij} = 0$ if $\vec{X}_j$ is not part of the set of neighbors of $\vec{X}_i$, and; 2) that the cost function is minimized so that the rows of W_{ij} sum to one. For any particular data point, these weights are invariant to rotations, rescalings, and translations of that data point from its neighbors, meaning these weights reflect intrinsic geometric properties of each neighborhood. It follows that the weights characterization of the local geometry in the original data space is also valid for local patches on the manifold [18].

The final step in the LLE algorithm is mapping the data set to the new lower dimensional space coordinates. Each high dimensional data point $\vec{X}_i$ is mapped to the lower dimensional vector $\vec{Y}_i$ representing the embedding coordinates. The embedding coordinates, $\vec{Y}_i$, are obtained by once again minimizing an embedding cost function (5.2).

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2 \tag{5.2}$$

Again, as with the previous function (5.1), this cost (5.2) is based on locally linear reconstruction errors, but the weights are now fixed while $\vec{Y}_i$ is optimized. This cost function can be manipulated into a quadratic form, which can be minimized by solving a sparse NxN eigenvalue problem whose bottom *d* non-zero eigenvectors provide the set of orthogonal coordinates centered on the origin, where *d* is the desired reduced dimension size [17]. In other words, $\vec{Y}_i$ is equal to these eigenvectors.

5.1. Implementation of Method 1

In Method 1, the LLE algorithm was used to reduce the dimensionality of the data set. The new, smaller dimensioned data set was then put into an ANFIS to finally predict the cost of the engine test. The weakness of this method lies in the LLE algorithm's reliance on the k nearest neighbor algorithm during the first step, which was difficult to accomplish due to the small data set we utilized.

Saul and Roweis provide the Matlab code that implements the locally linear embedding algorithm [18]. This code uses the Matlab eigensolver *eigs*, which unfortunately has convergence issues in version R12. In order to circumvent this problem, we used the version of *eigs* in R11 is used instead. This version of *eigs* was saved as a different function, *eigs_o*, and replaced in the LLE code. The code *lle.m* is included in Appendix 12.4. The LLE code simply required the data matrix in the form of the dimensionality by the number of data points, the K number of neighbors desired, and the reduced dimensionality desired. With these inputs, the LLE code performed the LLE algorithm described previously.

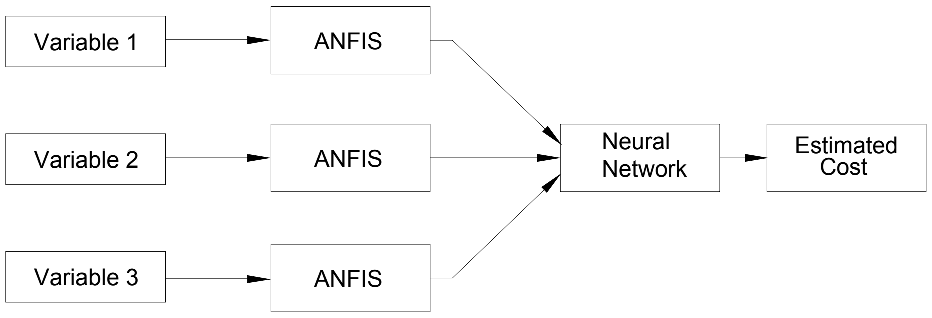
The ANFIS was then developed from the new lower dimensional data set that was transformed by the LLE code, using grid partitioning and a linear output membership function type. Several trials were performed to develop the best ANFIS by varying the number and type of membership functions. The best results of these trials are presented in section 8.1.

6. METHOD 2: ANFIS/NEURAL COMBINED SYSTEMS

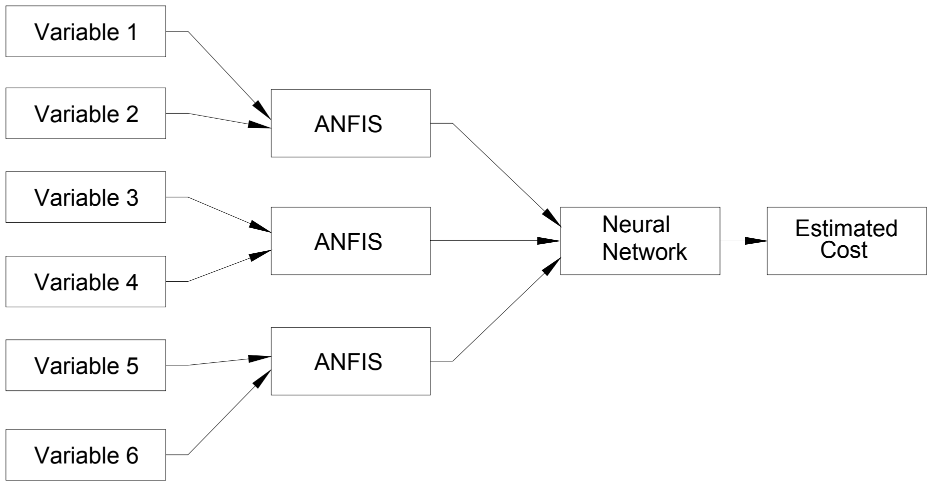
The second method implemented to estimate engine test cost consists of a combined ANFIS and neural network system. The data is either entered singly or in small groups into separate ANFIS. The outputs of these ANFIS are then combined into a single neural network to produce the final result. Figure 6.1 illustrates these methods visually. Method 2A has a single input variable into each individually developed ANFIS. The outputs of these ANFIS are then combined into a neural network. Method 2B is the same as Method 2A, except that multiple inputs are provided to each ANFIS network. Several different ANFIS were developed for this method.

The neural networks used for this method were feed-forward backpropagation trained networks created using the *newff* command in Matlab. The Matlab commands *traingdx* and l*earngdm* was chosen for the training and learning function of the network, respectively. These functions train the network using gradient descent with momentum and an adaptive learning rate. This means that, for each epoch, if the performance decreases towards the goal, then the learning rate is increased by a factor, however if the performance increases more than a certain factor, the learning rate is decreased and the change is not made. The error criteria used was sum of squared errors.

A two-layer network was used. More complicated networks were not practical due to the size of the data sets. The number of neurons in the input layer of the network was always set equal to the number of inputs to the network. The output layer consisted of a single neuron. The transfer functions of each layer were either *tansig* or *logsig*. Figure 6.2 shows both the *tansig* and *logsig* functions that were used as the transfer functions of the neurons. Figure 6.3 shows a typical neural network developed for this method.



(a) Method 2A: Single Input to ANFIS, Combined into Neural Network



(b) Method 2B: Multiple Inputs to ANFIS, Combined into Neural Network

Figure 6.1: Block diagrams of Method 2A and Method 2B

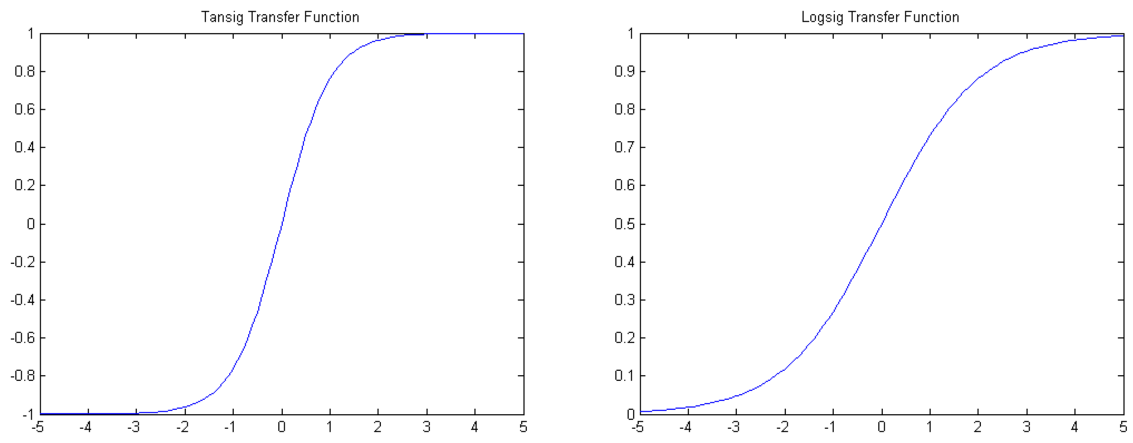


Figure 6.2: Graphs of Tansig and Logsig Transfer Functions

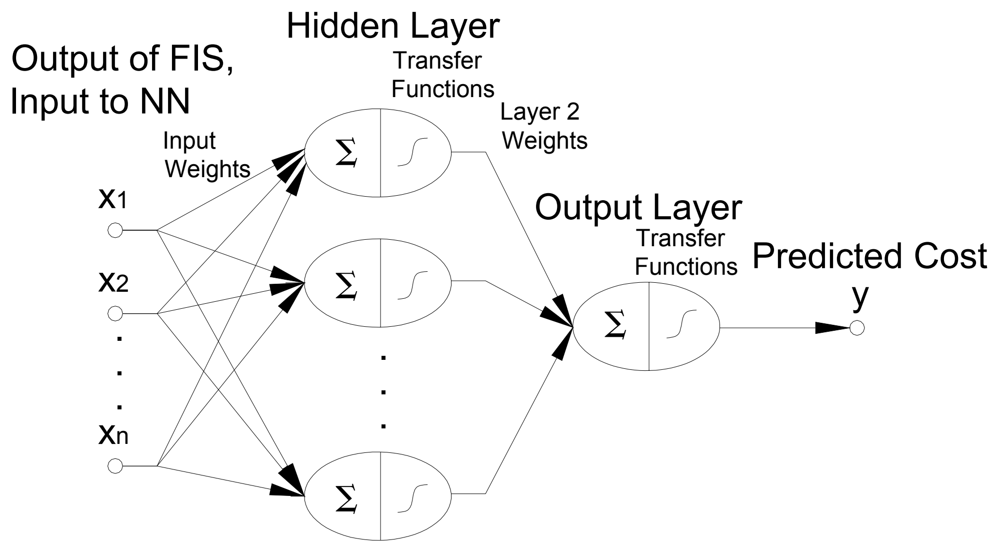


Figure 6.3: Typical Method 2 Neural Network

6.1. Method 2A: Single Input ANFIS

As shown in Figure 6.1a, this method involves individual variables to be used by individual ANFIS, respectively. These ANFIS outputs then provide an input for the neural network, which provides the final cost estimate. The variables used to develop the single ANFIS were found to be most predictive through the use of exhaustive and sequential searches, as described in section 3.2. Table 6.1 contains the list of variables used as single inputs to the ANFIS. Some of these variables were not used in the final neural stage because they were found to create inaccurate ANFIS. Various combinations of ANFIS results were used to find the best results from the neural network. The results of the ANFIS and neural networks are presented in section 8.2.

Table 6.1: List of Data Variable Inputs to ANFIS for Method 2A

	Variable	Description
1	DuratDd	Duration of tests in days
2	TestDurMax	Maximum duration of test
3	Pressurant	Pressurant code
4	Thrust	Thrust
5	FuelFlow	Fuel flow rate
6	OxidizerFL	Oxidizer flow rate
7	PressuraPr	Pressurant pressure
8	TestStand	Test stand code
9	TotalCost	Total cost of tests

6.2. Method 2B: Multiple Input ANFIS

In Method 2B, ANFIS were developed using two and three input variables. These ANFIS estimates were then combined into a neural network whose output is the final cost estimate for the engine test. The variables combined together were chosen by examining the response from the single input ANFIS. Variables that would compliment each other to attain more accuracy were placed as inputs into the same ANFIS. For example, if one variable always overestimated the cost in the single input ANFIS and another always underestimated the cost, they would be combined into a single ANFIS. A similar method was also used to choose the combination of variables for the three input ANFIS. Table 6.2 is a list of some of the combinations of inputs that were used for Method 2B. Varying numbers of inputs into the neural network were tried.

Table 6.2: List of Data Variable Input Combinations to ANFIS for Method 2B

	Variable 1	Variable 2	Variable 3
1	Thrust	TestStand	-
2	TestDurMax	PressuraPr	-
3	FuelFlow	OxydizerFL	-
4	DuratDd	Pressurant	-
5	FuelFlow	Thrust	-
6	Thrust	TestStand	DuratDd
7	TestDurMax	PressuraPr	Pressurant
8	FuelFlow	OxydizerFL	Fuel

Some of the data variables that have been used have many missing values, namely Pressurant, FuelFlow, and PressuraPr. In an effort to attain more accuracy, these values were filled in with various values (mean, median, or mode) then the results of the respective combined input ANFIS were compared to the ANFIS with no filled in values. The number used to fill in for missing Pressurant values was found by going back to the original data set of 38 and finding the most often occurring value, 2. This is more logical than taking the mean or median of the set, since Pressurant is a code referring to a type of pressurant. Unfortunately, Pressurant was only predictive in the three input variable systems. It was not predictive with or without filled in values in the two variable input systems even when paired with the most predictive quantities like Thrust. However, for both FuelFlow and PressuraPr, better results were obtained by using the filled in data value rather than leaving the missing information blank. Both the mean and median value of FuelFlow in the set of eleven was used to see which provided a more accurate result. The median value looked like a better choice since FuelFlow had a large range of values, however, the mean value always performed better. For PressuraPr, the median value provided the best results, and was chosen since PressuraPr also had a very large spread of values. All of the results for the systems are presented in section 8.

7. METHOD 3: RBFN

The third method implemented to estimate engine test costs is a radial basis function neural network (RBFN). RBFNs are similar to ANFIS in that they consist of membership functions that are adjusted through the training stage of a neural network. They typically consist of Gaussian-type membership functions. First, the centers of the Gaussian functions are found using a k-means clustering algorithm on the training data. In this case, the k-means algorithm groups the data sets into clusters, so that costs are associated with each cluster. After the clusters are found, the p-nearest neighboring clusters are found. The variance of each membership function is then found using the p-nearest neighbors. Equation 7.1 is the formula used to determine the variance of each cluster. The respective centers are represented by c_k, where k represents the cluster of interest. The variance is represented by σ_k and p represents the p-nearest neighbors.

$$\sigma_k = \sqrt{\frac{1}{p}\sum_{i=1}^{p}\left\|c_k - c_{ki}\right\|^2} \qquad (7.1)$$

After the variance of each cluster is determined, the training and testing sets can then be classified into the appropriate membership function. As with the ANFIS system, a percentage of belonging to each membership function is obtained. This is done to each data set with the following formula 7.2. The degree of belonging to each cluster is represented by ϕ_k and x represents each data point.

$$\phi_k = \exp\left(\frac{-\|x - c_k\|}{\sigma_k^2}\right) \tag{7.2}$$

After this percentage of belonging to each membership function is calculated, it is then normalized so that the percentages sum to one. Each percentage is multiplied by the calculated average cost of each cluster, and then summed into a single value. This final value is the predicted cost out of the RBF network. Figure 7.1 illustrates the RBF network used in method 3.

