STUDY OF AN INITIALIZATION METHOD AND STOPPING CRITERIA FOR NONLINEAR OPTIMIZATION

By

PRITHWIJIT GHOSHAL

Bachelor of Engineering in Chemical Engineering

Visveswaraya Technological University

Belgaum, Karnataka, India

2006

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2008

STUDY OF AN INITIALIZATION METHOD AND

STOPPING CRITERIA FOR NONLINEAR

OPTIMIZATION

Thesis Approved:

Dr R. Russell Rhinehart

Thesis Adviser

Dr Karen A. High

Dr Satish T.S. Bukkapatnam

Dr A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to place on record my sincere gratitude and thanks to Dr. R. Russell Rhinehart, my Advisor, for his guidance, suggestions, enthusiasm and encouragement he provided in my research work. I am also indebted to him for the financial support he extended during my stay at Oklahoma State University. I will cherish the knowledge he imparted to me and hope that I will be able to make good use of it in my future endeavors.

I would like to thank Dr. Karen High and Dr. Satish Bukkapatnam for being in my thesis advisory committee and for the help I received from them in completing my thesis. I would also like to thank Dr Eric L. Maase, Dr Arland H. Johannes, for their unhesitating help and encouragement. I also acknowledge the help rendered by the faculty, staff, and the graduate students of the department of Chemical Engineering at Oklahoma State University.

I acknowledge and thank my family for their support, and finally, I would like to thank Soumitra Ghosh, Kasturi Ghatak, Stacey Bridges, and Sayandeep Basak for their undying faith in me.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Optimization is the use of specific methods to determine the most cost effective and efficient solution to a problem or design for a process, making it one of the major quantitative tools used on industrial decision making. Optimization pervades the fields of science, engineering and business. In physics, for example, many different optimal principles are enunciated, which describe natural phenomena in the fields of optics and classical mechanics. Optimization is reflected in Statistical terms like “maximum likelihood,” minimum loss,” and “least squares”; and in business terms like “maximum profit,” “minimum use of resources,” “minimum cost,” and ”minimum effort”. Optimization is also important in engineering where a process can be described by a series of equations, or by experimental data. When a single performance criterion is considered, such as minimum cost, engineering optimization is used to find the values of the process variables which yield the best value of the performance criterion [1].

Optimization can be easily explained by an example:

Example 1.1: Minimize the function

$$f(x) = (x-3)^2 + 1 \qquad (1.1)$$

The function '*f*' *is* called the Objective Function (OF), the variable '*x*' is the Decision Variable (DV). The function can be plotted for various values of *x* which will reveal that the optimum for the function is at *x*=3, where the objective function attains a value of 1.

1.1. Optimization methods

There are two main categories of optimization. One is constrained optimization and the other is unconstrained optimization. Constrained optimization seeks the optima in a restricted region, which is defined using equality and inequality constraints, which are usually based on the probability of finding an optimum existing in the range. Unconstrained optimization seeks to determine an optimum in a range from $-\infty$ to $+\infty$. These two classes are manly used in practice to attain economic benefits and empirical modeling [2]. For example, the optimization of a set of process setpoints seeking to minimize process costs falls under the former, and optimization of model parameters to fit experimental data falls under the latter and it is generally called empirical modeling. This work mainly deals with empirical model optimization.

1.2. Empirical Modeling

In many fields, it is incumbent to describe a series of data points in terms of an empirical relation, which is easy to understand and implement. If there is only one independent variable in the data representation, they can be plotted in Cartesian coordinates and a line drawn through the points can be the graphical representation of the data points [7]. In real life, however, the data points can be dependent on more than one independent variable, which makes it more difficult to graphically represent the data. In these situations, it becomes necessary to find a functional form to represent the data. The functional form is

of particular interest since it can be easily implemented in calculations on computers, and because of the ease in interpolation between data points.

Typical relations for empirical models might be [1]:

$y = a_0 + a_1 x_1 + a_2 x_2 + \cdots$	linear in the variables and coefficients, i.e they don't have any exponents or indices associated with them.
$y = a_0 + a_{11} x_1^2 + a_{12} x_1 x_2 + \cdots$	linear in the coefficients, nonlinear in the variables (x_1,x_2)
$G(s) = \frac{1}{a_0 + a_1 s + a_2 s^2}$	nonlinear in all the coefficients
$Nu = a(\mathrm{Re})^b$	nonlinear in the coefficient b (Nu: Nusselt Number; Re: Reynolds number)

It has to be noted that the last two examples can me mathematically manipulated to give us linearized expressions, but they are nonlinear when considered as they are presented above.

The determination of the coefficients of a model from empirical data can be done using the principles of least squares. To compensate for the errors involved in experimental data, the number of data points should be greater than (about 3 times) the number of coefficients in the model. Least squares is just the application of optimization to obtain the "best" solution of the equations formed by implementing the data points to the model. In simpler terms, the sum of the squares of the errors between the predicted and the

experimental values of the dependent variables for each data point is minimized. The resulting model will be the closest functional representation of noisy experimental data [1].

1.3. Global Optimization

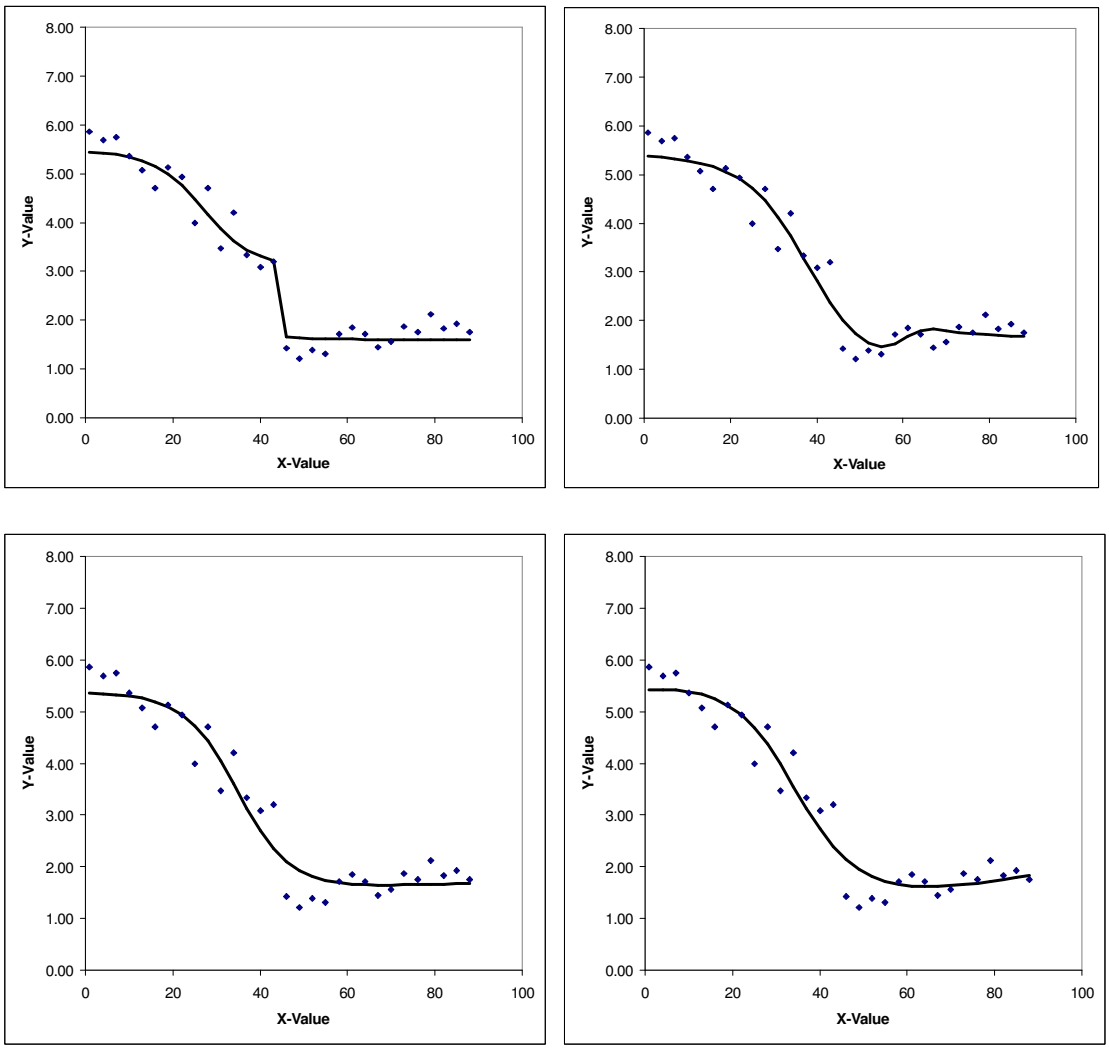
In many optimization problems, there are one or more solutions, all termed as local minima, and the best solution, i.e. the solution that returns the lowest objective function value, is termed as the global minima. This is the most sought after solution of them all.

A few examples of common multivariable optimizers used in the industry and in academia include, Marquardt-Levenberg, Gauss-Newton, Nelder-Mead Simplex, Hooke-Jeeves, Broydon-Fletcher-Goldfarb-Shanno, and successive quadratic. The common element to all of these optimizers is in the fact that it generates only one optimum for a given starting point, and there is no guarantee that it is the best solution. These optimizers are consequently termed as local optimizers.

The following example shows a series of data points being modeled by a neural network. The training of the said network yields a series of curves which have distinct differences between them. In each case, the data points are the same, i.e. the same process is being modeled using different initial values to start the optimization, but the neural network produces different curves to fit the same data.

Fig 1.1 (Clockwise from top left: a to d)

A Neural Network being trained for a set of process data.



The model prediction in Fig 1.1a is significantly different from the rest of the figures, with sharp bends in the model. Fig 1.1b displays a smooth curve, but I notable bump is observed between x values of 40 and 65. figures 1.1c and 1.1d may look identical, but a close inspection reveals that the tail of the curve in the former is flat, but in the latter, it is observed to curve upwards. This example clearly indicates the fact that a single model can give us more than one result.

This brings us to the realm of global optimization, where there is a need to seek out not just “a” solution, but the “best” solution to an objective. There are several developing algorithms being used and studied, and even though they are effective, it has to be noted that none of these techniques actually guarantee identification of the global optima. A few are enumerated below [1].

Simulated Annealing: This refers to a class of metaheuristics based on an analogy to the annealing of metals. The method depends on randomization to diversify the search, both in selecting a move to evaluate (all moves to neighboring points is equally likely) and in deciding whether or not accept the move. The basic SA algorithm can use the metropolis algorithm (Johnston *et al.*, 1989) to determine move acceptance, where downhill moves (where the difference in function values of the previous and present point is less than zero) are always accepted, and uphill moves (the above mentioned difference is greater than zero) are accepted with a probability.

Tabu Search: The basic idea involves allowing the algorithm to make moves that would not be allowed in a conventional local optimization program, thus the term “Tabu”. An example of this would be to change search directions or to make large steps when the optimizer approaches an optimum, the intention being to skip the present valley in hopes that a better optimum might then be found. The tabu moves are usually specified as moves to solutions with particular attributes. The moves are also specified to keep previously performed moves from being reversed, or to prevent previously visited solution from being revisited. It is widely accepted in the field of Operations research.