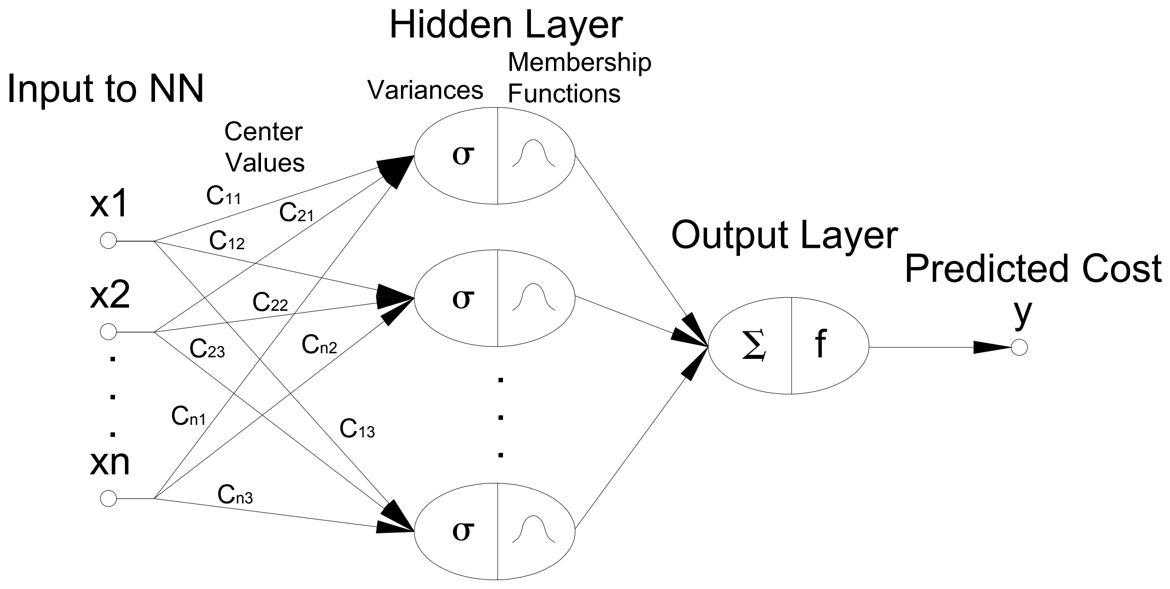


Figure 7.1: Method 3 Radial Basis Function Network

The RBFN was implemented in two different ways. The first implementation used the entire data set of eleven to train the network. This is not the correct way to build a network since part of the data set should be saved for testing, so that the network has never seen the testing set. However, because of the small size of the data set, k-means clustering is difficult. Using the entire data set of eleven allows for a better clustering. These results were then compared to the second and correct implementation where the set of eleven is divided into seven training and four testing sets. Both of the RBFNs were developed with the variables with missing data filled in as in Method 2B. The results of the RBFN are presented in section 8.3.

8. RESULTS

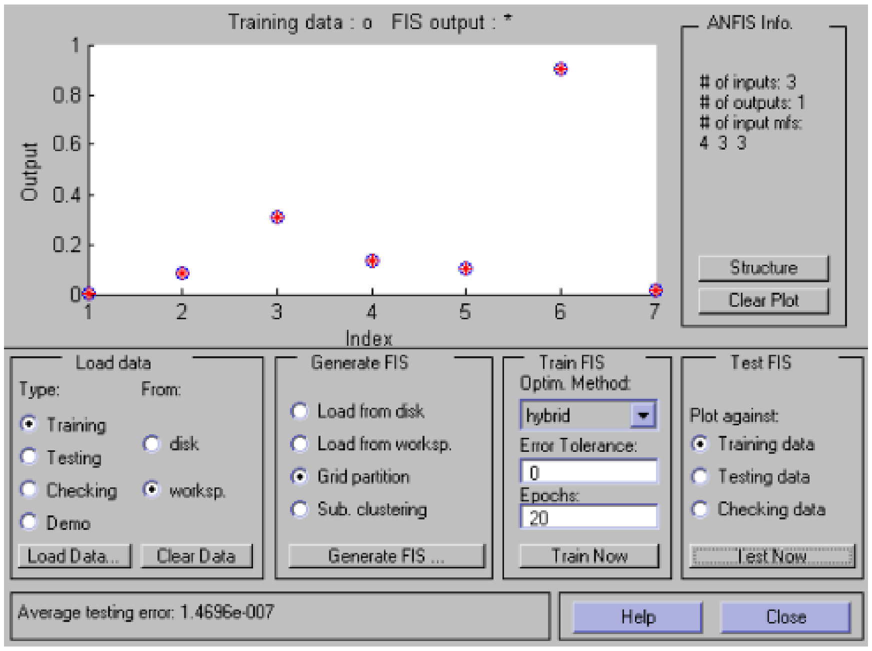
8.1. Results for Method 1: LLE

To implement the LLE method, all of the data was first normalized, as discussed in section 3.3. A smaller subset of the most predictive variables was chosen out of the eighteen. The best results were obtained using a set of eight variables shown in table 8.1. Using the full data set of dimension eighteen produced worse results as expected. It would be difficult to find a proper lower dimensional space that corresponded to the eighteen-dimensional space with such few data points. Different numbers of k clusters and d dimensions was chosen, however a k of three and a final dimension of three attained the best results.

Table 8.1: Method 1: LLE Variables

	Variable
1	DuratDd
2	TestDurMax
3	Pressurant
4	Thrust
5	FuelFlow
6	OxidizerFl
7	PressuraPr
8	TestStand

After the dimensionality of the data set was reduced, the new lower dimensional data set was then input into an ANFIS as described in section 5. Different membership function types and different numbers of membership functions were tried. The best results were obtained with a *gauss2mf* membership function of size 4, 3, and 3, for each variable respectively. This means that for the first input dimension 4 *gauss2mf* membership functions were used, for the second 3 *gauss2mf* membership functions were used, and for the third dimension 3 *gauss2mf* membership functions were used. Figures 8.1 and 8.2 show the training and testing results for the best ANFIS developed, respectively. As can be seen in figure 8.1, the ANFIS learned the training set very well, with a negligible amount of error. These results are still normalized, the un-normalized test result will be shown later. For the remainder of the ANFIS results presented blue will refer to the actual cost and red will refer to the predicted cost.



ANFIS Info.

# of inputs: 3
# of outputs: 1
# of input mfs:
4 3 3

Load data
Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]

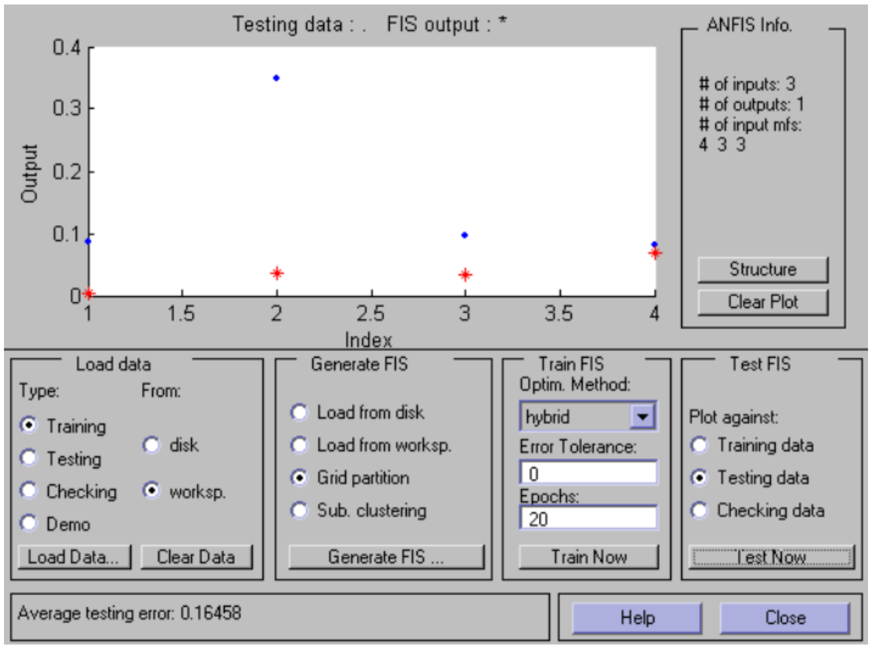
Generate FIS
Load from disk
Load from worksp.
[✓ Grid partition]
Sub. clustering

Train FIS
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 20

Test FIS
Plot against: [✓ Training data] Testing data Checking data

Average testing error: 1.4696e-007

Figure 8.1: Method 1: LLE Training Set



# of inputs: 3
# of outputs: 1
# of input mfs:
4 3 3

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]

Load from disk
Load from worksp.
[✓ Grid partition]
Sub. clustering

Optim. Method: hybrid
Error Tolerance: 0
Epochs: 20

Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.16458

Figure 8.2: Method 1: LLE Testing Set

The results using the LLE method to reduce the dimensionality were not promising. While the training results were good, the testing results were extremely inaccurate. The best results attained still have an average percentage error of around 66%. The first two testing set articles are both estimated at 90% lower than they actually are. Table 8.2 contains the actual costs attained, percentage errors, average percent error, and another measure of similarity defined in equation 8.1. This measure of similarity was developed since percent error is not necessarily the best measure to compare the values. If testing articles 1 and 2 are examined, their respective percent errors are very similar. However, by article 2 the cost estimate was approximately four million dollars too low, while article 1 was estimated approximately one million dollars too low. Article 1's under estimation was more acceptable than article 2's. Therefore, the sum of the total differences divided by the number of total sets (Eq. 8.1) is used as another measure of error. N represents the total number of sets, x_i represents actual cost, and $\hat{x}_i$ represents the estimated cost.

$$Sum\ Diff./Total\ \#of\ Sets = \frac{1}{N}\sum_{i}^{N}\left|x_i - \hat{x}_i\right| \tag{8.1}$$

The smaller the sum of total differences quantity, the closer the estimation is to the actual value. This is clearly seen in Table 8.2 where the training set's sum of differences divided is zero compared to the larger value for the testing set.

Table 8.2: Method 1: LLE Training and Testing Results

Training Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	550,000	550,000	1.05E-04
2	1,592,000	1,592,000	-9.02E-05
3	4,433,000	4,433,000	-8.69E-05
4	2,203,000	2,203,000	-2.17E-05
5	1,772,000	1,772,000	-4.99E-05
6	12,051,000	12,051,000	-2.92E-05
7	702,000	702,000	-2.06E-05
			Average (%)
			5.77E-05
			Sum of Diff./Total Number of Sets
			0

Testing Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	534,040	1,590,000	-96.44
2	948,360	4,935,000	-89.79
3	933,940	1,713,000	-63.96
4	1,370,900	1,540,966	-16.26
			Average (%)
			66.61
			Sum of Diff./Total Number of Sets
			1,497,932

A graphical presentation of the method 1 testing results is shown in Figure 8.3. These are the un-normalized values of the predicted and actual costs. These results clearly indicate that using LLE as a dimensionality reduction method is not a viable method for the engine test cost prediction. The only article that predicted with some degree of accuracy is article 4. All testing sets estimated lower than the actual cost. This most likely occurred because the data set was too small for the LLE method to find a valid lower dimensional coordinate system.

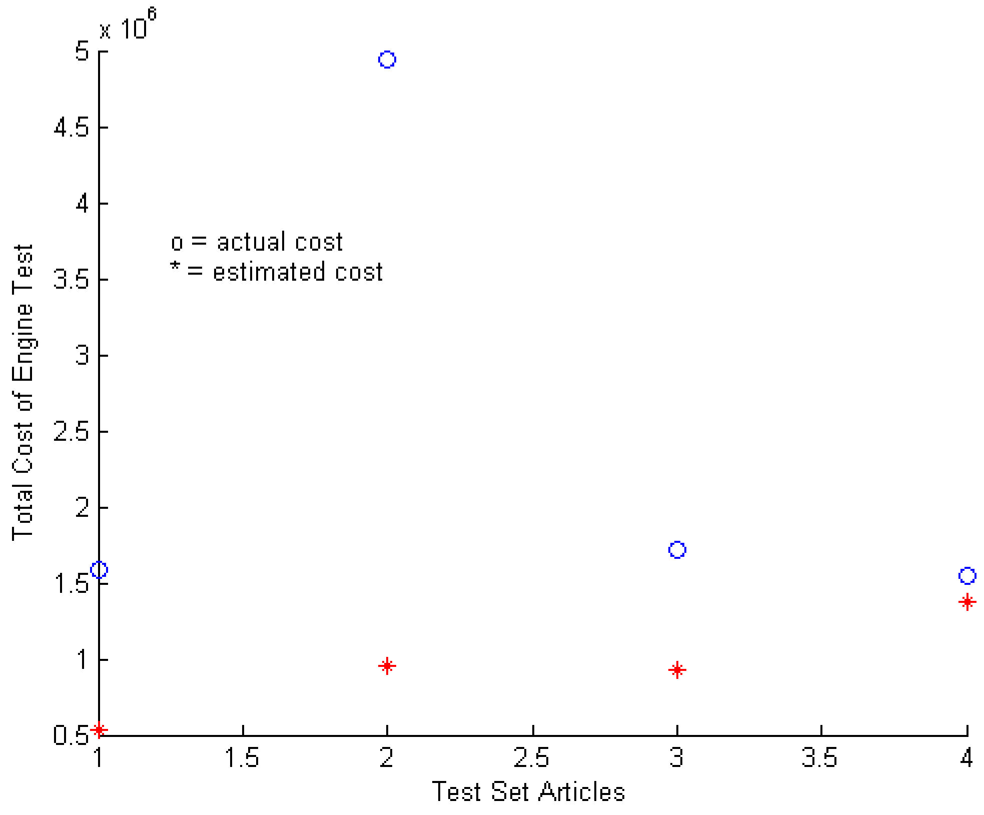


Figure 8.3: Method 1: LLE Testing Data Results

8.2. Results for Method 2: ANFIS/Neural Combined System

8.2.1. Method 2A

Method 2A used ANFIS with single inputs to produce an estimate. Several of these single input ANFIS outputs were then combined into a single backpropagation trained neural network whose output represents the final predicted cost. These single input ANFIS produced only very rough estimates of the predicted engine test, because the cost estimate was derived from only one single variable. Table 8.3 shows the actual and predicted normalized costs of each ANFIS developed for both the training and testing set. The columns with a variable name above them are the predicted cost that the ANFIS with that particular input produced. These numbers are presented in a normalized form, because they will become the inputs to the neural network in the next stage. Also, the average percent error along with the sum of differences quantity from equation 8.1 attained for each ANFIS developed is presented in Table 8.3. Figures 8.4 through 8.10 are the testing results of most of the single input ANFIS developed. The membership function types are also listed for each ANFIS developed.

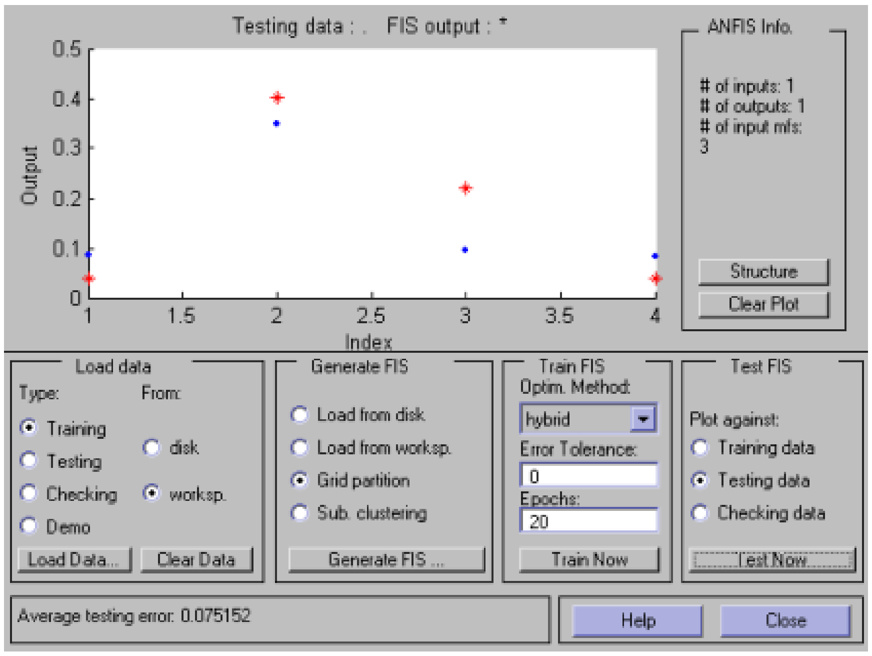
Table 8.3: Method 2A: Single Input ANFIS Training and Testing Results

	Training Set								
	TestStand	Thrust	FuelFlow	TestDurMax	PressuraPr	OxidizerFL	Pressurant	DuratDd	Actual Cost
1	0.040202	0.11681	0.075755	0.00431	0.022887	0.11279	0.0355	0.0043124	0.00431
2	0.085964	0.096306	0.17504	0.085964	0.080683	0.093042	0.0355	0.086098	0.085964
3	0.22122	0.12145	0.14126	0.16244	0.32635	0.11279	0.30859	0.30861	0.30859
4	0.22122	0.12354	0.12602	0.51967	0.13384	0.12324	0.13389	0.13371	0.13384
5	0.040202	0.077073	0.059087	0.10007	0.10007	0.093185	0.10007	0.10007	0.10007
6	0.90556	0.9056	0.90556	0.51967	0.89707	0.90556	0.90552	0.90554	0.90556
7	0.040202	0.11378	0.07185	0.16244	-0.006332	0.11396	0.0355	0.016219	0.016221
	Average Percent Errors								
	162.00	473.59	315.04	182.81	83.27	458.00	128.75	0.05	
	Sum of Differences/Total Number of Sets								
	0.0421	0.0630	0.0618	0.1520	0.0104	0.0609	0.0144	0.0000	

	Testing Set								
	TestStand	Thrust	FuelFlow	TestDurMax	PressuraPr	OxidizerFL	Pressurant	DuratDd	Actual Cost
1	0.040202	0.10332	0.059087	0	0.18928	0.094724	0.90552	0.11701	0.085808
2	0.40224	0.4455	0.36759	0.09858	0.27341	0.44799	0.11941	0.64972	0.34793
3	0.22122	0.12807	0.062714	0.099451	0.13925	0.133	0.90552	0.64972	0.095446
4	0.040202	0.10332	0.091077	0.057187	0.89707	0.11365	0.90552	0.73499	0.081965
	Average Percent Errors								
	62.87	27.17	20.55	52.68	295.59	29.29	718.62	375.13	
	Sum of Differences/Total Number of Sets								
	0.0669	0.0423	0.0221	0.0910	0.2592	0.0446	0.6705	0.3851	

Each variable chosen for these single input ANFIS proved relevant to the cost prediction in the exhaustive and sequential searches. Membership function types and numbers were changed for all ANFIS to determine which produced the best results. Once again, the best results were always obtained by using Gaussian type membership functions, either *gaussmf*, *gauss2mf*, or *gbell*. The number of membership functions was never allowed to be less than two or more than four because the system is so small that if more were included, the ANFIS would become overly complicated.

Not all of the ANFIS that were developed were used as inputs to the neural network. The variables Pressurant and DuratDd were eliminated due to their poor performance in the ANFIS . Figures 8.4, 8.5, 8.6, and 8.9 illustrate that the most predictive variables for the single input ANFIS are TestStand, Thrust, FuelFlow, and OxidizerFl.



# of inputs: 1
# of outputs: 1
# of input mfs: 3

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]

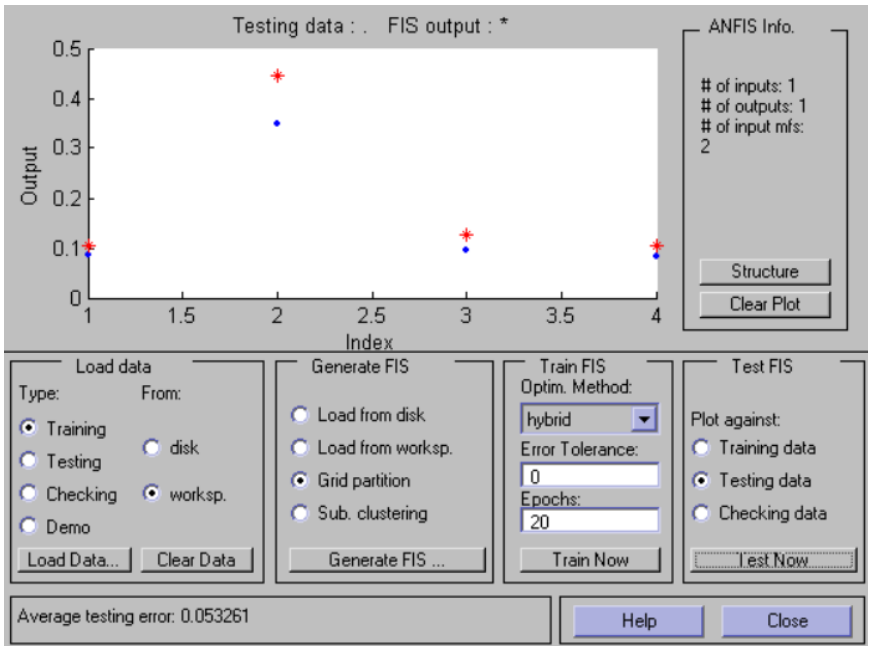
Load from disk | Load from worksp. | [✓ Grid partition] | Sub. clustering

Optim. Method: hybrid
Error Tolerance: 0
Epochs: 20

Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.075152

Figure 8.4: Single Input TestStand ANFIS Testing Results (gaussmf)



ANFIS Info.

# of inputs: 1
# of outputs: 1
# of input mfs:
2

Type: [✓ Training] Testing Checking Demo

From: disk [✓ worksp.]

Load from disk Load from worksp. [✓ Grid partition] Sub. clustering

Optim. Method: hybrid

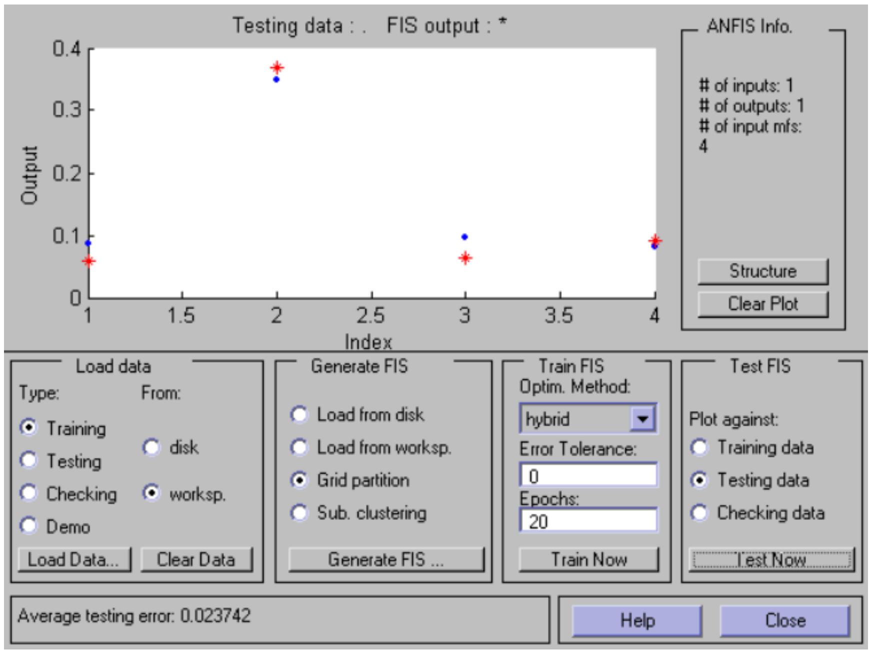
Error Tolerance: 0

Epochs: 20

Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.053261

Figure 8.5: Single Input Thrust ANFIS Testing Results (gaussmf)



ANFIS Info.

# of inputs: 1
# of outputs: 1
# of input mfs:
4

Type: [✓ Training] Testing Checking Demo

From: disk [✓ worksp.]

Load from disk Load from worksp. [✓ Grid partition] Sub. clustering

Optim. Method: hybrid

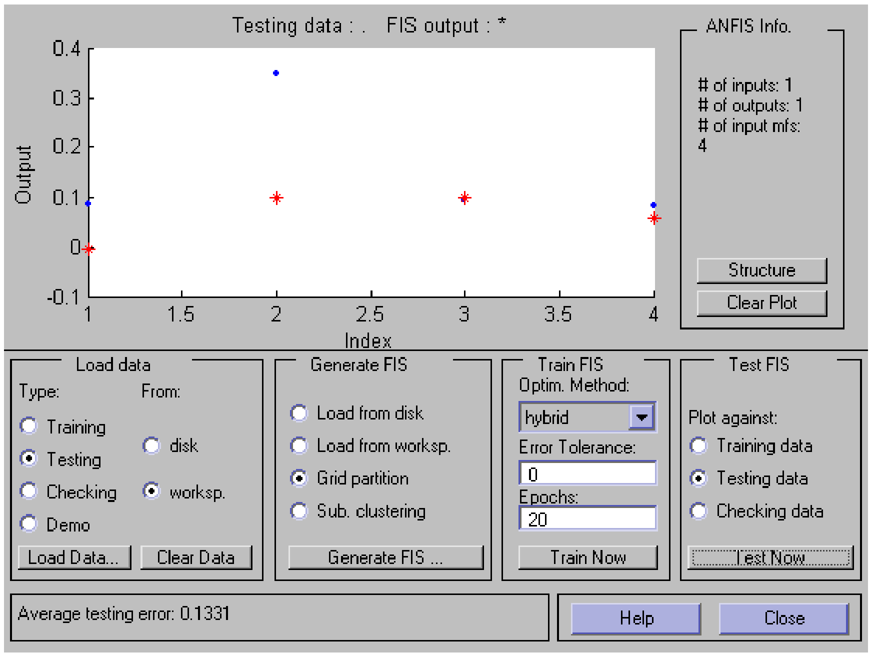
Error Tolerance: 0

Epochs: 20

Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.023742

Figure 8.6: Single Input FuelFlow ANFIS Testing Results (gaussmf)

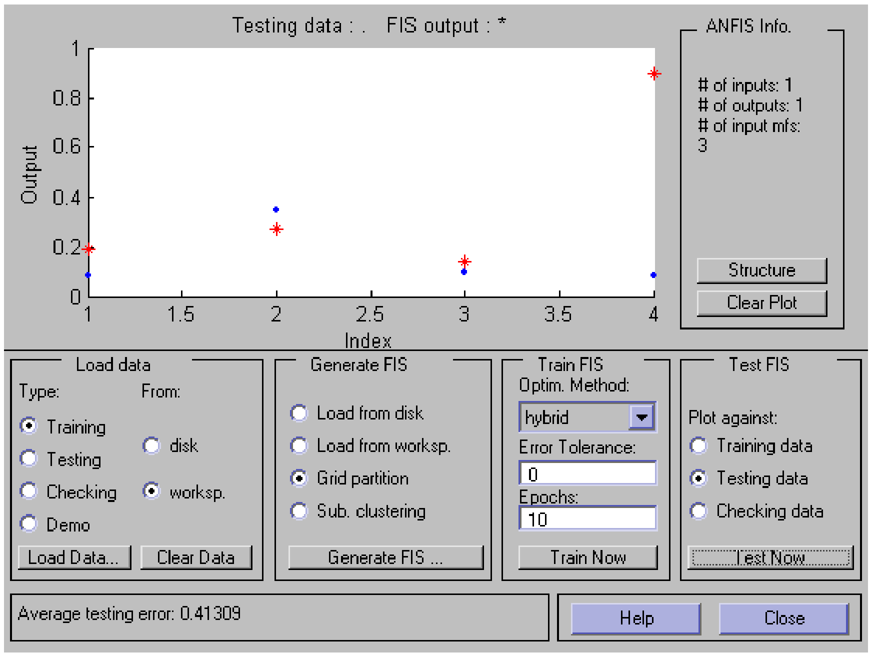


# of inputs: 1
# of outputs: 1
# of input mfs:
4

Type: Testing; From: worksp.; Grid partition; Optim. Method: hybrid; Error Tolerance: 0; Epochs: 20; Plot against: Testing data

Average testing error: 0.1331

Figure 8.7: Single Input TestDurMax ANFIS Testing Results (gauss2mf)

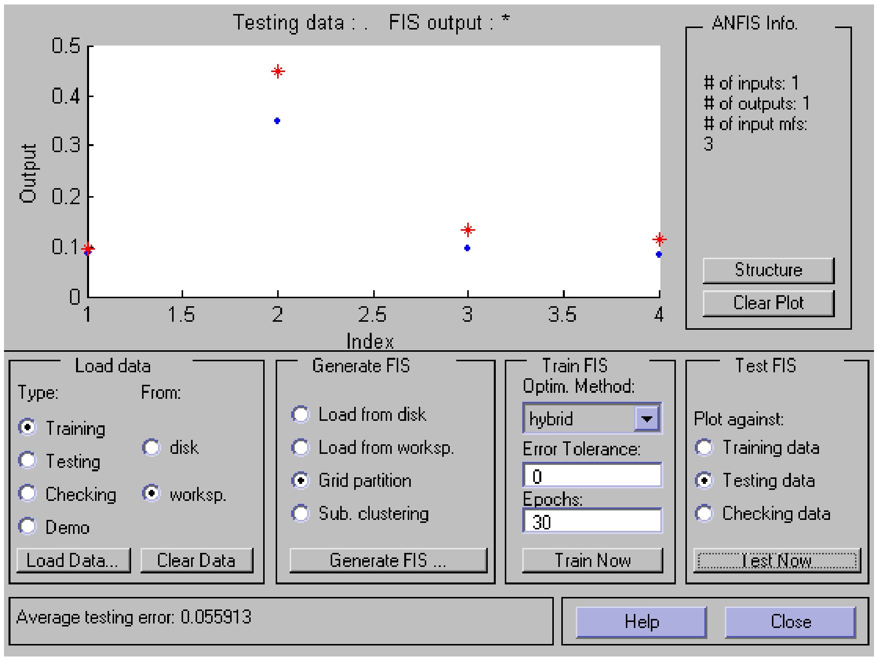


# of inputs: 1
# of outputs: 1
# of input mfs:
3

Type: Training; From: worksp.; Grid partition; Optim. Method: hybrid; Error Tolerance: 0; Epochs: 10; Plot against: Testing data

Average testing error: 0.41309

Figure 8.8: Single Input PressuraPr ANFIS Testing Results (gauss2mf)



ANFIS Info.: # of inputs: 1; # of outputs: 1; # of input mfs: 3

Load data — Type: [✓ Training] Testing Checking Demo; From: disk [✓ worksp.]

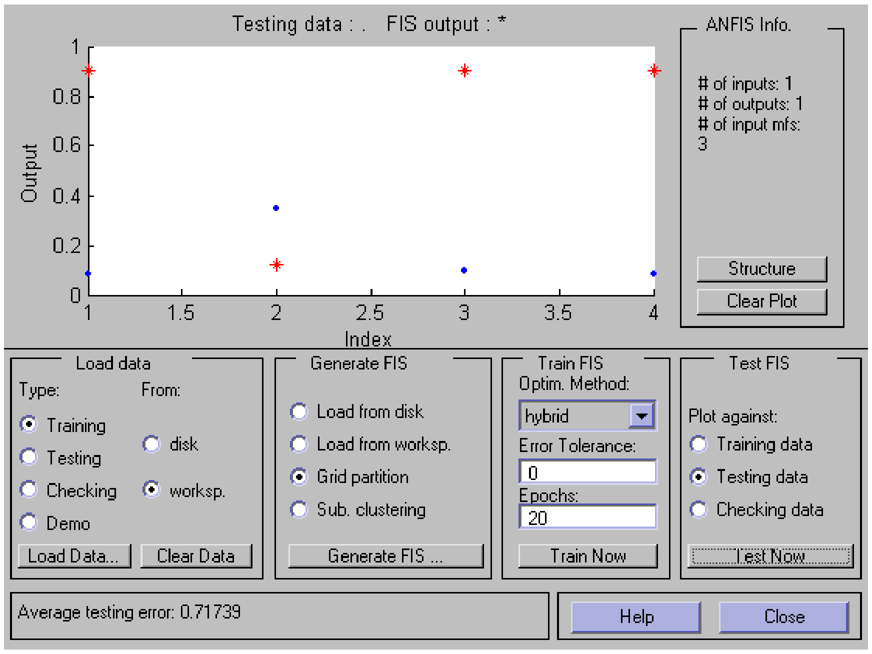
Generate FIS: Load from disk; Load from worksp.; [✓ Grid partition]; Sub. clustering

Train FIS — Optim. Method: hybrid; Error Tolerance: 0; Epochs: 30

Test FIS — Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.055913

Figure 8.9: Single Input OxidizerFl ANFIS Testing Results (gaussmf)



ANFIS Info.: # of inputs: 1; # of outputs: 1; # of input mfs: 3

Load data — Type: [✓ Training] Testing Checking Demo; From: disk [✓ worksp.]