Unfortunately, there is no general purpose tabu search software available, though it has been implemented in numerous problems.

Genetic Algorithms: They are another idea which removes a major drawback of simulated annealing and tabu searches. Both of the latter operate by transforming single solution at a single step. The genetic algorithms, on the other hand, work with a population of solutions, i.e. a set of possible solutions, and this population is modified during each iteration by replacing one or more individuals (a single solution in the set) with new solutions, which are created by combining two individuals (crossover), or by changing an individual (mutation). The procedure is inspired by the evolution of populations of living organisms, whose chromosomes undergo crossover and mutation due to reproduction.

Multistart Methods: they use standard, widely available nonlinear programming methods, i.e. local optimization techniques, in the search logic. The difference here lies in the fact that instead of using only one starting point, a series of points are used, and the optimizer is run for all the starting points, and then the best solution is selected as the global optimum. This method is simpler to use compared to the other methods discussed earlier since they do not involve added or new heuristics to the solution scheme, and they use optimization methods that have been effectively used and understood. The drawback lies in the fact that most of the solutions deal with local optima and this leads to a large amount of computing time going to waste. The starting points can be chosen randomly, or can be chosen based on a specific range of values. When we consider randomly chosen

points, there are various logics as described by Rinnoy, Kan and Timmer (1987, 1989) and more recently by Locatelli and Shoen (1999) which can be used [1].

The present study explores the application of a global optimization search logic developed by Rhinehart and Iyer [4] for neural network training. The basis for the idea is in the mathematics involved in engineering reliability and in the training of neural networks which is effectively the nonlinear empirical modeling of the parameters of a neural network.

1.4. Stopping Criteria

A numerical optimization routine will always need a stopping criterion. It becomes necessary since it is the only means of stopping the algorithm once the optimum has been reached. The criterion should desirably stop the search when subsequent changed in the decision variable do not cause any improvements in the objective function value.

Some of the commonly implemented stopping criteria include

1. A threshold in the objective function value, which terminates the optimization process when the OF value is less than the set value.

2. A threshold change in the objective function value, which terminates the optimization process when it observes no change in the OF value.

3. A threshold change in the decision variable is another widely used criterion, which terminates the process when it observes no change in the DV values.

4. A threshold change in the number if iterations, which terminates the optimization after carrying out a certain number of iterations irrespective of whether the desired values for the parameters are achieved.

5. A threshold value on the square of the error between previous and present objective function values or decision variable values.

6. A threshold value on the first derivative of the objective function approaches zero, indicating that the objective function is at the bottom of a valley, i.e. the optimum.

7. A rise in the Standard Square deviation or Root mean Square of a validation set [2].

Factors 1 to 5 require an approximate knowledge of the optimum (before the optimization is carried out) to set up the thresholds. This is important since a loose threshold (set way away from the optimum) can lead to the procedure stopping before the optimum is attained. On the other hand, if the threshold is set far below the optimum, the optimizer may never find the optimum or it might take an unnecessarily large amount of time and computing power to find it, both of which are undesirable [2]. Factor 6 has the obvious disadvantage that it requires the objective function to be relatively simple to ensure that the derivative is known. More complex functions can use derivative knowledge using numerical methods, but the approximation can reduce the sensitivity of the criteria in general. Factor 7 doesn't use the validation set in the optimization itself, and this can be a detriment to its proper application.

Numerically, when these ideas are implemented, the stopping criteria usually involve observing two or more successive values of the decision variable or the objective function. As the optimizer approached on optimum, the step sizes decreases and consequently, the difference between the successive function values decreases. When no significant difference is observed in the function values, which is determined by comparing the actual difference against a pre set threshold, the program terminates. This procedure, however, has one serious disadvantage. Small step lengths do not occur only when the optimum is nearby, but also when the search is moving through a narrow valley, where the ΔDV values are small, but the ΔOF values could be large. In this case the aforementioned difference (in this case the ΔDV) can go below the threshold before the sought optimum is actually reached [2]. A similar situation occurs when the optimizer is moving over a vary wide valley, where the opposite is true, i.e. we have small ΔDV values but small ΔOF values, and the threshold in the ΔOF values can lead to a premature termination of the trial.

The probability of the optimizer attaining the global optimum depends on the initial guess that starts the trial. If the initial guess is too far from the global solution, the optimizer either 1) takes a long time to get to the appropriate values, or 2) becomes stuck in a local optimum. In these cases, it is convenient and prudent to restart the trial with a new starting guess. Hence, it is required to fix a maximum number of iterations within which the optimizer should find the optima. In case the search is not complete by the time the maximum limit is reached, the search is terminated, a new set of initial values are chosen and the trial is started again.

The various stopping criteria discussed above are scale dependent, starting point dependent, and optimization algorithm dependent, and the right choices require human supervision. Most of them also require *a priori* knowledge of the objective function under consideration [7]. This should be avoided when we evaluate optimization algorithms, since they can lead to misleading results. For example, in a practical situation, there might be a need to optimize a process model to obtain the values associated with it. Since no information about the threshold value of the process model (objective function) is available, it is quite difficult to set up the right threshold value.

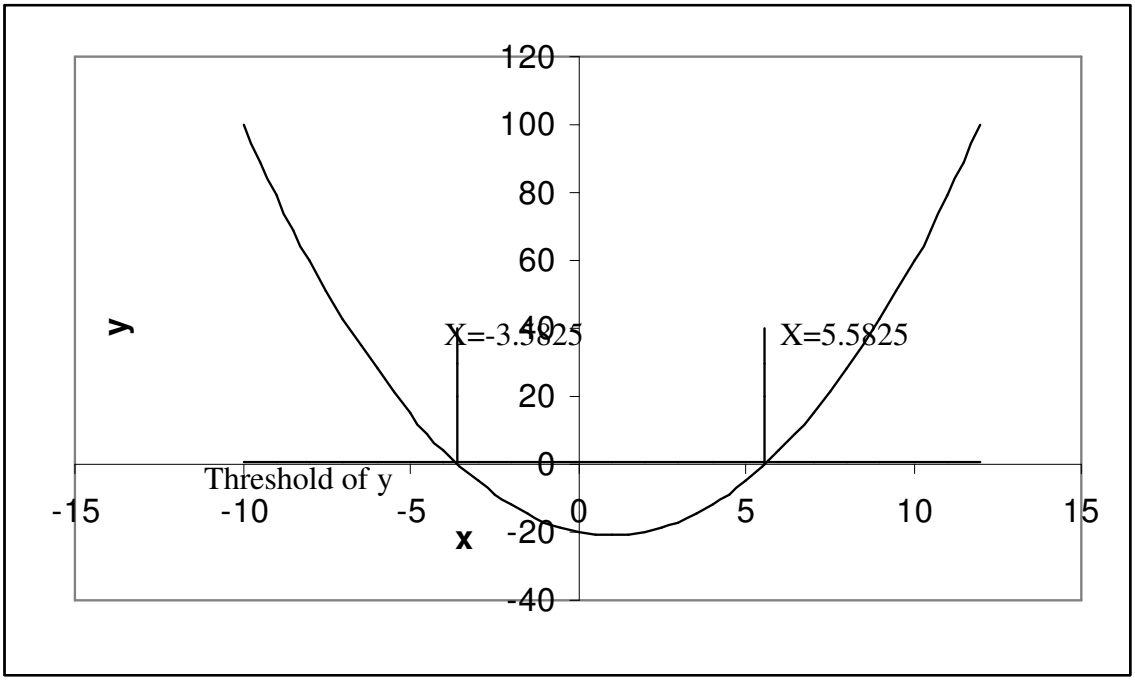
Consider the following example:

Example 1.2: Minimize

$$f(x) = x^2 - 2x - 20 \qquad (1.2)$$

As illustrated in Figure 1.2, the optima for this function is at x=1, where the objective function has a value of -21. If the user were not to know this and use a threshold value for the objective function to be close to zero, the trial would stop at x=-3.5825 or at x=5.5825, which would be the roots of the polynomial equation, but not the minima.

Fig 1.2 Optimization with threshold on objective function



We thus realize that the choice of most stopping criteria requires *a priori* information, and they can be scale dependent, application dependent, starting point dependent, and optimization algorithm dependent. Selection of the right stopping criterion feature or value would thus be a question of human supervision in the end [7].

The present work attempts to use the stopping criteria proposed by Cao and Rhinehart [3], for least squares optimization. This criterion is scale free, requires no prior knowledge of the optima, and stops the iteration when there is no statistical improvement in the data. The stopping criteria is combined with an initialization method proposed for neural network training [4] in order to provide a simple multistart global optimizer.

CHAPTER II

DESCRIPTION OF METHODS USED

Considering nonlinear optimization problems, the biggest issue is the tendency of the optimizer to get stuck in local minima. One of the ways to alleviate this problem is to run the optimizer repeatedly, starting with values based on a grid on the surface of the function or randomly generated values. Good examples of such applications are in neural network training, which always involves an optimization procedure to determine the weights of the network. Sha *et al*. have reported the use of 25 random starts in the use of neural networks for ship design. Park *et al*. have reported the results on prediction of sunspots based on 10 random starts.

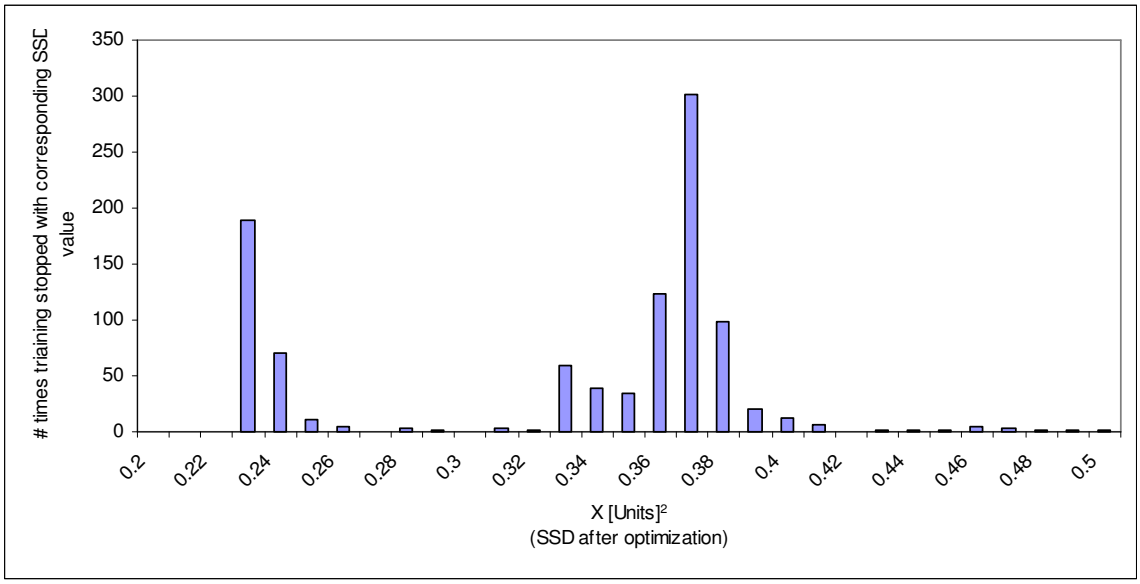
Rhinehart and Iyer [4], established a theoretical basis for the choice of the number of random starts in neural network training. The obvious implications of the study were that it can be extended to any other nonlinear optimization procedure. The concept used for this development was the “Best-of-N” or the “Weakest-Link-in-the-Chain” analysis. The present study applies this concept in regression modeling.

2.1. The Best-of-N or Weakest-link-in the-Chain Analysis

A chain is only as strong as its weakest link. In other words, the strength of a chain on N links, each of whose strengths is a distributed variable, is the strength of the weakest link. When we consider an optimization problem where the optimizer is used repeatedly, starting with randomly selected values, each individual optimization can be analogous to a link in a chain. The performance of the optimizer on our case is determined by the Sum-of-Squares Deviation (SSD) compared to a data set. This value is analogous to the strength of a link. The lowest error of several random starts is the strength of the weakest link. Consequently, the weakest link would mean the best solution among the repeated optimizations.

To further develop the idea, consider the following case study [5]. In it, a neural network was trained 1000 times, from 1000 independent random initial values for weights.

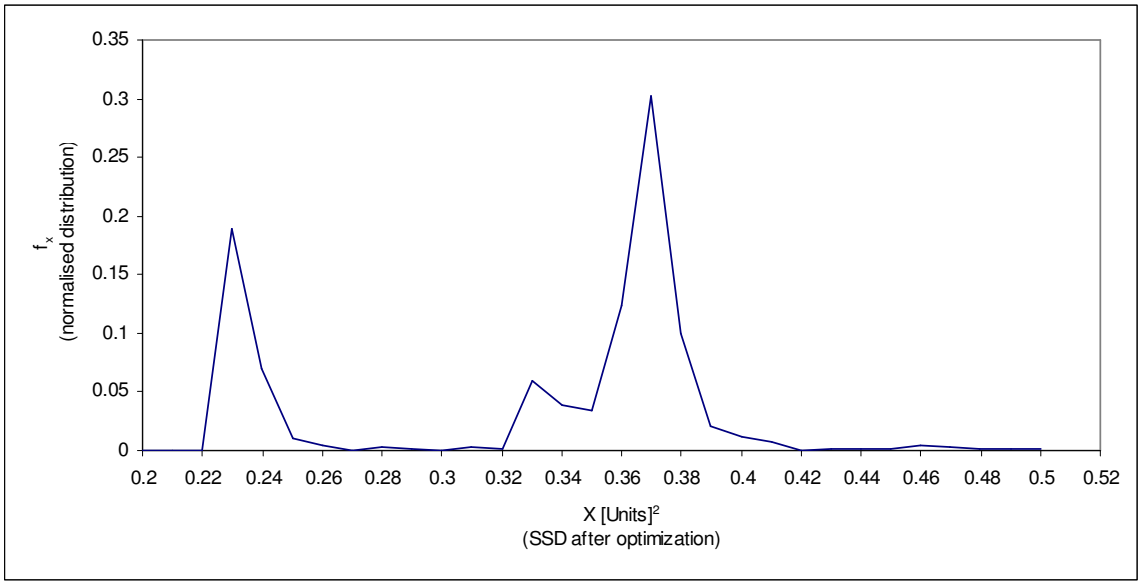
Fig 2.1 Distribution of SSD for 1000 NN trainings



From Figure 2.1. it is observed that sometimes the training ended with a SSD value of about 0.24 [unit^2]. This is Group 1, and represents the global optimum. Group 2 contains most of the training results; a broad local optimum centered around 3.75 as evidenced by the broad stopping range on the SSD.

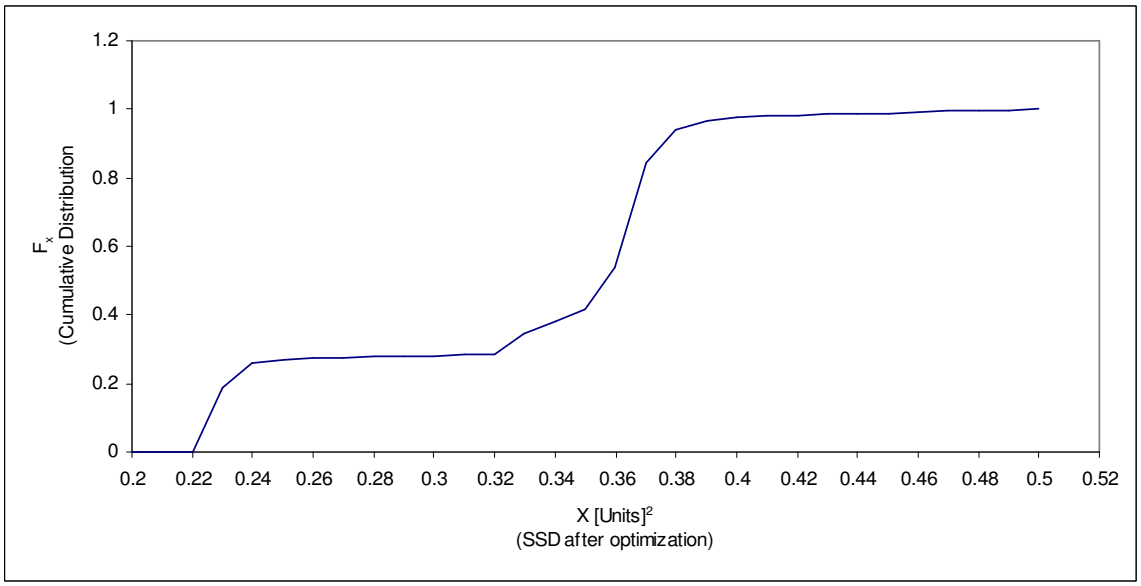
If connected by a smooth curve, and normalized so that the area under the curve equals 1, Figure 2.1 becomes Figure 2.2.

Fig 2.2 Normalized distribution fore 1000 NN trainings



The cumulative distribution can be obtained by integrating f_x over all x values. Figure 2.2 would thus yield Figure 2.3.

Fig 2.3 Cumulative Distribution for 1000 NN trainings



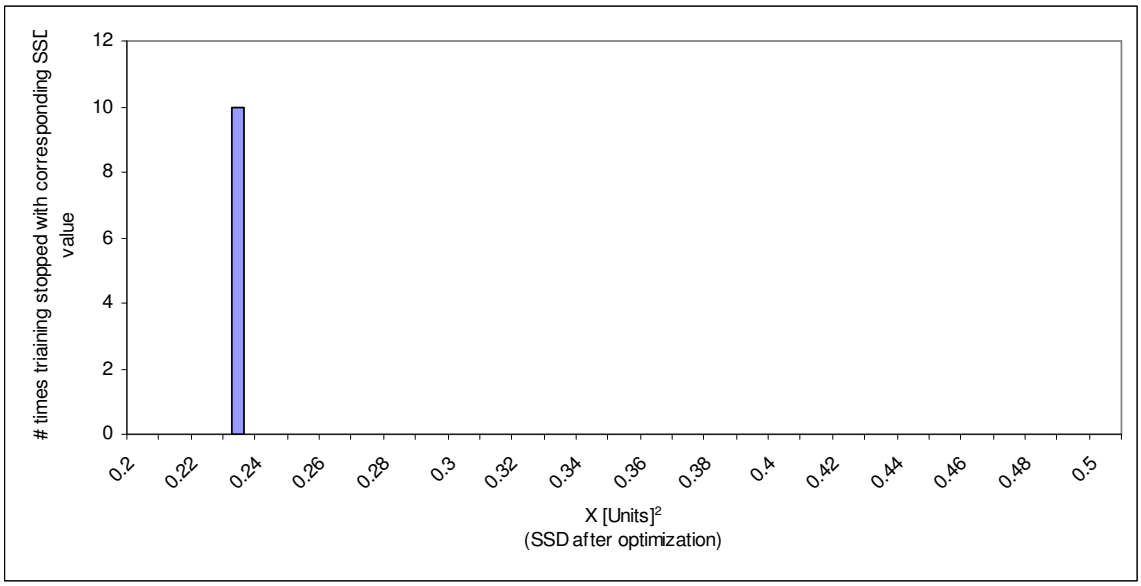
F_x in fig 2.3 reads as the fraction of events which had a corresponding or lower F_x value for a given x value. For example, 50% of the events stopped with an x value of

0.37[unit^2] or lower, or at a specific value of x = 0.23, only 10% or fewer events resulted in a better x value.

Figure 2.3 also reveals that about 20% of the trainings fell in Group 1, and the remaining 80% fell in Group 2. We also note that the figure is based on the initial Figure 2.1 which was a representation of the results of 1000 separate trainings.

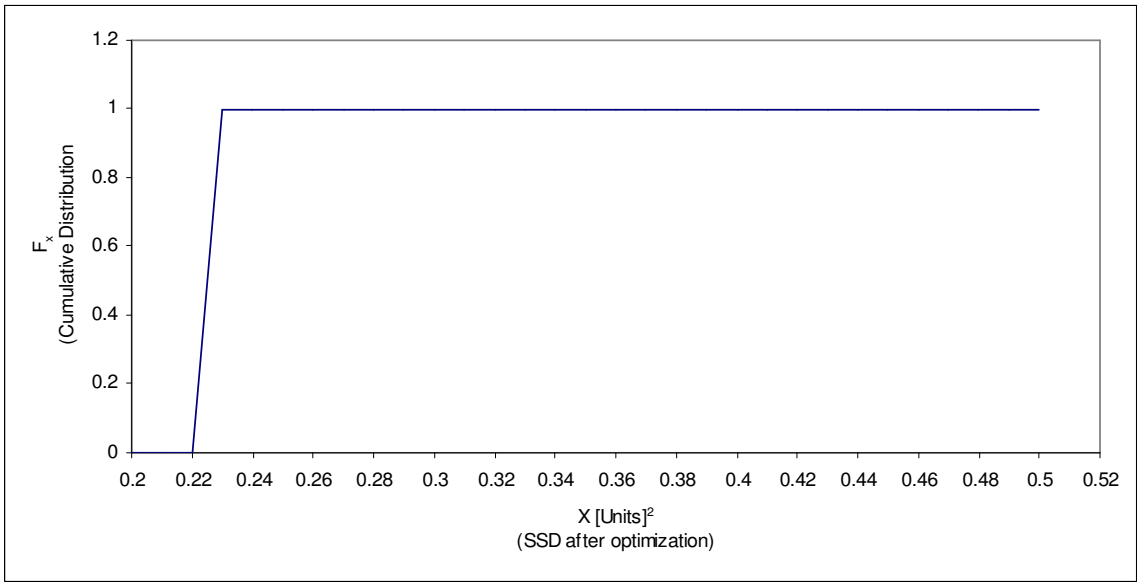
Now, if the neural network in consideration were to be trained about 100 times, 20% of the results would be expected to be in group 1, and 80% in group 2. The "best of 100" training histogram is shown in Figure 2.4. It has to be noted that in this case study, there was at least one point from Group 1 in each of the 10 cases, which in turn leads to only one bar in Figure 2.4.