Generate FIS: Load from disk; Load from worksp.; [✓ Grid partition]; Sub. clustering

Train FIS — Optim. Method: hybrid; Error Tolerance: 0; Epochs: 20

Test FIS — Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.71739

Figure 8.10: Single Input Pressurant ANFIS Testing Results (gaussmf)

As discussed in section 6, the neural network utilized was a backpropagation-trained network with a variable learning rate and momentum. The network has two layers, with the first layer size equal to the number of inputs to the network and the second layer size equal to one. Various initial learning rates were tried with the best results produced at 0.01. Varying numbers of inputs, different values for momentum, and different membership functions were tried. The best results were obtained with the set of six ANFIS outputs from the variables TestStand, Thrust, FuelFlow, TestDurMax, PressuraPr, and OxidizerFl, with a momentum value of 0.3, and logsig as the transfer function for both layers. Table 8.4 contains the predicted cost of each article out of the neural network for the best trial of Method 2A. This table also contains the percentage errors of each article, average percentage errors, and sum of differences value. Figure 8.11 shows graphically the predicted and actual cost values.

Table 8.4: Method 2A: Training and Testing Final Results from the Neural Network

Training Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	966,330	550,000	75.70
2	1,442,300	1,592,000	-9.40
3	4,471,900	4,433,000	0.88
4	2,173,100	2,203,000	-1.36
5	1,406,900	1,772,000	-20.60
6	12,045,000	12,051,000	-0.05
7	1,075,400	702,000	53.19
			Average Percent Error
			23.03
			Sum of Diff./Total Number of Sets
			197,047

Testing Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	1,377,400	1,590,000	-13.37
2	4,953,500	4,935,000	0.37
3	3,058,000	1,713,000	78.52
4	1,870,900	1,540,966	21.41
			Average Percent Error
			28.42
			Sum of Diff./Total Number of Sets
			476,509

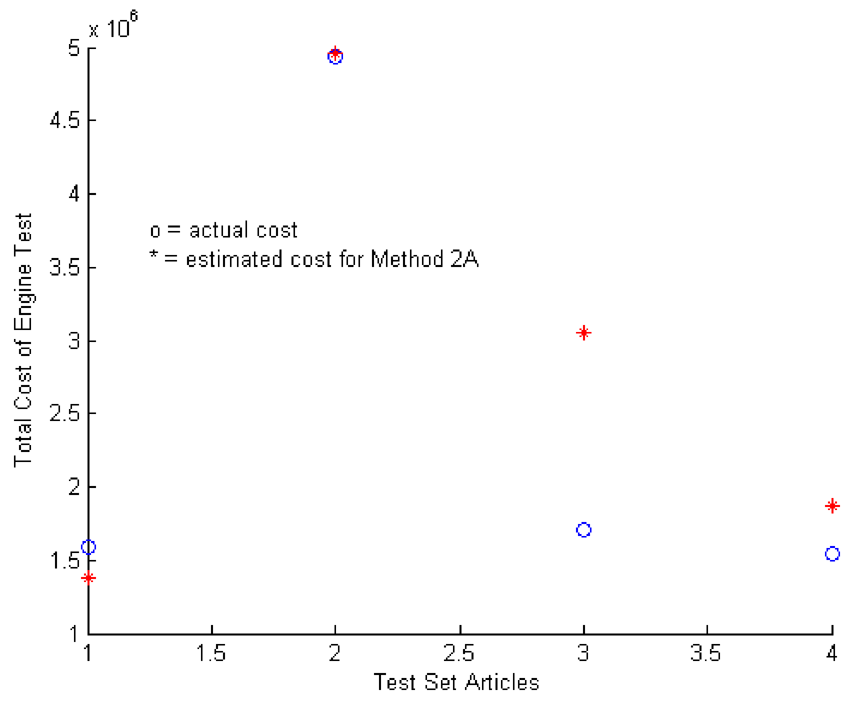


Figure 8.11: Method 2A Final Testing Results

While using method 2A to predict the engine test cost worked well for articles 1 and 2, articles 3 and 4 were not predicted well at all. Also, the training results had high percent errors. This problem that occurred repeatedly in many of the methods used for this cost prediction. The training errors should be negligible, but the errors are usually much larger due to the small size of the data set. Also, each variable singly does not strongly relate with the cost. This proved especially true with variables that have missing values (e.g. FuelFlow, Pressurant, and PressuraPr).

8.2.2. Method 2B

Method 2B used multiple inputs to ANFIS whose outputs were combined once again into a feed forward backpropagation trained neural network which produced the final predicted cost. Two different sets of ANFIS were developed. The first set of ANFIS utilized two variables as inputs with filled in values for missing quantities. The second set of ANFIS were developed using three variables as inputs, also with filled in values for missing quantities, as discussed in section 6. The ANFIS were developed similarly to the ones developed in Method 2A, varying membership function types and numbers. Again, Gaussian membership functions worked best and the number of membership functions was always maintained between two and four. Table 8.5 contains the training and testing results for the ANFIS with two inputs. The training results are included because some had significant errors. Table 8.6 contains the testing set results for the ANFIS developed with three inputs. The training data is not included because the error rates were all below 1%. Again, these values remain normalized because they will become the inputs to the neural network in the second stage. The average percent error and sum of differences value are included as well.

Table 8.5: Method 2B: Double Input ANFIS Training and Testing Results

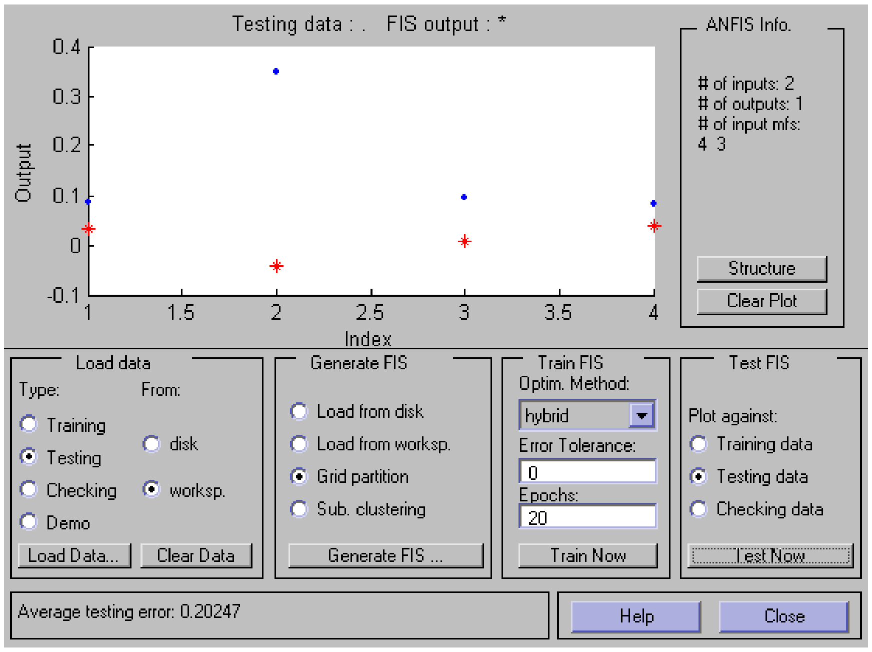
	Training Set				
	FuelFlow and Thrust	TestStand and Thrust	TestDurMax and PressuraPr	FuelFlow and OxidizerFL	Actual Cost
1	0.023922	0.0092527	0.0043098	0.019082	0.00431
2	0.087035	0.085964	0.085965	0.11078	0.085964
3	0.30067	0.22723	0.30859	0.27896	0.30859
4	0.13605	0.21521	0.13384	0.11682	0.13384
5	0.098533	0.10014	0.10007	0.20075	0.10007
6	0.90556	0.90556	0.90556	0.90555	0.90556
7	0.0027968	0.011209	0.016221	0.00020409	0.016221
	Average Percent Errors				
	77.827491	33.25776525	0.000693366	84.7544644	
	Sum of Differences/Total # of Sets				
	0.0065392	0.024679243	1.71429E-07	0.02899213	

	Testing Set				
	FuelFlow and Thrust	TestStand and Thrust	TestDurMax and PressuraPr	FuelFlow and OxidizerFL	Actual Cost
1	0.031928	0.018627	0.03019	0.21073	0.085808
2	-0.041581	0.50706	0.45494	-0.93182	0.34793
3	0.0090189	0.19006	0.10114	-0.064263	0.095446
4	0.038491	0.018627	0.060535	0.084687	0.081965
	Average Percent Errors				
	79.583	75.10825	31.920825	171.0126	
	Sum of Differences/Total # of Sets				
	0.143323	0.09606575	0.047438	0.39177575	

Table 8.6: Method 2B: Triple Input ANFIS Training and Testing Results

	Testing Set			
	TestStand, Thrust, and DuratDd	TestDurMax, PressuraPr, and Pressurant	FuelFlow, OxidyzerFl, and Fuel	Actual Cost
1	0.084547	0.057626	0.10373	0.085808
2	0.31657	0.45494	0.32568	0.34793
3	0.094349	0.060271	0.11495	0.095446
4	0.080778	0.10367	0.10772	0.081965
	Average Percent Errors (%)			
	3.270575	31.7335	19.7851	
	Sum of Differences/Total # of Sets			
	0.00872625	0.048018	0.02135775	

The variables paired in the double ANFIS were chosen by examining the results of the single input ANFIS and choosing variables that complemented each other. For example, if one variable always over estimated the cost then another variable that always underestimated the cost would be paired with it. Several pairings of variables were tried and these four ANFIS produced the best results. The variable Thrust was paired twice in the double input ANFIS because it was one of the most predictive variables, as is seen from the ANFIS results in method 2A. Also, double input ANFIS were developed with and without the missing values filled in. The filled in variables always produced the best ANFIS. The ANFIS developed with three inputs achieved very good results. Again, different pairings of three were tried, however these produced the best results. Also, a new variable, Fuel, was included because in the exhaustive and sequential searches, it was found to be a predictive variable for three input ANFIS. The ANFIS that used TestStand, Thrust, and DuratDd attained such good results that it could stand alone as a prediction without the neural network stage (Figure 8.16). However, the neural network takes more inputs into account, so it would operate more accurately with future use. The FuelFlow, OxidizerFl, and Fuel ANFIS also attained very good results, even though the variables individually were not the most predictive (Figure 8.18). Figures 8.12, 8.13, 8.14, and 8.15 are the testing results of the two input ANFIS. Figures 8.17, 8.18, and 8.19 show the testing results of the three input ANFIS.



ANFIS Info.
# of inputs: 2
# of outputs: 1
# of input mfs:
4 3

Load data
Type: Training [✓ Testing] Checking Demo
From: disk [✓ worksp.]

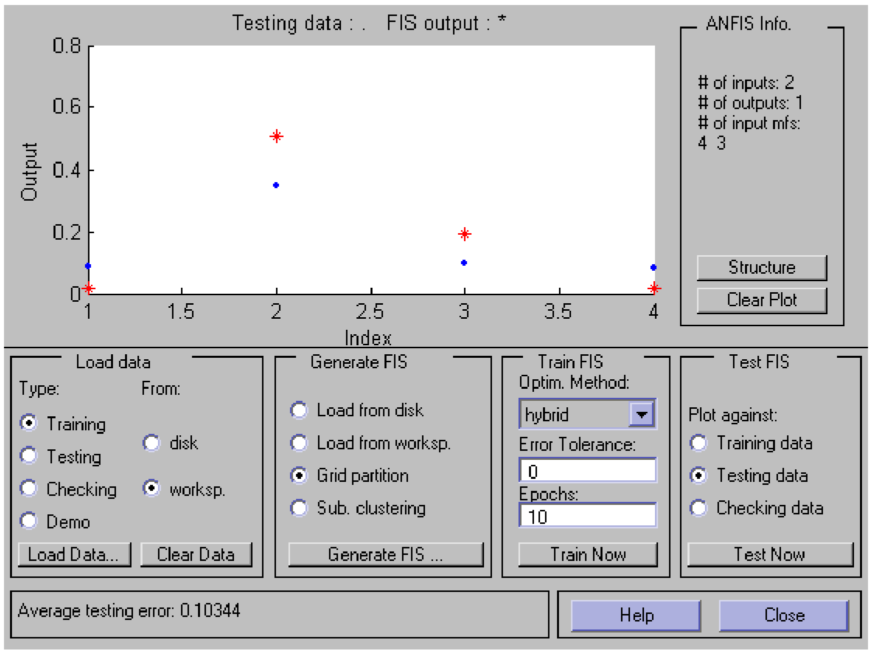
Generate FIS
Load from disk
Load from worksp.
[✓ Grid partition]
Sub. clustering

Train FIS
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 20

Test FIS
Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.20247

Figure 8.12: Method 2B Double Input FuelFlow and Thrust ANFIS Testing Results (gauss2mf)

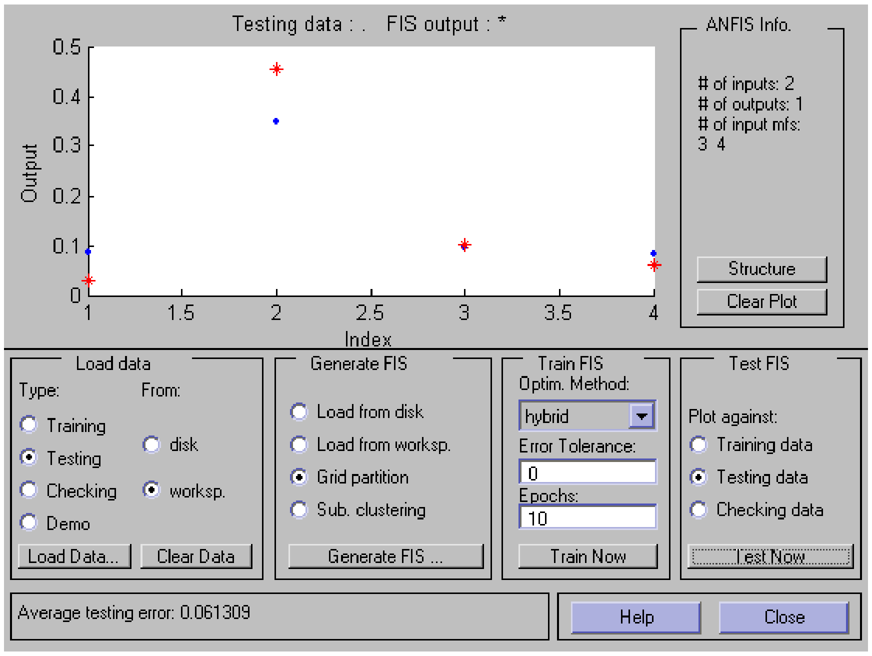


# of inputs: 2
# of outputs: 1
# of input mfs:
4 3

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]
Load from disk, Load from worksp., [✓ Grid partition], Sub. clustering
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 10
Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.10344

Figure 8.13: Method 2B Double Input TestStand and Thrust ANFIS Testing Results (gauss2mf)