Fig 2.4 Best of 100 training histogram



And the CDF would be,

Fig 2.5 Best of 100 cumulative distribution

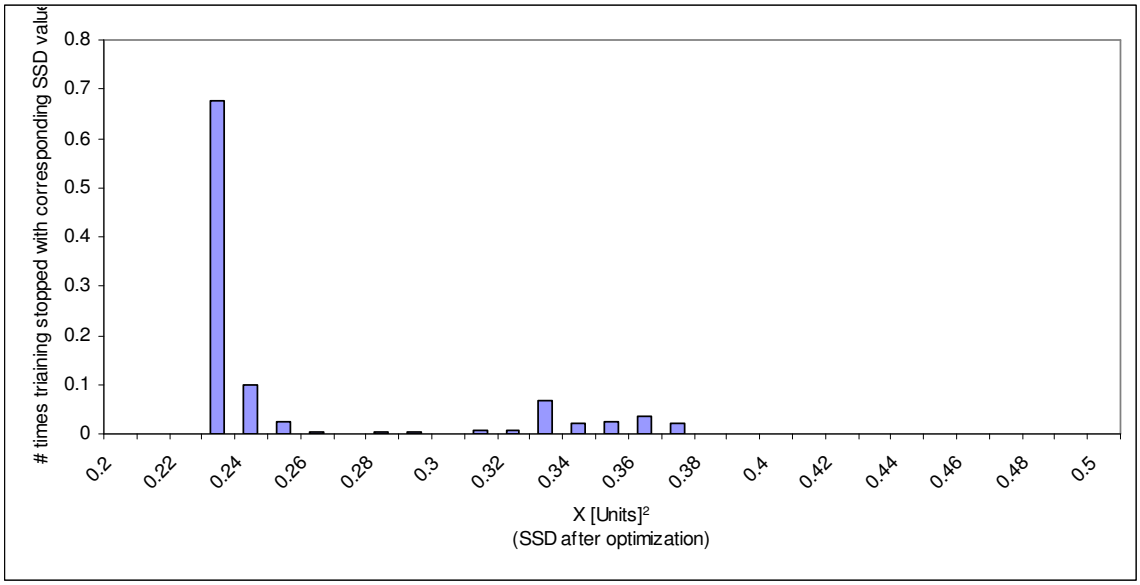


So, using N = 100 just about guarantees that the best of 100 will find the global optimum x-value of about 0.23[unit^2].

If a best-of-5 strategy is employed, with 20% expected in Group 1, it would be expected that 1 out of 5 would be in Group 1. In reality, some sets of 5 will have no values in Group 1, and some sets of 5 will have 2, 3, 4, or even 5 values falling in Group 1. Thus, in a best-of-5 strategy, the chance that one of five ends up in Group 1 is better than 20%, but there is still a possibility that none will.

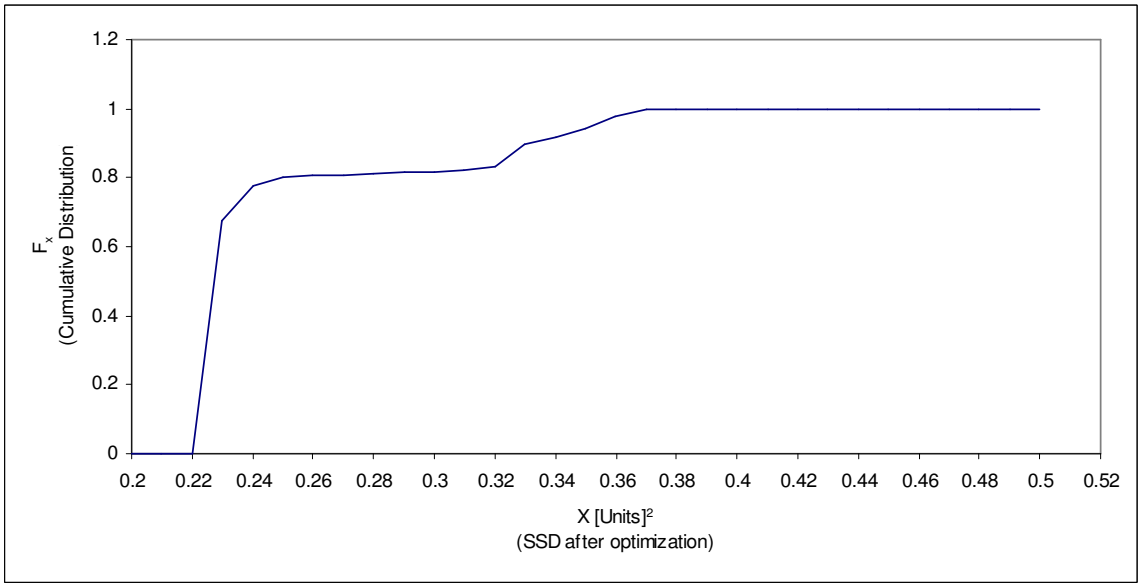
After 248 trials, the histogram for a best-of-5 is shown in Figure 2.6.

Fig 2.6 Best of 5 Training Histogram



From which the CDF is

Fig 2.7 Best of 5 Cumulative Distribution



From this illustration, 50% of the best-of-5 would end up on an x value of 0.23 [unit^2] or less.

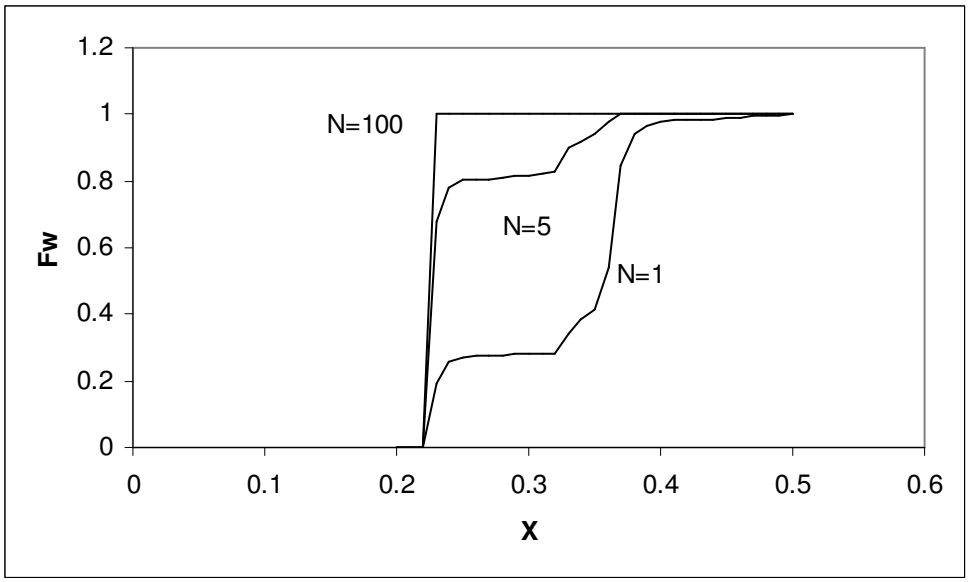
From Figure 2.7, we can infer that at x = 0.23 [unit^2] would give us one of the best 15% of all possible stopping places.

In the process of developing a logic to define the desired number of independent starts, define F_W as the confidence that at least one of the values generated is lower than or equal to a value in Group 1, i.e. an acceptable representation of the global optima. It can also be described as a representation of the CDF for the weakest link, from N links.

First, observe how F_w changes with N:

If it is desired that 99% of the trainings should find one of the best 10% the possible stopping outcomes, from Figure 2.8, 0.99 on F_W at N = 5 indicates that 99% of the stops will end up with a value of x = 0.37[unit^2] or less, which, Figure 2.3. then indicates is only in the best 85% of the best possible outcomes. From Figure 2.5, with N = 100, F_W = 0.99 reveals that 99% of the best-of-100 stops will have an x-value of 0.24 or less. Figure 2.3 indicates that this would be in the best 19% of all possible values. This brings us to an observation that F_W improves with an increasing N.

Fig 2.8 Cumulative distribution with changing N



The idea can also be mathematically developed using probability [4]. Consider N independent training runs. The probability that any single optimization has a SSD value, x, less then or equal to "*a*" is $F_X(a)$ where $F_X = \int_0^a f_X(x)\,dx$ is the value of the CDF at *a.*

Then the probability of x > *a* is $1 - F_X(a)$, and the probability that all points of the sample (in our case, this can be defined as the OF values upon stopping), of size N, have a value greater than a specific value, *a*, is $\left[1 - F_X(a)\right]^N$. Hence the probability that at least one of the elements has a lower value than, or equal to, a, is $1 - \left[1 - F_X(a)\right]^N$. Since we have used F_W to represent this earlier, we get the following expression:

$$F_W(a) = 1 - \left(1 - F_X(a)\right)^N \qquad (2.1.1)$$

Equation (2.1.1) explicitly defines the value of one of the three variables, in terms of the values of the other two. To reiterate them,

- N — The number of random, independent optimization starts from which the best will be chosen.
- X — The sum-of-squared deviations on any individual optimization.
- $F_X(a)$ — The fraction of random starts which would result in a value of X less than or equal to "a," and $0 \le F_X(a) \le 1$.

For Example, if $F_X(a)$ has a value of 0.2 this means that the X-value for the SSD is one of the best (lowest) 20% possible values. W is the best (lowest) value for x out of N starts. $F_W(a)$ is the fraction of the Best-of-N X-values that result in a value of W less then or

equal to "a," ≤ $F_W(a)$ ≤ 1. If $F_W(a)$ has a value of 0.99, this means that there is only a 1% chance that the Best-of-N X-values will be worse.

However, the present study requires the determination of the required number of random starts, based on user-defined values of $F_W(a)$, the level of confidence, and $F_X(a)$, the percentage vicinity of the lower tail of the distribution, which the Best-of-N is expected to provide. This can be done by rearranging Equation (2.1.1) to give,

$$N = \frac{\ln\left(1 - F_W\left(a\right)\right)}{\ln\left(1 - F_X\left(a\right)\right)} \qquad (2.1.2)$$

2.2. Online identification of Steady State.

In this exploration, the end point of an optimization procedure is identified using the concepts of steady state identification instead of the conventional methods of setting up thresholds [3]. The optimization parameter in nonlinear optimization of empirical data is the Sum of Square Deviations (SSD) between the data and the model. It has been observed that the Root Mean Square of the deviations (RMS) drops to an asymptotic minimum with progressive iterations.

The novelty of the method lies in the evaluation of the RMS of a Random Subset (RMS RS) of the data (a different random subset for each iteration). This RMS RS appears as a noisy signal relaxing to its noisy steady state value as the iterations progress. By using a random subset of data, the noise is independently distributed, and, at steady state, when convergence has been achieved, the noise reflects the variance in the data. The noise is chi-square distributed, with an average equal to the standard error of the residual (model-

to-data mismatch). When the noisy signal reaches a statistical steady state, the optimization has reached a point where there is no statistically significant improvement in the OF with respect to model standard error, and consequently the optimization should be stopped. Since the test looks at signal-to-noise ratio; it is scale independent.

Paraphrasing the development by Rhinehart and Iyer [3], the design of this method is styled after the F-test type of statistic. It is the ratio of variances, R, as measures on the same set of data by two different methods.

The primitive way of estimating variance would be:

$$\hat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(X_i - \overline{X_N}\right)^2 \qquad (2.2.1)$$

The modification (or simplification) begins with a conventional exponentially weighted moving average or conventional first-order filter of a decision variable X_i. this requires little storage and s computationally fast. In algebraic notation:

$$X_{f_i} = \lambda_1 X_i + (1-\lambda_1) X_{f_{i-1}} \qquad (2.2.2)$$

where $0 < \lambda_1 < 1$.

If the previous filtered value $X_{f_{i-1}}$ is used to replace the sample mean, $\overline{X}_N$, a mean square deviation can be defined as:

$$v^2 = E\left(\left(X_i - X_{f_{i-1}}\right)^2\right) \qquad (2.2.3)$$

and can be estimated by:

$$\hat{v}^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(X_i - X_{f_{i-1}}\right)^2 \qquad (2.2.4)$$

Assuming that $\{X_i\}$ is uncorrelated, using the previous value of X_f, X_{i-1}, prevents autocorrelation between X_i and $X_{f_{i-1}}$, and allows one to easily estimate σ^2 and v^2.

Define:

$$d_i = X_i - X_{f_{i-1}} \qquad (2.2.5)$$

if the process is at steady state conditions and there is no autocorrelation in the sequential measurement, then X_i and $X_{f_{i-1}}$ are independent, then the variance on d is related to the variance on X and X_f [8]:

$$\sigma_d^2 = \sigma_X^2 + \sigma_{X_f}^2 \qquad (2.2.6)$$

Further, for the exponentially weighted moving average, when $\{X_i\}$ are independent and stationary, the variance on X_f from Equation (2.2.2) becomes [9]:

$$\sigma_{X_f}^2 = \frac{\lambda_1}{1-\lambda_1}\sigma_X^2 \qquad (2.2.7)$$

Equations (2.2.6) and (2.2.7) yield:

$$\sigma_X^2 = \frac{2-\lambda_1}{2}\sigma_d^2 = \frac{2-\lambda_1}{2}v^2 \qquad (2.2.8)$$

from which the noise variance can be estimated if v^2 is known.

$$\hat{\sigma}_X^2 = \frac{2-\lambda_1}{2}\hat{v}^2 \qquad (2.2.9)$$

However, Equation (2.2.4) is computationally expensive; so, use a filtered value instead of a traditional average:

$$v_{f,i}^2 = \lambda_2\left(X_i - X_{f_{i-1}}\right) + \left(1-\lambda_2\right)v_{f,i-1}^2 \qquad (2.2.10)$$

If the process is stationary:

$$E\left(v_{f,i}^2\right) = E\left(\left(X_i - X_{f_{i-1}}\right)^2\right) = v^2 \qquad (2.2.11)$$

So, Equation (2.2.10) is an unbiased estimate of v^2, and the variance of $v_{f,i}^2$ is:

$$\text{var}\left(v_{f,i}^2\right) = \frac{\lambda_2}{2-\lambda_2}\text{var}\left(\left(X_i - X_{f_{i-1}}\right)^2\right) \qquad (2.2.12)$$

which means that Equation (2.2.10) provides a computationally efficient, unbiased estimate of $\left(X_i - X_{f_{i-1}}\right)^2$.

Then the estimate of the noise variance from this first approach will be:

$$s_{1,i}^2 = \frac{2-\lambda_1}{2} v_{f,i}^2 \qquad (2.2.13)$$

Actually since Equation (2.2.10) requires $X_{f_{i-1}}$ one would compute Equation (2.2.10) before Equation (2.2.2) to eliminate the need to store the previous 'average'.

Using this method, $s_{1,i}^2$ will be increased from it's steady-state value by a recent shift in the mean. Such a measure could be used to trigger the not-at-steady-state condition. However the threshold is dependent on both the measurements and the unknown process noise variance.

The second method to estimate variance will use the mean squared differences of successive data. Define:

$$\delta^2 = E\left(\left(X_i - X_{i-1}\right)^2\right) \tag{2.2.14}$$

and δ^2 could be estimated by:

$$E\left(s_{2,i}^2\right) = \frac{1}{2}E\left(X_i - X_{i-1}\right) \tag{2.2.15}$$

However, Equation (2.2.15) is computationally expensive; so, use a filtered approach:

$$\delta_{f,i}^2 = \lambda_3\left(X_i - X_{i-1}\right)^2 + \left(1-\lambda_3\right)\delta_{f,i-1}^2 \tag{2.2.16}$$

Again, Equation (2.2.16) gives an unbiased estimate of δ^2.

When there is no autocorrelation in $\{x\}$ the second estimate of the noise variance would be:

$$s_{2,i}^2 = \frac{\delta_{f,i}^2}{2} \tag{2.2.17}$$

Taking the ratio of the two estimates of variance as determined by Equation (2.2.10) to Equation (2.2.14):

$$R_i = \frac{s_{1,i}^2}{s_{2,i}^2} = \frac{\left(2-\lambda_1\right)v_{f,i}^2}{\delta_{f,i}^2} \tag{2.2.18}$$

To summarize, use Equation (2.2.10) to calculate $v_{f,i}^2$, then use Equation (2.2.2) to calculate $X_{f,i}$, then use Equation (2.2.16) to calculate $\delta_{f,i}^2$, and then use Equation (2.2.18) to calculate R_i. Each are direct, no logic, low storage, low operation calculations. In practice, it would be preferable to compare $\delta_{f,i}^2 R_{crit}$ (R_{crit} is the threshold value of R) to $\left(2-\lambda_1\right)v_{f,i}^2$ to prevent the possibility of a divide by zero in Equation (2.2.18). For each

observed variable, the method requires the direct and simple calculation of three filtered values. In total, there are three variables to be stored, 10 multiplications, eight additions, and one comparison per observed variable.

There are three possible process behaviors which affect the value of *R*:

1. If the process data is at steady-state (process mean is constant, additive noise is independent and identically distributed), the value of R will be near 1.
2. If the process data mean shifts, or if the noise is autocorrelated, then R will be greater than 1. When there is a shift on mean, both the calculations of the mean will be influenced temporarily. The first calculation will increase more and persist longer, so R will be greater than 1 for a period of time, and that is the way that the not-at-steady-state condition can be identified.
3. If the sequentially sampled variable values alternate between high and low extremes, then R will be less then 1. This doesn't apply to optimization applications and is not considered in our study.

The actual value of R, when implemented, is in effect a ratio of two noisy variables,and thus inherently has a good degree of noise associated with it. This can lead to the value of R being a normal distribution, and thus a threshold of R = 1 might not necessarily mean that the actual value of R is 1. to account for his sort of discrepancy, it is advisable to use a threshold value of 0.85, to ensure that the actual value of R is as close to 1.

CHAPTER III

METHODOLOGY

3.1. Optimization routines:

3.1.1. Direct methods:

Direct Methods are those which require only objective values, not derivative knowledge, to proceed. It is assumed that f(x) is continuous and ▼f(x) may or may not exist but certainly is not available. These methods can be broadly classified into heuristic techniques and theoretical techniques. The former refer to search methods constructed from geometric intuition for which no performance guarantees other than empirical results can be stated. The following two heuristic methods are used in the present study:

1. R. Russell Rhinehart's heuristic optimizer
2. Hooke-Jeeves Pattern Search Method

R. Russell Rhinehart's Heuristic Method

The method resembles a Cyclic Search but incorporates a set of factors which cause the subsequent steps in a particular direction to expand or contract depending on the success of the step.

The algorithm is described below:

Step 1. Define:

The starting point $x^{(0)}$

The increments $\Delta^{(k)}$ for k = 1,2,3,…, N

The Expansion factor (Expand_factor) and

The Contraction factor (Contract_factor)

Step 2: $x^{(k+1)} = x^{(k)} + \Delta^{(k)}$

Step 3: Was a lower point found?

Yes: $x^{(k)} = x^{(k+1)}$.

$\Delta^{(k+1)} = \Delta^{(k)} * \text{Expand\_factor}$

No: $x^{(k+1)} = x^{(k)}$.

$\Delta^{(k+1)} = \Delta^{(k)} * \text{Contract\_factor}$

Step 4: Check for termination

Is the termination criteria satisfied?

Yes: Stop; current point approximates x^*.

No: Go to 2.

Hooke Jeeves Pattern Search

This algorithm was one of the first to incorporate the previous history of a sequence of iterations into the generation of a new search direction. It is basically a combination of a "exploratory moves" of the one-variable-at-a-time kind with "pattern" or acceleration moves regulated by a set of heuristics.

The algorithm is described below:

Step 1. Define:

The starting point $x^{(0)}$

The increments Δ_k for k = 1,2,3,…, N

The step reduction factor $\alpha > 1$

Step 2. Perform Exploratory Search

Step 3. Was exploratory search successful (i.e. was a lower point found)?

Yes: Go to 5.

No: Continue.

Step 4. Check for termination

Is the termination criteria satisfied?

Yes: Stop; the current point approximates x^*.

No: Reduce the increments:

k=1,2,3,…, N

Goto 2.

Step 5. Perform the pattern move:

$$x_p^{(k+1)} = x^{(k)} + (x^{(k)} - x^{(k-1)})$$

Step 6. Perform exploratory search using $x_p^{(k+1)}$ as the base point; let the result be $x^{(k+1)}$

Step 7. if $f(x^{(k+1)}) < f(x^{(k)})$

Yes: set $x^{(k-1)= } x^{(k)}$; $x^{(k)}=x^{(k+1)}$.

Go to 5.

No: Go to 4.

3.1.2. Gradient based methods:

The inherent problem in the direct methods is the excessive number of function evaluation required to locate the solution. This combined with the inherent desire to find stationary points motivates us to consider methods that employ gradient information to determine the search direction. The present study uses a Quasi-Newton search algorithm, namely, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) Method, which exclusively uses first derivative information.

The algorithm is described below:

Step 1. Define:

The starting point $x^{(0)}$

The increments Δ_k for k = 1,2,3,…, N

The step reduction factor $\alpha > 1$

Set search direction, $s^{(0)} = -\nabla f(x^{(0)})$

Hessian approximation, $\mathbf{A}^{(0)} = \mathbf{I}$

Step 2: Perform a Line Search in the search direction (s^k) to determine x^{k+1}

Step 3: Compute $f(x^{(k+1)})$ and $\nabla f(x^{(k+1)})$

Step 4: Check for termination

True: Report results and Stop.

False: Continue to step 5.

Step 5: Compute $\Delta x^k = x^{k+1} - x^k$

$$\Delta g^k = \nabla f(x^{(k+1)}) - \nabla f(x^{(k)})$$

Step 6: Compute A^{k+1} based on the following update formula:

$$\mathbf{A}^{k+1} = \left[\mathbf{I} - \frac{\Delta x^{(k)} \Delta g^{(k)T}}{\Delta x^{(k)T} \Delta g^{(k)}}\right] \mathbf{A}^{(k)} \left[\mathbf{I} - \frac{\Delta x^{(k)} \Delta g^{(k)T}}{\Delta x^{(k)T} \Delta g^{(k)}}\right]^{T} + \frac{\Delta x^{(k)} \Delta x^{(k)T}}{\Delta x^{(k)T} \Delta g^{(k)}}$$

Step 7: compute the search direction using $s^{(k+1)} = - \mathbf{A}^{(k+1)} \nabla f(x^{(k+1)})$

Repeat from step 2 onwards.