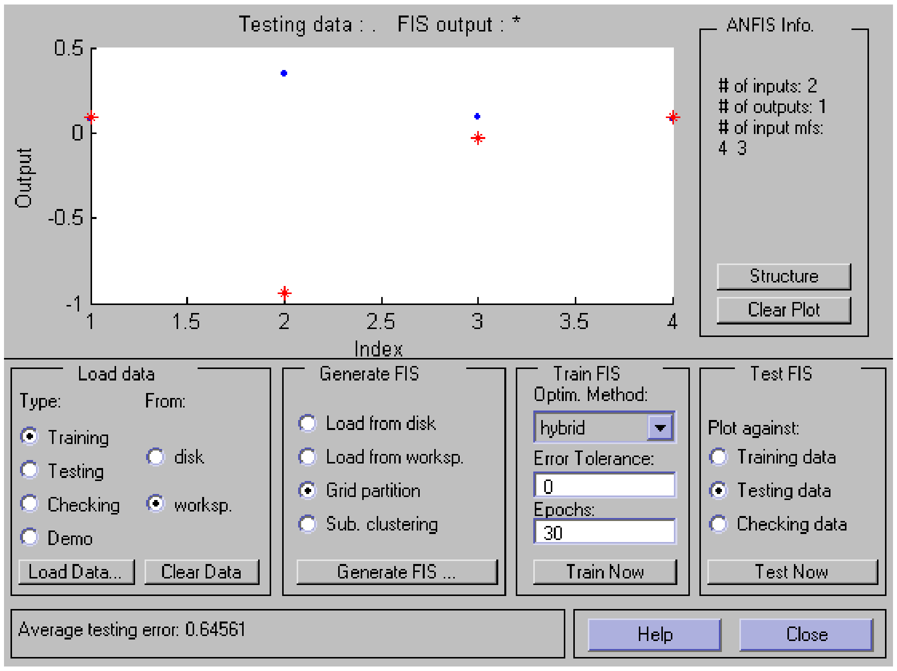


# of inputs: 2
# of outputs: 1
# of input mfs:
3 4

Type: Training [✓ Testing] Checking Demo
From: disk [✓ worksp.]
Load from disk, Load from worksp., [✓ Grid partition], Sub. clustering
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 10
Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.061309

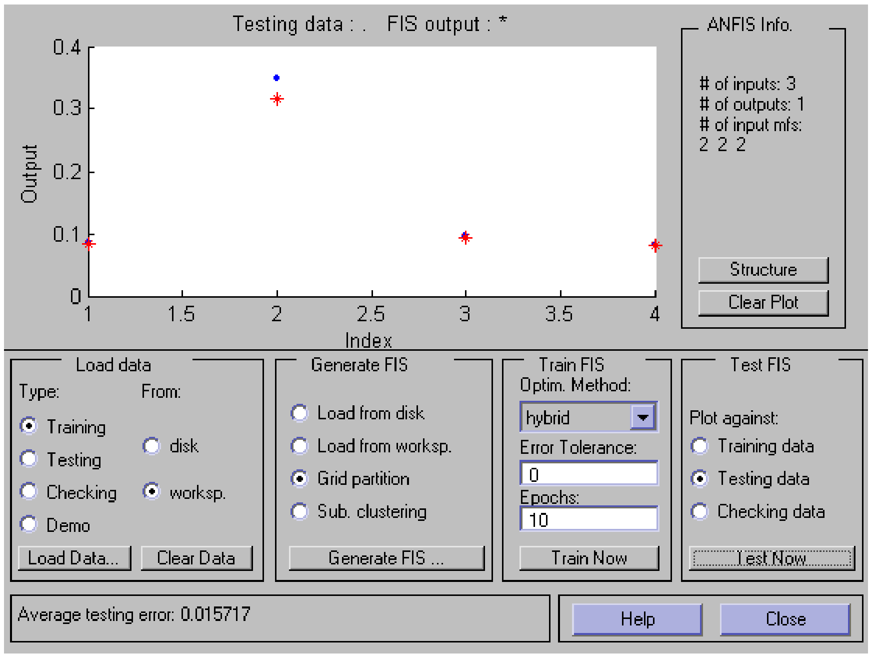
Figure 8.14: Method 2B Double Input TestDurMax and PressuraPr ANFIS Testing Results (gaussmf)



# of inputs: 2
# of outputs: 1
# of input mfs:
4 3

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]
Load from disk, Load from worksp., [✓ Grid partition], Sub. clustering
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 30
Plot against: Training data [✓ Testing data] Checking data
Average testing error: 0.64561

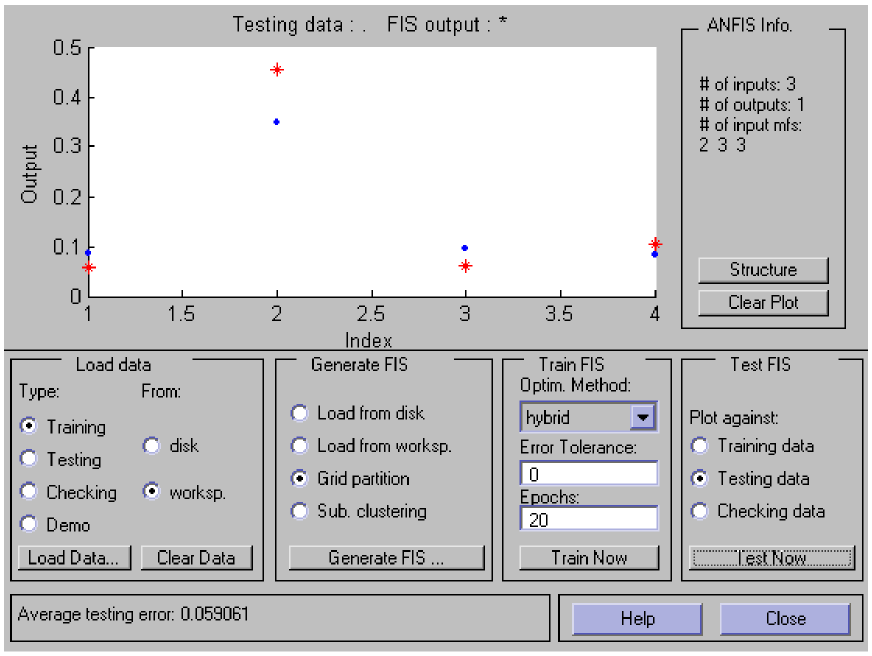
Figure 8.15: Method 2B Double Input FuelFlow and OxidizerFl ANFIS Testing Results (gauss2mf)



# of inputs: 3
# of outputs: 1
# of input mfs:
2 2 2

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]
Load from disk, Load from worksp., [✓ Grid partition], Sub. clustering
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 10
Plot against: Training data [✓ Testing data] Checking data
Average testing error: 0.015717

Figure 8.16: Method 2B Triple Input TestStand, Thrust, and DuratDd ANFIS Testing Results (gbellmf)

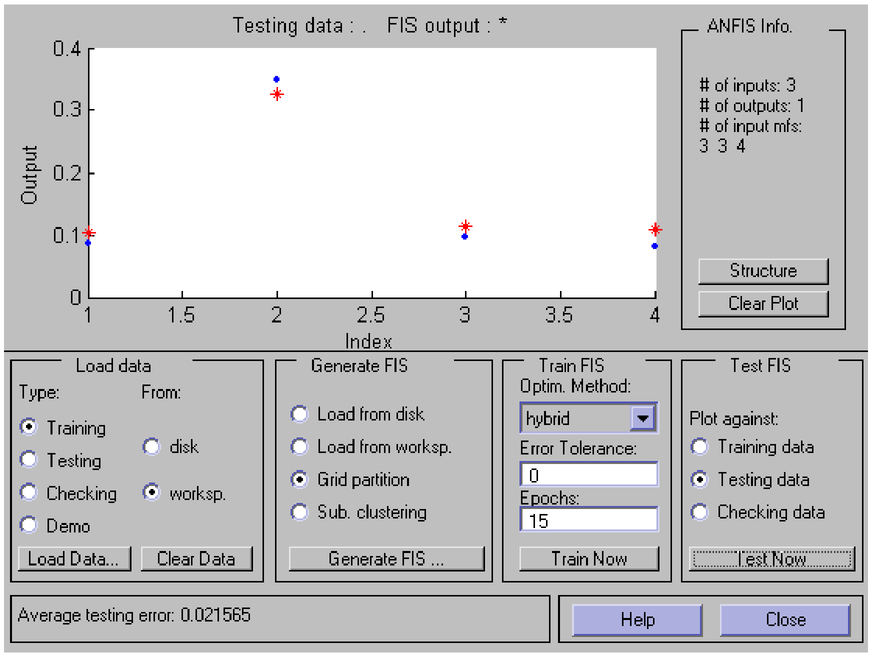


# of inputs: 3
# of outputs: 1
# of input mfs:
2 3 3

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]
Generate FIS: Load from disk, Load from worksp., [✓ Grid partition], Sub. clustering
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 20
Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.059061

Figure 8.17: Method 2B Triple Input TestDurMax, PressuraPr, and Pressurant ANFIS Testing Results (gaussmf)



# of inputs: 3
# of outputs: 1
# of input mfs:
3 3 4

Type: [✓ Training] Testing Checking Demo
From: disk [✓ worksp.]
Generate FIS: Load from disk, Load from worksp., [✓ Grid partition], Sub. clustering
Optim. Method: hybrid
Error Tolerance: 0
Epochs: 15
Plot against: Training data [✓ Testing data] Checking data

Average testing error: 0.021565

Figure 8.18: Method 2B Triple Input FuelFlow, OxidizerFl, and Fuel ANFIS Testing Results (gaussmf)

The neural network developed for both the two and three input ANFIS was very similar to Method 2A. Again, the number of first layer neurons was set equal to the number of inputs and a single neuron was used for the second layer. The initial learning rate used for both neural networks developed is 0.01. The momentum and types of transfer functions were varied as before. Varying inputs were tried for the neural network using the two input ANFIS. The best results were obtained by using all of the four two input ANFIS developed. For the results presented here a momentum value of 0.3 and transfer functions of type *logsig* were used. The network developed using the three input ANFIS also performed very well. The results presented here did not have the lowest average percent error for the testing set that was found, however, some of the training average percent errors were higher. This network was chosen to represent the best results because it was the only network developed that had percent errors of close to 10% for both the training and testing set. Tables 8.7 and 8.8 contain the results of both the neural networks developed for the two input ANFIS and the three input ANFIS, respectively. Figure 8.19 shows the predicted costs of both the networks compared to the actual cost.

Table 8.7: Method 2B: Double Input ANFIS Training and Testing Final Results from the Neural Network

Training Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	823,990	550,000	49.82
2	1,320,800	1,592,000	-17.04
3	4,446,500	4,433,000	0.30
4	2,235,700	2,203,000	1.48
5	1,785,000	1,772,000	0.73
6	12,042,000	12,051,000	-0.07
7	814,860	702,000	16.08
			Average Percent Error
			12.22
			Sum of Diff./Total Number of Sets
			103,750

Testing Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	1,314,100	1,590,000	-17.35
2	5,231,100	4,935,000	6.00
3	1,268,500	1,713,000	-25.95
4	1,074,200	1,540,966	-30.29
			Average Percent Error
			19.90
			Sum of Diff./Total Number of Sets
			370,817

Table 8.8: Method 2B: Triple Input ANFIS Training and Testing Final Results from the Neural Network

Training Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	747,600	550,000	35.93
2	1,436,600	1,592,000	-9.76
3	4,613,100	4,433,000	4.06
4	2,058,700	2,203,000	-6.55
5	1,606,900	1,772,000	-9.32
6	11,976,000	12,051,000	-0.62
7	816,590	702,000	16.32
			Average Percent Error
			11.79
			Sum of Diff./Total Number of Sets
			147,441

Testing Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	1,441,500	1,590,000	-9.34
2	5,306,200	4,935,000	7.52
3	1,546,200	1,713,000	-9.74
4	1,562,100	1,540,966	1.37
			Average Percent Error
			6.99
			Sum of Diff./Total Number of Sets
			176,909

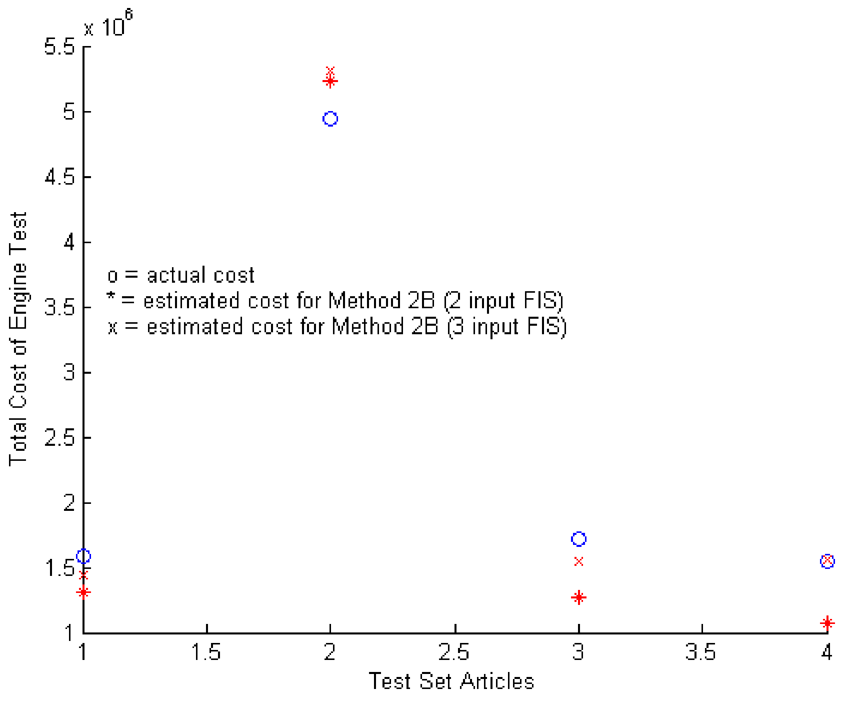


Figure 8.19: Method 2B Final Testing Results

The results attained from using method 2B were very good. Even the double input ANFIS that initially presented very large individual percentage errors produced good cost predictions after the neural network stage. The best results were derived from the systems created from the three input ANFIS which produced an average percent error of well under 10%.

8.3. Results for Method 3: RBFN

Method 3 uses an RBFN to predict the engine test cost. As previously discussed, two RBFNs were implemented: one with a training set of the entire data set (eleven) and the second with a training set of seven. The RBFN with the training set of eleven is used simply as a comparison to the RBFN with a training set of seven due to the difficulty of training an RBFN, which uses a k-means clustering algorithm, on such a small data set of size seven. This is evident from the results of method 1 which also uses k-means in the LLE algorithm. The first RBFN's (training set of eleven) results are misleading because unlike in every other method implemented, the RBFN has already seen the testing set as part of the training set. The comparison of the two RBFNs, however, demonstrates that this would be a more effective method if more data sets were available.

The RBFNs were both developed by varying the number of clusters k and the number of nearest neighbors p. The number of inputs was also varied, however, both RBFNs always performed better with the full data set of eighteen variables. The best results were usually attained when k and p were allowed to reach their highest values, 6 and 4 respectively. This is true of the RBFN developed with the training set of eleven values. The exception to this would be the RBFN developed with the training set of seven values, which attained the best results with three clusters and two neighbors ($k = 3$, $p = 2$). Tables 8.9 and 8.10 contain the predicted values and percentage errors of the RBFN with a training set of 11 and a training set of 7, respectively. Figure 8.20 shows a graphical representation of the predicted costs of both RBFNs compared to the actual costs. Appendix 12.7 contains the actual center values, variances, and average costs of clusters for the results presented here.