3.2. Simulation:

Since the objective of the study is to model nonlinear systems we need data sets to test our algorithm. The initial testing during the construction and debugging of the algorithm, was done on sets of contrived data, and the subsequent testing to observe the practical use of the algorithm was done on actual experimental data.

3.2.1. Contrived Data:

The contrived data used in the study were representations of nonlinear systems with a sufficient degree of noise incorporated in them. Considering a nonlinear system involving only two variables, the initial set of contrived data used in the study, only incorporated noise in the dependent variable. The further testing of the algorithm was done on a different set of data with noise incorporated in both the dependent and the independent variables, which would give us a better approximation of a real world process with measurement uncertainties. In both cases the data is scaled between 0 and 1 before it is implemented in the modeling procedure.

The following models were used in the modeling of the above described data:

1. Third degree Polynomial: It is the simplest way to represent a nonlinear system. Here, we have "y" as a nonlinear function of "x," but the power on each coefficient is unity, i.e. the model is linear in the parameters of optimization, and consequently, the regression modeling is trivial.

It can be represented as:

$$y = A + B\,x + C\,x^2 + D\,x^3 \qquad (3.1)$$

2. Neural Network. With the progression of order in the polynomial equation, we would find that the results are more accurate. The next step is to use a Neural Network. In this study, a two layered, bipolar sigmoidal neural network is used with two neurons in each layer. Larger neural networks are not used because it increases the computation time required by the computer for the evaluation of the weights.

3.2.2. Experimental Data:

Two-phase flow is the simultaneous flow of both gas and liquid phase fluids through a pipe or tube. There are five mail flow regimes associated with two-phase flow through pipes: bubble, slug, churn, annular, and mist. These flow regimes are characterized by the composition and flow characteristics of the fluid mixture. The present system under consideration is defined by the presence of air in a column of water.

The apparatus used consists of a vertical pipe for the air/water mixture, a control computer, pressure transducers, orifice meters, paired with control valves, piping, pressure gauges and rotameters for air flow and water flow respectively. The user can monitor and control the flow rates using the CAMILE control system. The flow rates of air and water are set using the control valves. Real time flow rates are monitored through the orifice meters.

The modeling objective in this experiment is to model the pressure drop of the system based on a set of predetermined modeling equations. The optimizer routine is used to determine the coefficients in the model.

It has to be noted that both models work only when the data provided is in one single regime, because the parameters being optimized have different values based on the different flow regimes.

Lockhart-Martinelli Correlation

The Bernoulli equation states that the mechanical energy of a fluid is constant between two points along a streamline. The pressure drop per unit length for a two-phase system between two points takes the form:

$$-\left(\frac{\Delta P}{L}\right)=\left(\frac{\Delta P_f}{L}\right)_{TP}+\rho_{TP}\cdot g \qquad (3.2)$$

$\left(\frac{\Delta P_f}{L}\right)_{TP}$ =frictional pressure drop for two-phase flow

$\rho_{TP} \cdot g$ =hydrostatic pressure drop

The frictional pressure drop term can be evaluated by using either of the following equations:

$$\left(\frac{\Delta P_f}{L}\right)_{TP} = (\phi_g)^2 \cdot \left(\frac{\Delta P_f}{L}\right)_g$$
$$\left(\frac{\Delta P_f}{L}\right)_{TP} = (\phi_l)^2 \cdot \left(\frac{\Delta P_f}{L}\right)_l \qquad (3.3)$$

Where, $\left(\frac{\Delta P_f}{L}\right)_g$ and $\left(\frac{\Delta P_f}{L}\right)_l$ are the single phase frictional pressure drops for the gas and liquid phased calculated at their individual fluxes. They are calculated from the following equations

$$\left(\frac{\Delta P_f}{L}\right)_g = \frac{2 \cdot f_g \cdot m^2 \cdot x_g^2}{\rho_g \cdot D}$$
$$\left(\frac{\Delta P_f}{L}\right)_l = \frac{2 \cdot f_l \cdot m^2 \cdot x_l^2}{\rho_l \cdot D} \qquad (3.4)$$

The (ϕ) terms are frictional multipliers that can be obtained from the Lockhart-Martinelli correlation, using the Martinelli multiplier which is defined as:

$$X^2 = \frac{\left(\frac{\Delta P_f}{L}\right)_l}{\left(\frac{\Delta P_f}{L}\right)_g} \qquad (3.5)$$

Which is then used in the following equations to yield ϕ_g and ϕ_l

$$\left(\phi_g\right)^2 = 1 + C \cdot X + X^2$$
$$\left(\phi_g\right)^2 = 1 + \frac{C}{X} + \frac{1}{X^2} \qquad (3.6)$$

"C" is a constant that can be found in the literature, and it can be optimized.

3.3. Application of the optimizer:

The previous pages describes the various algorithms and the models used in the study. The application of this information is done using the following basic algorithm, which is modified depending on the optimizer and the model used.

Step 1. Determine the number of data points to be used.

Step 2. Inputs:

Dependent variables: coefficients of the model selected:

Percentage of Confidence (fraction between 0 and1)

Best Fraction of the data set required (fraction between 0 and 1)

Percentage of the dataset to be used in the Steady State Stopping Criterion.

Step 3. Use the percentage of confidence and the best fraction, calculate the number of trials required (Num_Trials).

Step 4. Use the selected optimization routine to calculate the minima based on a random starting point. The stopping criteria used in the routines are:

1. Maximum number of iterations.
2. Steady State Stopping Criterion

Step 5. Repeat Step 4 for Num_Trials (the calculated number of trials) and store the

results of each trial, i.e. the Sum of Square Deviation and the values of the coefficients.

Step 6. Find the lowest Sum of Square Deviation (SSD) from Step 5.

Step 7. The coefficients corresponding to the lowest SSD will yield the global minima of the given objective, and thus the closest model fitting the data.

The function evaluations used in the optimization routine in Step 4, are a series of calculations which are used to determine the Sum of Square Deviation between the actual data and the points generated by the model based on the coefficients of that particular step. In the case of the Indirect method, the derivatives for the same are calculated based on a numerical forward difference approach, with an error order of one. Higher error orders and central difference approaches are avoided because of the increased number of calculations they require, and, thus increasing the time required for the computation.

3.4. Testing Best-of-N equation for best number of trials:

The Best-of-N formula is based on a pre-defined confidence level, and the best fraction of all the possible answers. The best way to test the validity of the formula is by letting the optimizer run for a very large number (perhaps 100,000) of runs. The following algorithm is then employed to determine the validity of the formula used:

Step 1. Use the Best-of-N formula to calculate the number of required runs.

Step 2. Use the data set of (say) 10,000 runs, calculate the value of the sum of square deviations that will correspond to the best fraction used in step 1.

Step 3. Select the calculated number of runs randomly from the data set.

Step 4. If at least one run yields answers less than or equal to the value calculated in step 2, the step is a success. If not, the step is a failure.

Step 5. Repeat steps 3 and 4, 1,000 times and count the number of successes in step 4.

Step 6. If the percentage of successes (calculated from step 5) is similar to the percentage of confidence used in the neural network formula (Step 1) then the validity of the formula cannot be rejected.

It has to be noted that the result obtained in Step 6 will not be exactly equal to the percentage of confidence used in the original formula. This can be attributed to the

amount of data acquired and the consequent changes in the standard deviation which is calculated based on the number of sets being considered.

3.5. Testing the Steady State Stopping Criterion:

The Steady State Stopping Criterion can be evaluated by plotting the sum of the square deviations with the filtered values against the number of trials. To test the criteria, the optimizer is run without the stopping criteria, and the parameters mentioned above, are plotted. The plots have to be observed to determine the accuracy of the predicted result and the optimum generated if the optimizer were to run based on a maximum number of iterations. If the results generated in both cases are the same, the Steady State Stopping Criterion can be validated.

CHAPTER IV

RESULTS AND DISCUSSION

The results obtained from both contrived data and experimental data are discussed below. Both contrived data sets were based on nonlinear models, and a series of nonlinear models were used to model them. The experimental data was based on Venkatram Padmanabhan's thesis results [2], as well as independently generated data for pressure drop in a two-phase flow apparatus.

4.1. Results from simulated data

Two sets of contrived data were used in this study. Both sets were based on nonlinear models of varying complexity. In order to make the data representative of actual experimental data, noise was added to it using normally distributed random numbers with a set variance. In the first set of data (designated in future as Set A), the noise was incorporated only on the dependent variable. This can be mathematically described by:

$$\hat{Y} = Y\left(X\right) + Y_{noise} \qquad (4.1)$$

In real experimental data, the inaccuracies caused by measuring devices create uncertainty in the value of the independent variables too. To represent this, a second set

of data (designated as Set B), has noise incorporated in both the dependent variable and in the independent variable. This can be mathematically represented by:

$$\hat{Y} = Y\left(X + X_{noise}\right) + Y_{noise} \qquad (4.2)$$

The data is fed to the optimizer and the resulting set of terminal values are used to check if the predicted curve fits the data or not. The goodness of the fit is checked based on the Sum of the Squared Deviations (SSD) and the average of the squared deviations between the model values and the actual values. The Steady State Stopping Criterion is checked for each situation against an optimization trial with an excessive number of iterations (in this case, 200 iterations). To simplify the presentation of the plots, a subsystem of case designations is used which is described in detail in Appendix D.

The Weakest-Link-in-the-Chain analysis is validated against each optimization routine for a polynomial function (nonlinear in the dependent variable, but linear in the coefficients) and against a neural network (nonlinear in both response and coefficients). The results of this analysis are reported later on in this chapter.

Optimization of models based on Set A:

In Set A we are considering the data for the dependent variable ‘y’ to be noisy. This is generated by the *Rand()* function in MS EXCEL. The random numbers are Gaussian distributed and ranges from -0.5 to 0.5.

4.1.1 Model used: Third degree polynomial equation

$$y = a + bx + cx^2 + dx^3 \qquad (4.3)$$

In terms of the optimization, the parameters 'a' to'd' are the decision variables that need to be determined by the optimizer. The optimization algorithms are written in Visual Basic for Applications (VBA), and the data is displayed on MS EXCEL.

For the purposes of our study, we also run the optimizer for one trial using the Steady State Stopping Criterion, and then run the optimizer for the same initial guess without the Steady State Criteria. The limit of 200 iterations in each of the algorithms is used to terminate the search. The number 200 is used because it is about 4 to5 times the average number of iterations executed before the trial is terminated by the stopping criteria. The purpose of this endeavor is to determine the effectiveness of the stopping criterion in getting to the required minima for the trial.

In each case, the Weakest-Link-in-the-Chain analysis is used to determine the best number of trials that would give us the best 10% of the solutions with a confidence of 90%. On substituting the numbers in Equation 2.1.2, gives a result of 21.85434 runs. This number is rounded to 22 runs. It is also noted that the slightly higher number of runs would give us slightly better performance. The optimizer is run based on this number of required trials, and then the best answer from the set of 22 is selected and reported as the

global optimum for the given data. The threshold on $R_{statistic}$ in the Steady State Stopping Criterion is kept at 0.85 as per the discussion presented in Chapter II.

In order to test the accuracy of the above mentioned formula, a separate series of excessive trial runs are executed. Then the formula is used to pick a certain number of trials, which are then used to determine if the required best fraction of the results is reported with the required confidence. This series of tests is reported later on in the chapter.

Case 4.1.1.1 Optimization algorithm used: RRR's optimizer

The initial values of the four parameters in Equation (4.3) are randomly selected with each trial using the "*rnd*" function in Visual Basic for Applications, and the optimization was run for the required number of trials. The solution reported by the optimizer is given in Table 4.1 along with the SSD to give the reader an idea of the goodness of the fit.

Table 4.1: Final Optimization results for Case 4.1.1.1

Parameter	a	b	c	d	SSD
Value	0.414552	0.695335	-0.77846	0.282316	3.354435

The procedure is repeated for a single trial with the Steady State Stopping Criterion, and the same initial guess of 2 for each parameter is used to run the trial again without the

stopping criteria. The value of 2 has no special significance. The only thing that matters is that both the runs described start from the same point. The plot of the RMS error versus the filtered value of the error for both cases are displayed below.

Fig 4.1a RMS Error vs Filtered Error (Case A1)

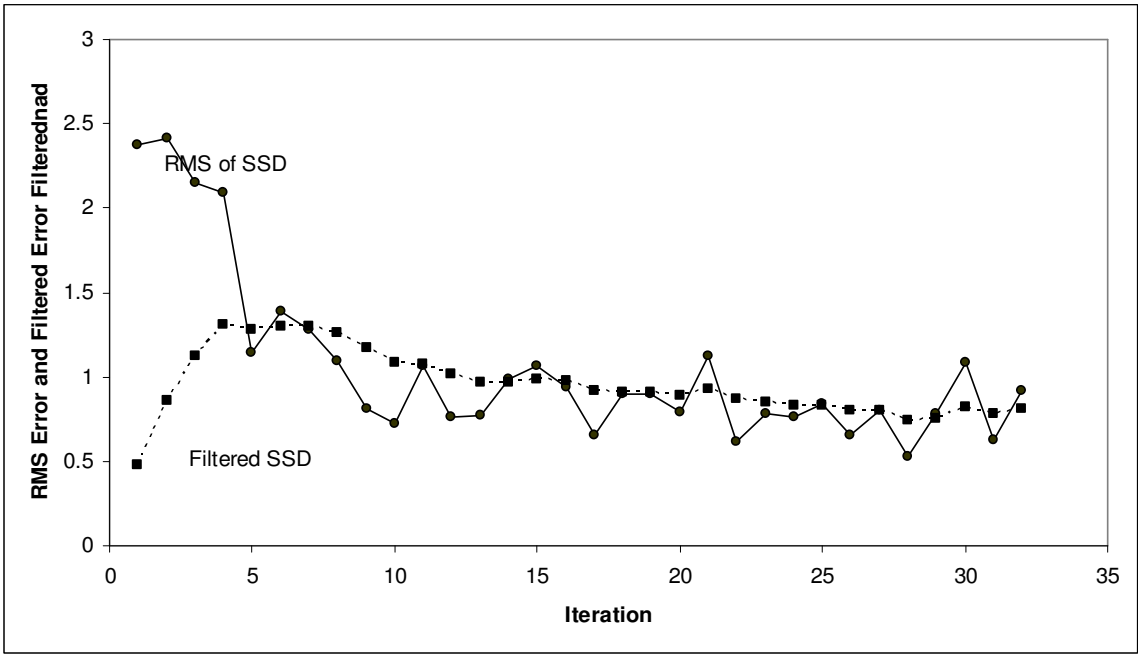
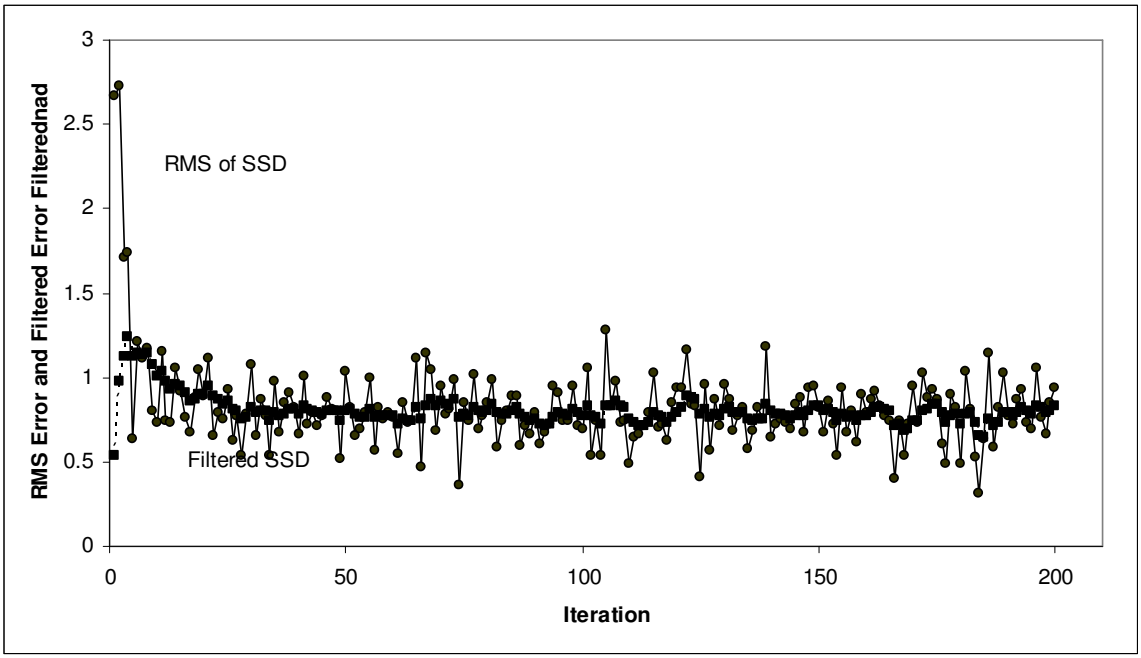


Fig 4.1b RMS Error vs Filtered Error (Case A2)



the final results of the two runs are shown below.

Table 4.2: Parameter values for Case (4.1.1)

Parameters	with Excessive iterations	Steady State Stopping Criterion
a	0.4144652	0.416332531
b	0.70228147	0.671817207
c	-0.778293	-0.776080926
d	0.26847124	0.335439205
SSD	3.35395357	3.364525408

From this we observe that there is a 0.314% improvement in the SSD when the Steady State Stopping Criterion is not used. It is also observed that the Stopping criteria terminated the trial at 32 iterations.

Case 4.1.1.2 Optimizer used: Hooke-Jeeves algorithm

The "*rnd*" function is used again to generate random starting guesses for the optimizer. The optimizer is run for the calculated number of trials and the best answer is reported.

Table 4.3: Final Optimization results for Case (4.1.1.2)

Parameter	a	b	c	d	SSD
Value	0.415	0.702	-0.778	0.268	3.353952

Again, a single trial is executed using an initial guess of 1 for each parameter and the results are compared with a similar trial with the same initial guess, but without the Steady State Stopping Criterion.

Fig 4.2a RMS Error vs Filtered Error (Case B1)

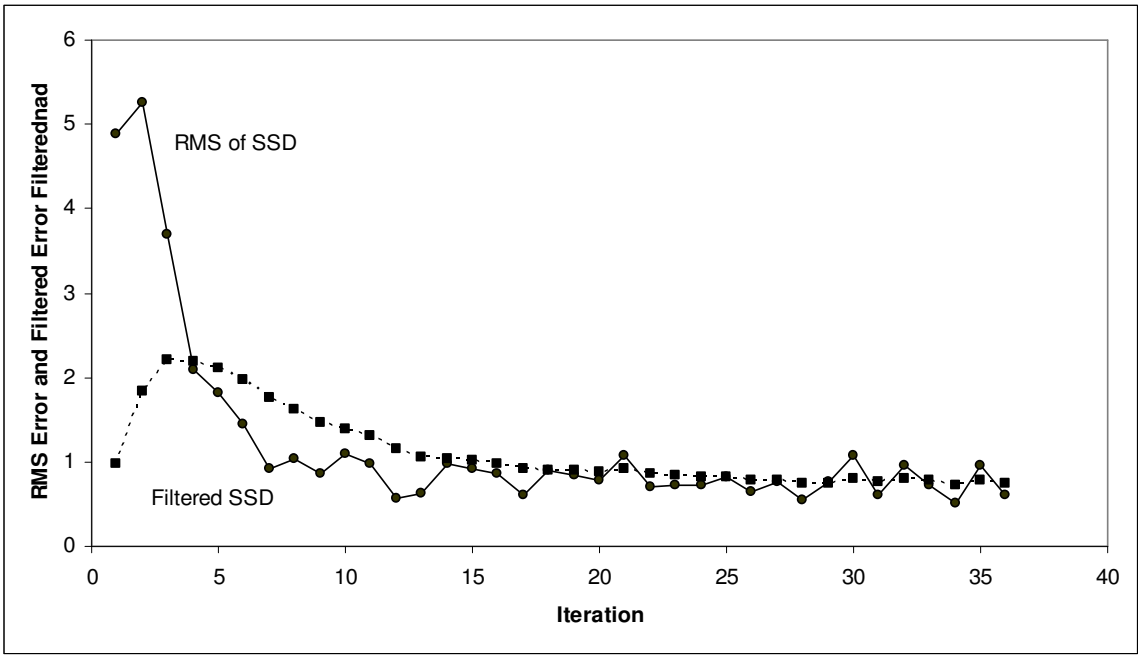
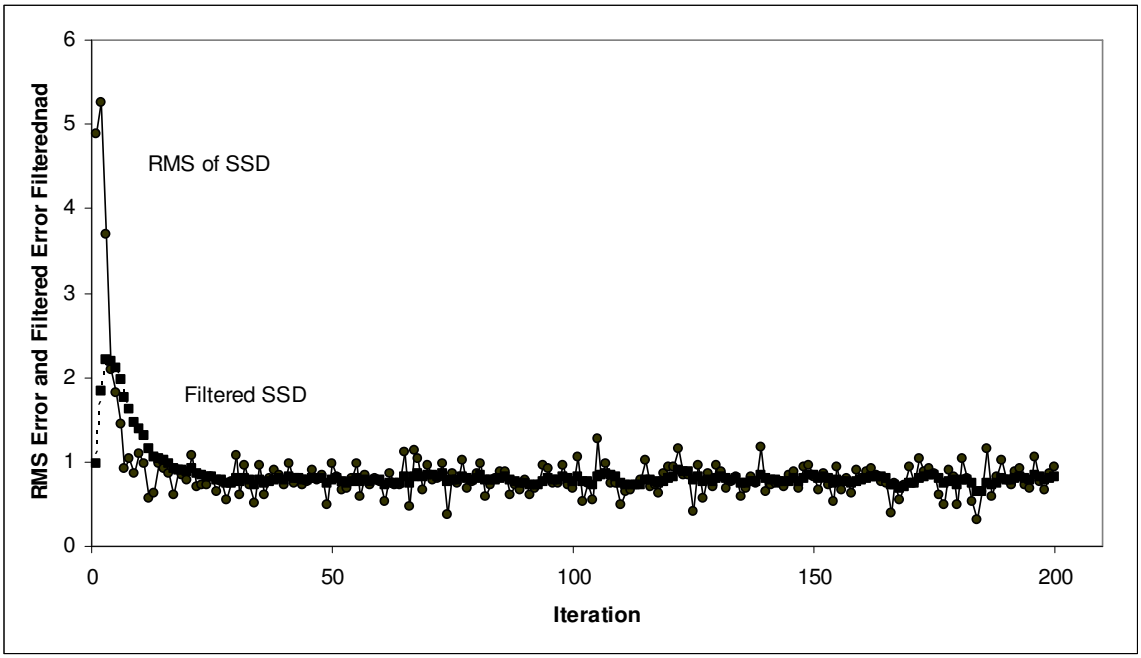


Fig 4.2b RMS Error vs Filtered Error (Case B2)



The final results are shown below.