Table 8.9: Method 3: RBFN (Training Set of 11) Training and Testing Final Results

Training Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	2,205,800	550,000	301.05
2	2,143,800	1,592,000	34.66
3	1,767,800	4,433,000	-60.12
4	1,813,100	2,203,000	-17.70
5	1,695,900	1,772,000	-4.29
6	2,139,400	12,051,000	-82.25
7	1,697,100	702,000	141.75
8	1,542,400	1,590,000	-2.99
9	2,668,900	4,935,000	-45.92
10	1,726,800	1,713,000	0.81
11	1,674,000	1,540,966	8.63
			Average Percent Error
			91.69
			Sum of Diff./Total Number of Sets
			2,320,786

Testing Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	1,542,400	1,590,000	-2.99
2	2,668,900	4,935,000	-45.92
3	1,726,800	1,713,000	0.81
4	1,674,000	1,540,966	8.63
			Average Percent Error
			14.59
			Sum of Diff./Total Number of Sets
			615,134

Table 8.10: Method 3: RBFN (Training Set of 7) Training and Testing Final Results

Training Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	2,519,200	550,000	358.04
2	2,339,300	1,592,000	46.94
3	1,567,200	4,433,000	-64.65
4	1,625,000	2,203,000	-26.24
5	1,788,300	1,772,000	0.92
6	2,389,400	12,051,000	-80.17
7	1,578,300	702,000	124.83
			Average Percent Error
			100.25
			Sum of Diff./Total Number of Sets
			2,387,786

Testing Set			
	Predicted Cost	Actual Cost	Percent Errors (%)
1	1,802,100	1,590,000	13.34
2	1,850,200	4,935,000	-62.51
3	1,825,400	1,713,000	6.56
4	1,813,900	1,540,966	17.71
			Average Percent Error
			25.03
			Sum of Diff./Total Number of Sets
			920,559

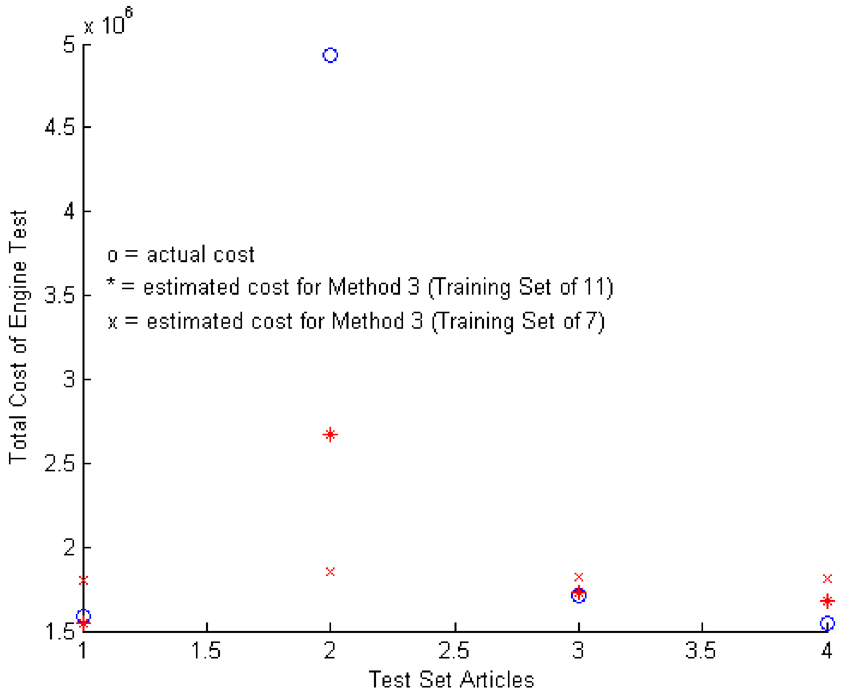


Figure 8.20: Method 3 Final Testing Results

Both RBFNs predicted the cost of articles 1, 3, and 4 fairly accurately, but had trouble predicting article 2. This is understandable since article 2 is so much larger than the others. The RBFN with a training set of 11 had better results, as expected, since it has already seen the testing set. It also had lower errors for the training set as well. This shows that if the data set were larger, the network could train better, which would produce better results. Our data reflects only a very small change because even with increasing the training set to eleven, it is still quite a small set for the eighteen dimensions it has.

8.4. Comparison of Results for All Methods

Table 8.11 compares the percentage errors obtained for the training and testing results for all methods. Figure 8.22 contains a graphical representation of the comparison of all predicted costs to the actual costs for the testing sets. The best results were obtained from Method 2B, especially the results obtained for the network developed using the three input ANFIS, which achieved a testing average percent error of 7% with no quantity individually estimated with an error above 10%. Other methods were not effective, namely method 1, which did a poor job of predicting almost all the articles. Method 2A had trouble predicting article 3. This was unexpected because it would be logical to assume it would have more trouble with article 2 due to its larger cost. It is difficult for the neural network to find an accurate prediction when the outputs of the ANFIS are so varied as is the case in Method 2A. Method 3 also had trouble predicting the costs of the engine tests. This is due to the size of the training set, since it, like the LLE method, must rely on clustering methods that usually require large data sets.

Table 8.11: Training and Testing Final Results for All Methods

Training Set Percent Errors (%)						
	Method 1	Method 2A	Method 2B (2 Input ANFIS)	Method 2B (3 Input ANFIS)	Method 3 (Training Set of 11)*	Method 3 (Training Set of 7)
1	1.05E-04	75.70	49.82	**35.93**	301.05	358.04
2	-9.02E-05	-9.40	-17.04	**-9.76**	34.66	46.94
3	-8.69E-05	0.88	0.30	**4.06**	-60.12	-64.65
4	-2.17E-05	-1.36	1.48	**-6.55**	-17.70	-26.24
5	-4.99E-05	-20.60	0.73	**-9.32**	-4.29	0.92
6	-2.92E-05	-0.05	-0.07	**-0.62**	-82.25	-80.17
7	-2.06E-05	53.19	16.08	**16.32**	141.75	124.83
	Average Percent Error (%)					
	5.77E-05	23.03	12.22	**11.79**	91.69	100.25

Testing Set Percent Errors (%)						
	Method 1	Method 2A	Method 2B (Double Input ANFIS)	Method 2B (Triple Input ANFIS)	Method 3 (Training Set of 11)	Method 3 (Training Set of 7)
1	-96.435	-13.37	-17.35	**-9.34**	-2.99	13.34
2	-89.789	0.37	6.00	**7.52**	-45.92	-62.51
3	-63.963	78.52	-25.95	**-9.74**	0.81	6.56
4	-16.258	21.41	-30.29	**1.37**	8.63	17.71
	Average Percent Error (%)					
	66.61125	28.42	19.90	**6.99**	14.59	25.03

* The last four training sets are not placed here since they are identical to the four testing sets.

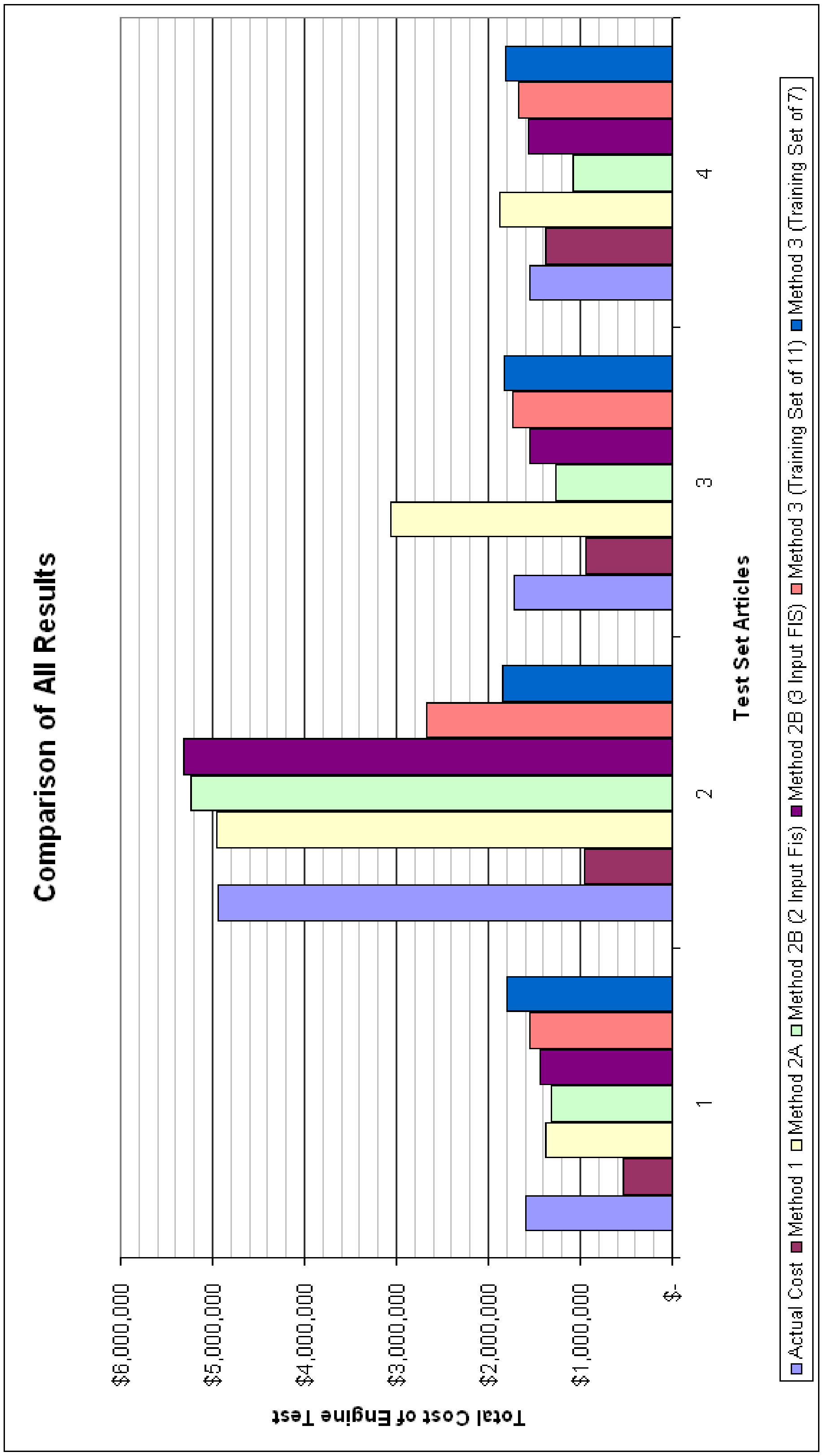


Figure 8.21: Final Testing Results for All Methods

9. CONCLUSIONS

Several different fuzzy and neural methods were implemented to solve the problem of predicting the cost of performing engine tests, with a small incomplete data set. Methods 1 (LLE) and 3 (RBFN) were found to be largely ineffective in this problem due to the small size of the data set. Method 3, however, shows promise for use if the data set is ever enlarged, as indicated in the comparison of the two RBFNs that were developed. Method 2 proved to perform the best for small data set sizes, particularly method 2B with multiple variables as inputs into the ANFIS stage. The problem of incomplete data sets was mitigated by filling in values with either their mean, median, or mode values from the entire variable set. This was proven with the method 2B ANFIS where both filled-in and non-filled-in ANFIS were developed. The best engine test cost predicting system was method 2B using the three input ANFIS where an error under 10% was achieved for every test article. In the future, if any further data could be obtained, these same methods could be implemented to attain better results.

9.1. Suggestions for Future Work

A graphical user interface could be utilized when implementing the three input ANFIS system developed in method 2B to make the task of actually predicting new data easier. Currently, the data set must be manually entered into the Matlab script. Also, some way of condensing the number of files would be helpful. Presently, the three ANFIS and the neural network developed must be placed in the Matlab work file for the estimation to run.

Another approach to create this cost estimating model could be to use a bimodal approach where certain networks are used for very expensive tests and other networks are used for the less expensive tests. For example the RBFN method generally worked very well with everything but the most expensive test article. This method could be used for less expensive tests. However, only method 2 predicted the most expensive test accurately. So these two systems could be used based upon whether a test was expected to be expensive or less expensive.

Another addition to the method 2B three input ANFIS system could be to take into account the fuzziness inherent in the variables themselves. This system is used to create a prediction of cost, which means that the variables themselves still have some degree of uncertainty. This uncertainty could be taken into account to give a probability of accuracy for the cost prediction itself. For example if the Thrust was known with a 90% surety and TestDurMax was known with a 50% surety, then the cost prediction would have some degree of accuracy associated with itself as well. Although difficult to implement, this would be a useful consideration when planning and proposing future projects.

Finally, a study comparing these methods with the current methods for use at NASA and other methods used to estimate cost in this field would also be a helpful addition to this work, to determine which methods perform the best.

10. REFERENCES

[1] E. J. Kaminsky, F. Douglas, “A fuzzy-neural highly accurate cost estimating model (HACEM),” in *CIEF’2003 Conf. (3rd. int. workshop on Computational Intelligence in Economics and Finance),* Cary, NC, Sept. 26-30, 2003; pp. 1035-1039.

[2] E. J. Kaminsky, “Highly accurate cost estimating model (HACEM),” Project Final Report LA BoR Contract No. NASA(2001)-Stennis-15, NASA-SSC Space Operations, May 2002.

[3] J. Tenenbaum, V. Silva, J. Langford, “A global geometric framework for nonlinear dimensionality reduction,” *Science*, December 2000; 290:2319-2322.

[4] D. Gering, “Linear and nonlinear data dimensionality reduction,” University of Michigan [online] April 2002, http://www.ai.mit.edu/people/gering/areaexam/areaexam.pdf, (Accessed: November 2003).

[5] T. Friedrich, “Nonlinear dimensionality reduction with locally linear embedding and isomap,” Ph.D. Dissertation, University of Sheffield, 2003.

[6] P. Demartines and J. Herault, “Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets,” *IEEE Transactions on Neural Networks,* January 1997; 8.1:148-154.

[7] M. Brand, “Continuous nonlinear dimensionality reduction by kernel eigenmaps,” Mitsubishi Electric Research Laboratory [online] November 2002, http://www.merl.com/reports/docs/TR2003-21.pdf, (Accessed: November 2003).

[8] H. Ishibuchi, A. Miyazaki, and H. Tanaka, “*Neural-network-based diagnosis systems for incomplete data with missing inputs*,” *Proceedings of IEEE International Conference on Neural Networks*, Orlando, USA, June1994;3457-3460.

[9] E. Granger, M. Rubin, S. Grossberg, P. Lavoie, “Classification of incomplete data using the fuzzy ARTMAP neural network,” *Proceedings of IEEE International Joint Conference on Neural Networks*, Como, Italy, July 2000; 6:35-40.

[10] H. Chongfu, “Deriving samples from incomplete data,” *IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, May 1998; 1:645-650.

[11] Y. Zhou, "Neural network learning from incomplete data," Washington University [online] August 2000, http://www.cs.wustl.edu/~zy/learn.pdf, (Accessed: November 2003).

[12] J.S. Jang, C.T. Sun, "ANFIS: Adaptive-network-based fuzzy interference system," *IEEE Transactions on Systems, Man, and Cybernetics,* June 1993; 23:665-685.

[13] J.S. Jang, C.T. Sun, "Neuro-fuzzy modeling and control," *Proceedings of the IEEE,* March 1995; 83.3:378-403.

[14] J. Mendel, "Uncertain rule-based fuzzy logic systems: Introduction and new directions," Prentice Hall, 2001.

[15] T. Takagi, M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man, and Cybernetics,* Marseille, France, February 1985; 1:116-132.

[16] T. Takagi, M. Sugeno, "Deriviation of fuzzy controls rules from human operator's control actions," *Iternational Federation of Automatic Control symposium,* July 1983; 55-60.

[17] S. Roweis, L. Saul, " Nonlinear dimensionality reduction by locally linear embedding," *Science* December 2000; 290:2323-2326.

[18] L. K. Saul and S. T. Roweis, "An introduction to locally linear embedding," Toronto University [online] , http://www.cs.toronto.edu/~roweis/lle/papers/lleintro.pdf, (Accessed: December 2003).

11. APPENDICES

11.1. Engine Test Data Set

HACEM		NASA Test	Duration	Number	Max test	Fuel	Fuel pressurant	Oxydizer		Fuel flow	Fuel pressurant flow	Oxidizer	Pressurant
Engine Data	Article Number	Article Number	in days	of Tests	Duration	Type	Code	Type	Thrust	Rate	Flow rate	Flow rate	Presure
Training Set	1	1	9	2	70	**5**	4	1	4250	4.75	10.44	25.00	600
	2	2	252	48	150	5	4	1	80000	64.00	0.55	139.00	800
	3	5	180	**70**	**200**	1	2	5	2500	30.00	25.00	25.00	150
	4	6	240	10	180	5	1	7	**1740**	174.00	172.00	**12.00**	1512
	5	35	360	14	34	23	7	1	60000	0	12	140	2000
	6	15	75	10	180	5	0	1	1800000	2,035.00	0.00	4,620.00	0
	7	28	45	25	200	8	4	6	5450	3.6	**20**	23.4	400
Testing Set	1	29	540	153	100	24	**0**	1	10000	0	0	150	3000
	2	30	120	42	8	2	9	1	645000	267	200	1667	15000
	3	34	120	21	45	23	**0**	6	150	1	0	1.78	1000
	4	14	105	17	60	22	0	1	10000	9.52	0.00	23.81	0
		Measure	Cooling?	Number of	Safety?	Handling?	Test	Facility	Total	Equipment +	Design	Fabrication	Modification
		Thrust?	Cooling	Gimbal axes	Safety	Handling	Stand	Modification Level	Cost	Material Cost	Cost	Cost	Cost
		1	1	2	0	0	3	3	550000	74000	87000	204000	185000
		0	1	1	0	0	5	2	1592000	50000	117000	725000	10000
		0	0	0	0	0	2	1	4433000	NaN	NaN	NaN	1619000
		0	1	0	0	0	2	1	2203000	NaN	1746000	NaN	NaN
		1	0	0	0	0	3	1	1772000	NAN	77000	1109000	85000
		1	1	0	0	0	4	1	12051000	NaN	7073000	NaN	NaN
		1	0	0	1	1	3	1	702000	NAN	59000	517000	58000
		1	0	0	0	0	3	1	1590000	NAN	65000	626000	65000
		0	0	0	0	1	1	2	4935000	150000	210000	1390000	390000
		1	1	0	1	1	2	2	1713000	NAN	NAN	NAN	NAN
		1	0	0	0	0	3	1	1540966	108500	NaN	NaN	NaN

11.2. Variable Codes

Fuel Codes		
Fuel Long Name	**Fuel short name**	**Code**
Liquid Oxygen	LOX	1
Liquid Hydrogen	LH2	2
Liquid Nitrogen	LN2	3
Gaseous Helium	GHe	4
	RP1	5
Hydrogen Peroxide	H202	6
Gaseous Nitrogen	GN2	7
	JP8	8
Gaseous Hydrogen	GH2	9
	GAr	10
	Missile grade air	11
	Natural gas	12
	Diesel	13
	CH4	14
	JAk	15
Gasous oxygen	GOX	16
	LN	17
Water	H20	18
Carbon Dioxide	CO2	19
High Pressure Air	HPA	20
Carbon Monoxide	CO	21
Liquified Nitrogen Gas	LNG	22
Proprietary	PRO	23
Standard Hybrid Rocket Fuel (Hydroxyl-Terminated Polybutadiene)	HTPB	24

Test Stand Codes	
Test Stand	**Code**
E1	1
E2	2
E3	3
B1	4
B2	5
H1	6

Facility Modification Codes	
Modification	**Code**
Low	1
Medium	2
High	3

11.3. Method 1: LLE/ANFIS Development

```
% lleTesting.m
close all;clear all;
% Method 1: LLE/ANFIS Development
% the variables used are:

input_name=['DuratDd   ';'NoTest    ';'TestDurMax';'Fuel      ';'Pressurant';...
        'Oxydizer  ';'Thrust    ';'FuelFlow  ';'FuelPresFl';'OxydizerFl';...
        'PressuraPr';'ThrustMeas';'Cooling?  ';'GimbalAxes';'Safety?   ';...
        'Handling? ';'TestStand ';'FacilitMod';'TotalCost '];

%   1   2 3   4   5   6   7     8     9   10    11    12  13  14  15  16  17  18  19
ttb=[9      2        70       5        4        1        4250      4.75  10.44        25        600       1        1
            2        0        0        3        3        550000;
   252      48       150      5        4        1        80000     64         0.55     139       800       0        1
            1        0        0        5        2        1592000;
   180      70       200      1        2        5        2500      30         25       25        150       0        0
            0        0        0        2        1        4433000;
   240      10       180      5        1        7        1740      174        172      12        1512      0        1
            0        0        0        2        1        2203000;
   360      14       34       23       7        1        60000     0          12       140       2000      1        0
            0        0        0        3        1        1772000;
   75       10       180      5        0        1        1800000   2035  0     4620     0          1        1        0
            0        0        4        1        12051000;
   45       25       200      8        4        6        5450      3.6        20       23.4 400   1        0        0
            1        1        3        1        702000;
   540      153      100      24       0        1        10000     0          0          150     3000      1        0
            0        0        0        3        1        1590000;
   120      42       8        2        9        1        645000    267        200      1667 15000          0        0
            0        0        1        1        2        4935000;
   120      21       45       23       0        6        150         1        0          1.78     1000  1  1        0
            1        1        2        2        1713000;
   105      17       60       22       0        1        10000     9.52  0      23.81  0    1        0        0        0
            0        3        1        1540966];

% Normalization
x_min = 0.9*min(ttb(:,19));
x_max = 1.1*max(ttb(:,19));

data(:,1) = (ttb(:,1)-.9*min(ttb(:,1)))/(1.1*max(ttb(:,1))-.9*min(ttb(:,1)));
data(:,2) = (ttb(:,3)-.9*min(ttb(:,3)))/(1.1*max(ttb(:,3))-.9*min(ttb(:,3)));
data(:,3) = (ttb(:,5)-1)/(9-1);
data(:,4) = (ttb(:,7)-.9*min(ttb(:,7)))/(1.1*max(ttb(:,7))-.9*min(ttb(:,7)));
data(:,5) = (ttb(:,8)-.9*min(ttb(:,8)))/(1.1*max(ttb(:,8))-.9*min(ttb(:,8)));
data(:,6) = (ttb(:,10)-.9*min(ttb(:,10)))/(1.1*max(ttb(:,10))-.9*min(ttb(:,10)));
data(:,7) = (ttb(:,11)-.9*min(ttb(:,11)))/(1.1*max(ttb(:,11))-.9*min(ttb(:,11)));
data(:,8) = (ttb(:,17)-1)/(6-1); % Special Normalization for TestStand (since there are 6)
data(:,9) = (ttb(:,19)-x_min)/(x_max-x_min);

train = data(1:7,:);                    % use 7 rows for training data
test = data(8:11,:);                    % use 4 rows for testing data

% Training and Testing
p = train(:,1:8)';
t = train(:,9)';                                        % use the last column (total cost)
p2 = test(:,1:8)';
t2 = test(:,9)';                        % use the last column (total cost)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LLE Transformation
% Training Data
K = 3;                  % LLE number of neighbors
d = 3;                  % Number of desired dimensions
p_new=lle(p,K,d);           % LLE transformed input set (p)
in_train = [p_new;t]';
% Testing Data
K2 = 3;                  % LLE number of neighbors
```

```
d2 = 3;                    % Number of desired dimensions
p_new2 = lle(p2,K2,d2);    % LLE transformed input set (p)
in_test = [p_new2;t2]';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicting Training Data
fismat = readfis('lletest_final04');
fuz_train(:,1) = (evalfis(p_new,fismat));
e_train(:,1) = ((t'-fuz_train)./t')*100;
er_train = mean(abs(e_train))

res_train = fuz_train.*(x_max-x_min)+x_min; % Multiplying by normilization factor
tar_train = t.*(x_max-x_min)+x_min;         % Multiplying by normilization factor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicting Testing Data
fismat = readfis('lletest_final04');
fuz_test(:,1) = (evalfis(p_new2,fismat));
e_test(:,1) = ((t2'-fuz_test)./t2')*100;
er_test = mean(abs(e_test))

res_test = fuz_test.*(x_max-x_min)+x_min;   % Multiplying by normilization factor
tar_test = t2.*(x_max-x_min)+x_min;         % Multiplying by normilization factor

hold on
plot(res_test,'r*')
plot(tar_test,'o')
hold off
xlabel('Test Set Articles')
ylabel('Total Cost of Engine Test')

text(1.25,3.75e6,'o = actual cost')
text(1.25,3.55e6,'* = estimated cost')
```

11.4. LLE Code

```
% LLE ALGORITHM (using K nearest neighbors)
%
% [Y] = lle(X,K,dmax)
%
% X = data as D x N matrix (D = dimensionality, N = #points)
% K = number of neighbors
% dmax = max embedding dimensionality
% Y = embedding as dmax x N matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Y] = lle(X,K,d)

[D,N] = size(X);
fprintf(1,'LLE running on %d points in %d dimensions\n',N,D);

% STEP1: COMPUTE PAIRWISE DISTANCES & FIND NEIGHBORS
fprintf(1,'-->Finding %d nearest neighbours.\n',K);

X2 = sum(X.^2,1);
distance = repmat(X2,N,1)+repmat(X2',1,N)-2*X'*X;

[sorted,index] = sort(distance);
neighborhood = index(2:(1+K),:);

% STEP2: SOLVE FOR RECONSTRUCTION WEIGHTS
fprintf(1,'-->Solving for reconstruction weights.\n');

if(K>D)
  fprintf(1,'   [note: K>D; regularization will be used]\n');
  tol=1e-3; % regularlizer in case constrained fits are ill conditioned
else
  tol=0;
end

W = zeros(K,N);
for ii=1:N
   z = X(:,neighborhood(:,ii))-repmat(X(:,ii),1,K); % shift ith pt to origin
   C = z'*z;                                    % local covariance
   C = C + eye(K,K)*tol*trace(C);                  % regularlization (K>D)
   W(:,ii) = C\ones(K,1);                        % solve Cw=1
   W(:,ii) = W(:,ii)/sum(W(:,ii));                 % enforce sum(w)=1
end;

% STEP 3: COMPUTE EMBEDDING FROM EIGENVECTS OF COST MATRIX M=(I-W)'(I-W)
fprintf(1,'-->Computing embedding.\n');

% M=eye(N,N); % use a sparse matrix with storage for 4KN nonzero elements
M = sparse(1:N,1:N,ones(1,N),N,N,4*K*N);
for ii=1:N
   w = W(:,ii);
   jj = neighborhood(:,ii);
   M(ii,jj) = M(ii,jj) - w';
   M(jj,ii) = M(jj,ii) - w;
   M(jj,jj) = M(jj,jj) + w*w';
end;

% CALCULATION OF EMBEDDING
options.disp = 0; options.isreal = 1; options.issym = 1;
[Y,eigenvals] = eigs(M,d+1,0,options);
Y = Y(:,2:d+1)'*sqrt(N); % bottom evect is [1,1,1,1...] with eval 0


fprintf(1,'Done.\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other possible regularizers for K>D
%   C = C + tol*diag(diag(C));                  % regularlization
%   C = C + eye(K,K)*tol*trace(C)*K;              % regularlization
```

11.5. Example of Method 2 Development

```
% method2Testing.m
% 3/8/04
% Method 2: ANFIS/Neural Development
% Properly normalized, 7 training, 4 testing, double fis 10
% filling in missing data in columns 5, 8, 11 (mean or median)
close all;clear all;
% ANFIS HACEM Developing & Processing for Engines
% the variables used are:

input_name=['DuratDd   ';'NoTest    ';'TestDurMax';'Fuel      ';'Pressurant';...
        'Oxydizer  ';'Thrust    ';'FuelFlow  ';'FuelPresFl';'OxidizerFl';...
        'PressuraPr';'ThrustMeas';'Cooling?  ';'GimbalAxes';'Safety?   ';...
        'Handling? ';'TestStand ';'FacilitMod';'TotalCost '];

%    1   2  3   4   5   6   7      8      9   10    11    12  13  14  15  16  17  18  19
ttb=[9     2      70      5      4      1      4250      4.75  10.44      25      600      1      1
       2      0       0      3      3      550000;
   252    48     150      5      4      1      80000     64        0.55     139     800      0      1
       1      0       0      5      2      1592000;
   180    70     200      1      2      5      2500      30        25       25      150      0      0
       0      0       0      2      1      4433000;
   240    10     180      5      1      7      1740      174       172      12      1512     0      1
       0      0       0      2      1      2203000;
   360    14      34     23      7      1      60000     0         12       140     2000     1      0
       0      0       0      3      1      1772000;
    75    10     180      5      0      1      1800000   2035  0      4620      0          1      1      0
       0      0       4      1      12051000;
    45    25     200      8      4      6      5450      3.6       20       23.4 400   1        0      0
       1      1       3      1      702000;
   540   153     100     24      0      1      10000     0         0          150     3000     1      0
       0      0       0      3      1      1590000;
   120    42       8      2      9      1      645000    267       200      1667 15000          0      0
       0      0       1      1      2      4935000;
   120    21      45     23      0      6      150         1       0          1.78      1000  1    1      0
       1      1       2      2      1713000;
   105    17      60     22      0      1      10000     9.52  0       23.81  0      1       0      0      0
       0      3       1      1540966];

% Filling in missing data
% Correcting for Pressurant
pres = median(ttb(:,5));
[a,b] = find(ttb(:,5)==0);
ttb(a,5) = pres;
% Correcting for FuelFlow
ff = mean(ttb(:,8));
[a,b] = find(ttb(:,8)==0);
ttb(a,8) = ff;
% Correcting for PressuraPr
ppr = median(ttb(:,11));
[a,b] = find(ttb(:,11)==0);
ttb(a,11) = ppr;

% Data Variable Selection
data_un(:,1) = ttb(:,17);   % Select TestStand as the first column
data_un(:,2) = ttb(:,7);    % Select Thrust as the second column
data_un(:,3) = ttb(:,8);    % Select FuelFlow as the third column
data_un(:,4) = ttb(:,3);    % Select TestDurMax as the fourth column
data_un(:,5) = ttb(:,11);   % Select PressuraPr as fifth column
data_un(:,6) = ttb(:,10);   % Select OxidizerFl as sixth column
data_un(:,7) = ttb(:,19);   % Select Total Cost as the seventh column

% Normalization
x_min = 0.9*min(data_un(:,7));
x_max = 1.1*max(data_un(:,7));
data(:,1) = (data_un(:,1)-1)/(6-1);% Special Normalization for TestStand (since there are 6)
data(:,2) = (data_un(:,2)-.9*min(data_un(:,2)))/(1.1*max(data_un(:,2))-.9*min(data_un(:,2)));
data(:,3) = (data_un(:,3)-.9*min(data_un(:,3)))/(1.1*max(data_un(:,3))-.9*min(data_un(:,3)));
data(:,4) = (data_un(:,4)-.9*min(data_un(:,4)))/(1.1*max(data_un(:,4))-.9*min(data_un(:,4)));
data(:,5) = (data_un(:,5)-.9*min(data_un(:,5)))/(1.1*max(data_un(:,5))-.9*min(data_un(:,5)));
data(:,6) = (data_un(:,6)-.9*min(data_un(:,6)))/(1.1*max(data_un(:,6))-.9*min(data_un(:,6)));
data(:,7) = (data_un(:,7)-x_min)/(x_max-x_min);

train = data(1:7,:);                    % use 7 rows for training data
test = data(8:11,:);                    % use 4 rows for testing data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fuzzy Determination
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TestStand and Thrust
% Training Data
p(:,1) = train(:,1);
p(:,2) = train(:,2);
t = train(:,7);             % Cost is the last data column
in_train = [p,t];
% Testing data
p2(:,1) = test(:,1);
p2(:,2) = test(:,2);
t2 = test(:,7);             % Cost is the last data column
in_test = [p2,t2];
p_1 = p;
p2_1 = p2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TestDurMax and PressuraPr
% Training Data
p(:,1) = train(:,4);
p(:,2) = train(:,5);
t = train(:,7);             % Cost is the last data column
in_train = [p,t];
% Testing data
p2(:,1) = test(:,4);
p2(:,2) = test(:,5);
t2 = test(:,7);             % Cost is the last data column
in_test = [p2,t2];
p_2 = p;
p2_2 = p2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FuelFlow and OxidizerFL
% Training Data
p(:,1) = train(:,3);
p(:,2) = train(:,6);
t = train(:,7);             % Cost is the last data column
in_train = [p,t];
% Testing data
p2(:,1) = test(:,3);
p2(:,2) = test(:,6);
t2 = test(:,7);             % Cost is the last data column
in_test = [p2,t2];
p_3 = p;
p2_3 = p2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FuelFlow and Thrust
% Training Data
p(:,1) = train(:,3);
p(:,2) = train(:,2);
t = train(:,7);             % Cost is the last data column
in_train = [p,t];
% Testing data
p2(:,1) = test(:,3);
p2(:,2) = test(:,2);
t2 = test(:,7);             % Cost is the last data column
in_test = [p2,t2];
p_4 = p;
p2_4 = p2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Predicting Training Data
fismat = readfis('m1_TS_Thrust_09');
ctrain(:,1) = (evalfis(p_1,fismat));
fismat = readfis('m1_TDM_PPr_10');
ctrain(:,2) = (evalfis(p_2,fismat));
fismat = readfis('m1_FF_OFL_10');
ctrain(:,3) = (evalfis(p_3,fismat));
fismat = readfis('m1_FF_Thrust_10');
ctrain(:,4) = (evalfis(p_4,fismat));

e_ctrain(:,1) = ((train(:,7)-ctrain(:,1))./train(:,7))*100;
e_ctrain(:,2) = ((train(:,7)-ctrain(:,2))./train(:,7))*100;
e_ctrain(:,3) = ((train(:,7)-ctrain(:,3))./train(:,7))*100;
e_ctrain(:,4) = ((train(:,7)-ctrain(:,4))./train(:,7))*100;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicting Testing Data
fismat = readfis('m1_TS_Thrust_09');
ctest(:,1) = (evalfis(p2_1,fismat));
fismat = readfis('m1_TDM_PPr_10');
ctest(:,2) = (evalfis(p2_2,fismat));
fismat = readfis('m1_FF_OFL_10');
ctest(:,3) = (evalfis(p2_3,fismat));
fismat = readfis('m1_FF_Thrust_10');
ctest(:,4) = (evalfis(p2_4,fismat));

e_ctest(:,1) = ((test(:,7)-ctest(:,1))./test(:,7))*100;
e_ctest(:,2) = ((test(:,7)-ctest(:,2))./test(:,7))*100;
e_ctest(:,3) = ((test(:,7)-ctest(:,3))./test(:,7))*100;
e_ctest(:,4) = ((test(:,7)-ctest(:,4))./test(:,7))*100;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NN input
ctrain_nn = ((ctrain(:,1:4)))';
ctest_nn = (ctest(:,1:4))';
tar_train = (train(:,7))';
tar_test = (test(:,7))';
dataMinMax = minmax(ctrain_nn);     % Quantity to give the neural network for size determination

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Neural Network Creation (Backpropagation)
% Create a new network with the newff command (Backpropagation)
% net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
% traingdx is for batch training with adaptive learning rate and momentum
% learngdm is for gradient descent with momentum
% sse is the the sum-squared error criterion
nBP = newff(dataMinMax, [3 1], {'logsig','logsig'}, 'traingdx', 'learngdm', 'sse');
nBP.trainParam.goal = 0.001;        % Desired maximum error
nBP.trainParam.epochs = 3000;       % Maximum no. of epochs
nBP.trainParam.lr = 0.01;           % Learning rate
nBP.trainParam.mc = 0.3;            % Momentum

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Training of network
nBP = init(nBP);                    % Initialize the new network
[nBP, trc] = train(nBP, ctrain_nn, tar_train);  % Train the network

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing Data
c_results = [];
c_results = sim(nBP, ctest_nn);     % Simulating the network using test data
c_results = c_results.*(x_max-x_min)+x_min;     % Multiplying by normilization factor
tar_test = tar_test.*(x_max-x_min)+x_min;       % Multiplying by normilization factor
error = [];
error = ((tar_test-c_results)./tar_test).*100
err = mean(abs(error))              % Mean error
sserror = 1/(length(c_results)).*sum((tar_test - c_results).^2)
c_results = c_results';
error = error';
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Training Data
c_results = [];
c_results = sim(nBP, ctrain_nn);    % Simulating the network using test data
c_results = c_results.*(x_max-x_min)+x_min;     % Normilizating factor
tar_train = tar_train.*(x_max-x_min)+x_min;     % Normilizating factor
error = [];
error = ((tar_train-c_results)./tar_train).*100
err = mean(abs(error))              % Mean error
sserror = 1/(length(c_results)).*sum((tar_train - c_results).^2)
c_results = c_results';
error = error';
```

11.6. Method 3: RBFN Development

```
% m3RBFN.m
% 3/14/04
% Method 3: RBFN Development
% Properly normalized, 7 training, 4 testing, double fis 10
% filling in missing data in columns 5, 8, 11 (mean or median)
% RBFN
close all;clear all;
% ANFIS HACEM Developing & Processing for Engines
% the variables used are:

input_name=['DuratDd   ';'NoTest    ';'TestDurMax';'Fuel      ';'Pressurant';...
        'Oxydizer  ';'Thrust    ';'FuelFlow  ';'FuelPresFl';'OxidizerFl';...
        'PressuraPr';'ThrustMeas';'Cooling?  ';'GimbalAxes';'Safety?   ';...
        'Handling? ';'TestStand ';'FacilitMod';'TotalCost '];

%   1   2  3   4   5   6   7      8      9   10    11    12  13  14  15  16  17  18  19
ttb=[9     2         70        5         4         1         4250      4.75  10.44        25        600       1         1
           2         0         0         3         3         550000;
   252     48        150       5         4         1         80000     64         0.55      139       800       0         1
           1         0         0         5         2         1592000;
   180     70        200       1         2         5         2500      30         25        25        150       0         0
           0         0         0         2         1         4433000;
   240     10        180       5         1         7         1740      174        172       12        1512      0         1
           0         0         0         2         1         2203000;
   360     14        34        23        7         1         60000     0          12        140       2000      1         0
           0         0         0         3         1         1772000;
   75      10        180       5         0         1         1800000   2035  0     4620      0          1        1         0
           0         0         4         1         12051000;
   45      25        200       8         4         6         5450      3.6        20        23.4 400   1        0         0
           1         1         3         1         702000;
   540     153       100       24        0         1         10000     0          0          150      3000      1         0
           0         0         0         3         1         1590000;
   120     42        8         2         9         1         645000    267        200       1667 15000          0         0
           0         0         1         1         2         4935000;
   120     21        45        23        0         6         150         1        0          1.78      1000  1  1         0
           1         1         2         2         1713000;
   105     17        60        22        0         1         10000     9.52  0     23.81  0    1         0         0         0
           0         3         1         1540966];

% Filling in missing data
% Correcting for Pressurant
pres = median(ttb(:,5));
[a,b] = find(ttb(:,5)==0);
ttb(a,5) = pres;
% Correcting for FuelFlow
ff = mean(ttb(:,8));
[a,b] = find(ttb(:,8)==0);
ttb(a,8) = ff;
% Correcting for PressuraPr
ppr = median(ttb(:,11));
[a,b] = find(ttb(:,11)==0);
ttb(a,11) = ppr;
% Correcting for FuelPresFl
fpf = median(ttb(:,9));
[a,b] = find(ttb(:,9)==0);
ttb(a,9) = fpf;

x_min = 0.9*min(ttb(:,19));
x_max = 1.1*max(ttb(:,19));

data(:,1) = (ttb(:,1)-.9*min(ttb(:,1)))/(1.1*max(ttb(:,1))-.9*min(ttb(:,1)));
data(:,2) = (ttb(:,2)-.9*min(ttb(:,2)))/(1.1*max(ttb(:,2))-.9*min(ttb(:,2)));
data(:,3) = (ttb(:,3)-.9*min(ttb(:,3)))/(1.1*max(ttb(:,3))-.9*min(ttb(:,3)));
data(:,4) = (ttb(:,4)-1)/(24-1);
data(:,5) = (ttb(:,5)-1)/(9-1);
data(:,6) = (ttb(:,6)-1)/(7-1);
data(:,7) = (ttb(:,7)-.9*min(ttb(:,7)))/(1.1*max(ttb(:,7))-.9*min(ttb(:,7)));
data(:,8) = (ttb(:,8)-.9*min(ttb(:,8)))/(1.1*max(ttb(:,8))-.9*min(ttb(:,8)));
data(:,9) = (ttb(:,9)-.9*min(ttb(:,9)))/(1.1*max(ttb(:,9))-.9*min(ttb(:,9)));
data(:,10) = (ttb(:,10)-.9*min(ttb(:,10)))/(1.1*max(ttb(:,10))-.9*min(ttb(:,10)));
data(:,11) = (ttb(:,11)-.9*min(ttb(:,11)))/(1.1*max(ttb(:,11))-.9*min(ttb(:,11)));
data(:,12) = (ttb(:,12));
data(:,13) = (ttb(:,13));
data(:,14) = (ttb(:,14)-0)/(2-0);
data(:,15) = (ttb(:,15));
data(:,16) = (ttb(:,16));
data(:,17) = (ttb(:,17)-1)/(6-1); % Special Normalization for TestStand (since there are 6)
data(:,18) = (ttb(:,18)-1)/(3-1);
data(:,19) = (ttb(:,19)-x_min)/(x_max-x_min);

train = data(1:7,:);                    % use 7 rows for training data
test = data(8:11,:);                    % use 4 rows for testing data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NN input
ctrain_nn = (train(:,1:18));
ctest_nn = (test(:,1:18));
tar_train = (train(:,19));
tar_test = (test(:,19));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Neural Network Creation (RBFN)
k = 6;                          % Number of clusters k
p = 4;                          % Number of p-nearest neighbors chosen
[clus,cent] = kmeans(ctrain_nn,k);  % Kmeans algorithm
ave_cost = [];                    % Finding the average cost of each cluster
for i = 1:k
   x = [];
   y = [];
   [x y] = find(clus(:,1)==i);
   ave_cost(i,:) = mean(data(x,7));
end
vari = zeros(k,1);                % P-nearest neighbor
n = [];
n_sort = [];
n = dist(cent');                  % Euclidean distances of each center for nearest determination
n_sort = sort(n,1);
for i = 1:k                       % For the number of clusters k
   a = 0;
   for j = 1:p
      x = [];
      y = [];
      [x y] = find(n(:,i) == n_sort(1+p,i));
      a = sum((cent(i,:)-cent(x,:)).^2) + a;
   end
   vari(i,1) = (1/p*a)^0.5;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Training
phi_k1 = [];
result1 =[];
for i = 1:size(ctrain_nn,1)        % For the length of the training data set
   for j = 1:k                     % Calculating the output
      phi_k1(i,j) = exp(-(sum((cent(j,:)-ctrain_nn(i,:)).^2))^0.5/vari(j,:));
   end
   phi_k1(i,:) = 1/sum(phi_k1(i,:))*(phi_k1(i,:));% Normalizing probablities to sum to one
   result1(i,1) = phi_k1(i,:)*ave_cost;% Multiplying the propabilities by the
end                                % average cost of the respective cluster

% Error calculations
res_train = result1.*(x_max-x_min)+x_min; % Un-normilizating output
tar_train = tar_train.*(x_max-x_min)+x_min;
error = [];
err = [];
```

```
error = ((tar_train-res_train)./tar_train).*100 % Error calculation
err = mean(abs(error))              % Mean error
sserror = 1/(length(res_train)).*sum((tar_train - res_train).^2) % Sum squared error

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing
phi_k2 = [];
result2 =[];
for i = 1:size(ctest_nn,1)          % For the length of the testing data set
    for j = 1:k                     % Calculate the output
        phi_k2(i,j) = exp(-(sum((cent(j,:)-ctest_nn(i,:)).^2))^0.5/vari(j,:));
    end
    phi_k2(i,:) = 1/sum(phi_k2(i,:))*(phi_k2(i,:));% Normalizing probablities to sum to one
    result2(i,1) = phi_k2(i,:)*ave_cost;% Multiplying the propabilities by the
end                                 % average cost of the respective cluster

% Error calculations
res_test = result2.*(x_max-x_min)+x_min;  % Un-normilizating output
tar_test = tar_test.*(x_max-x_min)+x_min;
error = [];
err = [];
error = ((tar_test-res_test)./tar_test).*100 % Error calculation
err = mean(abs(error))              % Mean error
sserror = 1/(length(res_test)).*sum((tar_test - res_test).^2) % Sum squared error
```

11.7. Method 3: RBFN Cluster Costs, Center Values, and Variance Values

For RBFN (11 for training)

		RBFN: k = 6, p = 4					
	Actual Costs	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.00431	0.14287	0.15005	0.15226	0.14491	0.15315	0.25676
2	0.085964	0.14697	0.14749	0.13535	0.18453	0.14104	0.24462
3	0.30859	0.15344	0.14703	0.14191	0.26302	0.14288	0.15172
4	0.13384	0.13147	0.14705	0.14868	0.28485	0.12378	0.16417
5	0.10007	0.19653	0.13449	0.1335	0.12317	0.2729	0.13941
6	0.90556	0.15415	0.14913	0.14835	0.15682	0.15031	0.24124
7	0.016221	0.142	0.14692	0.27997	0.15394	0.14288	0.1343
8	0.085808	0.33066	0.11491	0.12164	0.12164	0.18682	0.12432
9	0.34793	0.11553	0.3955	0.12344	0.12788	0.11586	0.12178
10	0.095446	0.14143	0.14137	0.29146	0.13734	0.14095	0.14744
11	0.081965	0.19152	0.12367	0.14041	0.13265	0.26627	0.14548
		Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
	Average Cost of Cluster	0.004983	0.32571	0.001346	0.001003	0.01761	0.31717
		Center					
		0.90783	0.19099	0.12698	0.3446	0.383	0.17733
		0.90811	0.24144	0.12733	0.22943	0.082282	0.10931
		0.43609	0.003759	0.54182	0.85902	0.18703	0.59273
		1	0.043478	0.63043	0.086957	0.93478	0.17391
		0.125	1	0.25	0.0625	0.4375	0.29167
		0	0	0.83333	0.83333	0	0
		0.004983	0.32571	0.001346	0.001003	0.01761	0.31717
		0.10478	0.11892	0.000626	0.045182	0.054315	0.31299
		0.045306	0.90889	0.067083	0.44648	0.04886	0.030288
		0.02921	0.32781	0.002163	0.003326	0.015806	0.31357
		0.17507	0.90834	0.034525	0.04253	0.077299	0.036562
		1	0	1	0	1	0.66667
		0	0	0.5	0.5	0	1
		0	0	0	0	0	0.5
		0	0	1	0	0	0
		0	1	1	0	0	0
		0.4	0	0.3	0.2	0.4	0.6
		0	0.5	0.25	0	0	0.5
		Variance					
		2.1051	2.4511	2.1051	1.9903	1.865	1.9923

For RBFN (7 for training)

		RBFN: k = 3, p = 2		
	Actual Costs	Cluster 1	Cluster 2	Cluster 3
1	0.00431	0.48523	0.27206	0.24272
2	0.085964	0.44197	0.2451	0.31293
3	0.30859	0.24892	0.27964	0.47144
4	0.13384	0.26562	0.2308	0.50358
5	0.10007	0.29516	0.45465	0.25019
6	0.90556	0.45267	0.28009	0.26723
7	0.016221	0.24188	0.47962	0.2785
8	0.085808	0.30066	0.41204	0.2873
9	0.34793	0.31521	0.35841	0.32637
10	0.095446	0.30613	0.41839	0.27548
11	0.081965	0.30267	0.43059	0.26673
		Cluster 1	Cluster 2	Cluster 3
	Average Cost of Cluster	0.31717	0.016461	0.001003
		Center		
		0.17733	0.3318	0.3446
		0.10931	0.10631	0.22943
		0.59273	0.51598	0.85902
		0.17391	0.63043	0.086957
		0.29167	0.5625	0.0625
		0	0.41667	0.83333
		0.31717	0.016461	0.001003
		0.31299	0.052992	0.045182
		0.030288	0.070636	0.44648
		0.31357	0.015766	0.003326
		0.036562	0.065078	0.04253
		0.66667	1	0
		1	0	0.5
		0.5	0	0
		0	0.5	0
		0	0.5	0
		0.6	0.4	0.2
		0.5	0	0
		Variance		
		1.6986	1.6986	1.6686

12. VITA

Holly Danker-McDermot was born in Miami, Florida. She graduated magnum cum laude from the University of New Orleans with a Bachelor of Science in Electrical Engineering in the spring of 2002. While attending the University of New Orleans as an undergraduate, in May of 2001 she was awarded the Electrical Engineering Outstanding Achievement Award. In May of the following year she was then awarded the Electrical Engineering Robert Lee Chandler Outstanding Graduate Award. She is currently a member of IEEE as well as the engineering honor society Tau Beta Pi.