From the results, it is observed that there is a 0.0005% difference between the SSD values, and the Steady State Stopping Criterion terminated the search in the thirty sixth iteration.

Table 4.4: Parameter values for Case 4.1.1.2

Parameters	with Excessive iterations	Steady State Stopping Criterion
A	0.41456223	0.41484375
B	0.70234375	0.703125
C	-0.7784996	-0.77890625
D	0.26844101	0.26640625
SSD	3.35395247	3.353969601

Another point is noted in the case of the Hooke Jeeves algorithm. There were cases where the steady state criteria was observed to have taken more time to terminate a trial compared to conventional criteria based on threshold values of the error. The stopping

criterion was also observed to terminate the trials before other threshold based stopping criteria. On an observation of 22 trials, the other conventional stopping criteria terminated five trails, and the rest were terminated by the steady state stopping criterion.

Case 4.1.1.3 Optimizer used: Broydon-Fletcher-Goldfarb-Shanno (BFGS) algorithm

The same procedure as before is repeated, where the parameters are randomly selected before each trial and the best result is reported as the global minima.

Table 4.5: Final Optimization results for Case (4.1.1.3)

Parameter	a	b	c	d	SSD
Value	0.414514	0.702425	-0.77839	0.268191	3.353952533

The Steady State Stopping Criterion is evaluated by running a trail with starting values of 2 for each parameter and then running the same trial without the criteria. The results are displayed below:

Fig 4.3a RMS Error vs Filtered Error (Case C1)

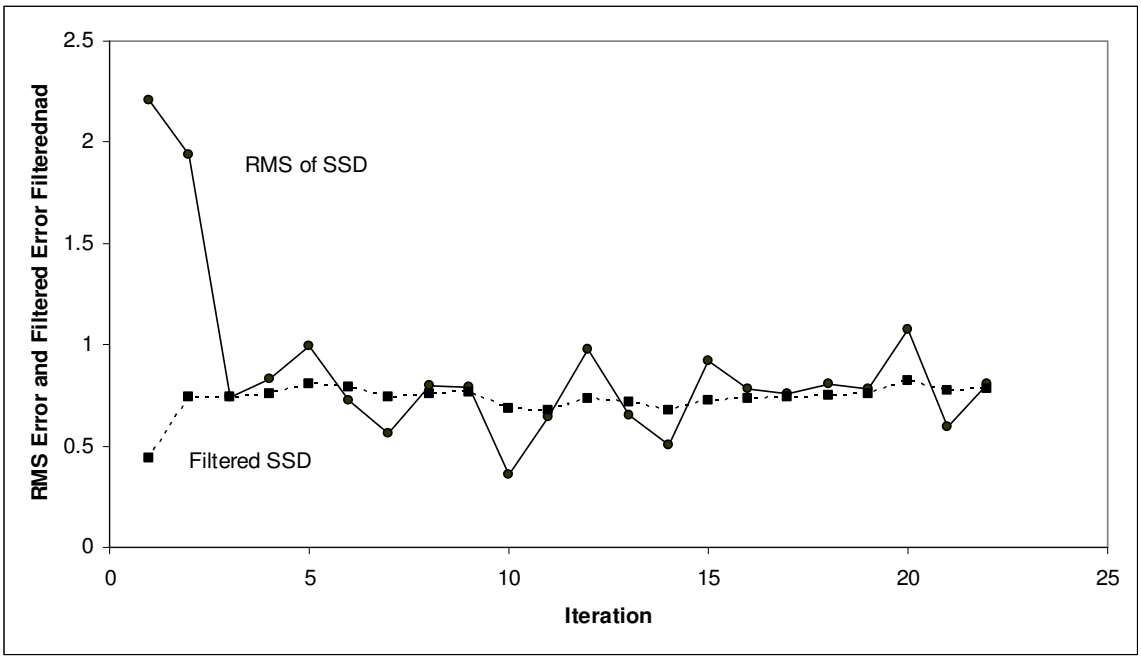


Fig 4.3b RMS Error vs Filtered Error (Case C2)

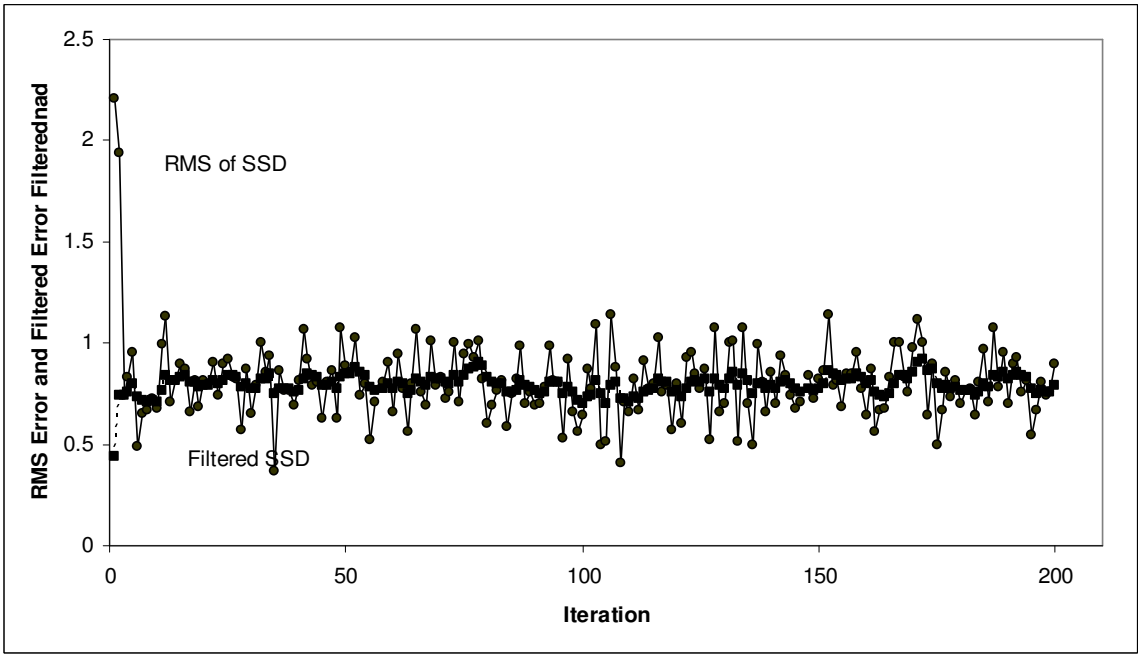


Table 4.6: Parameter values for Case (4.1.1.3)

Parameters	with Excessive iterations	Steady State Stopping Criterion
A	0.41446524	0.41443425
B	0.70228159	0.702190102
C	-0.7782931	-0.778233507
D	0.26847101	0.268648939
SSD	3.35395357	3.353954236

It is observed that there is almost no difference between the SSD value reported in the two cases, and the stopping criteria terminated the trial at 28 iterations, which for the case of the BFGS optimizer is very advantageous if we consider the computation time required.

4.1.2 Model used: neural network

A bipolar sigmoidal neural network is used to model the process. The neural network is nonlinear in terms of the parameters and in terms of the variables.

Case 4.1.2.1 Optimizer used: RRR's Optimizer

The seven parameters of the neural network are randomly selected using the "*rnd*" function in VBA at the start of each new trial. The optimizer is run for the calculated number of trials and the best solution is reported. The SSD is also reported since it gives an idea of how close the neural network is to modeling the actual process.

Table 4.7: Final Optimization results for Case (4.1.2.1)

Bias	b-hidden	x-hidden	hidden-out	SSD
-1.669559	2.894438	-9.75150	-0.2449644	2.684418
	-0.3494423	2.06260	0.51463575	

The Steady State Stopping Criterion is then tested by running one trial of the optimizer and comparing the results using the same initial guess and making the optimizer run for a large number of iterations (in this case 200).

Fig 4.4a RMS Error vs Filtered Error (Case D1)

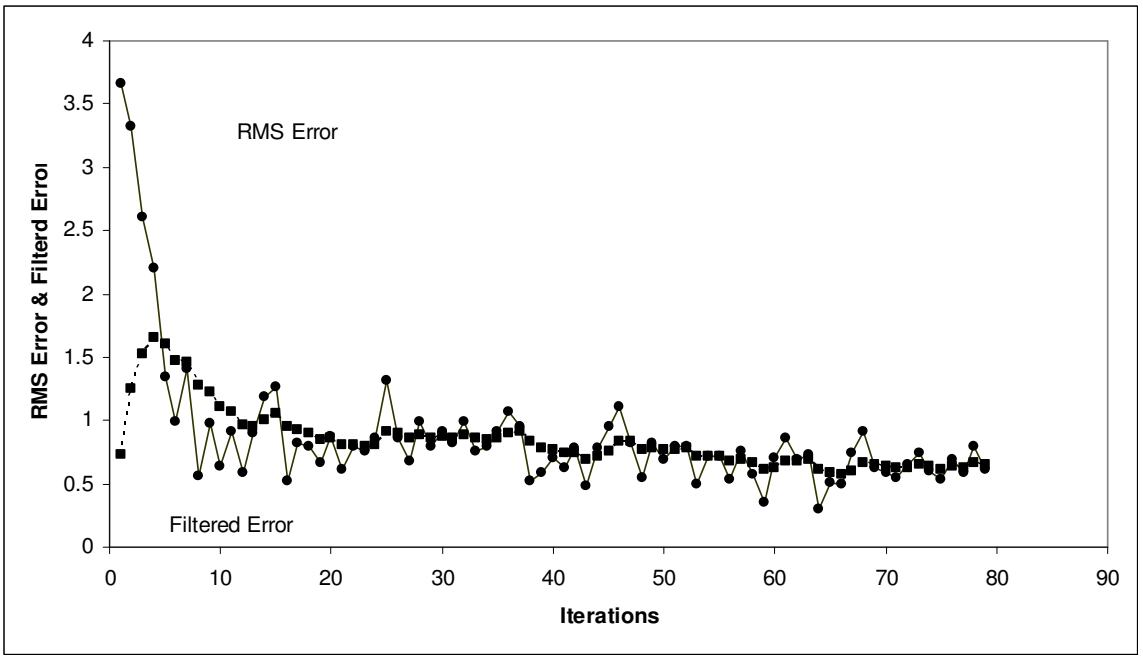
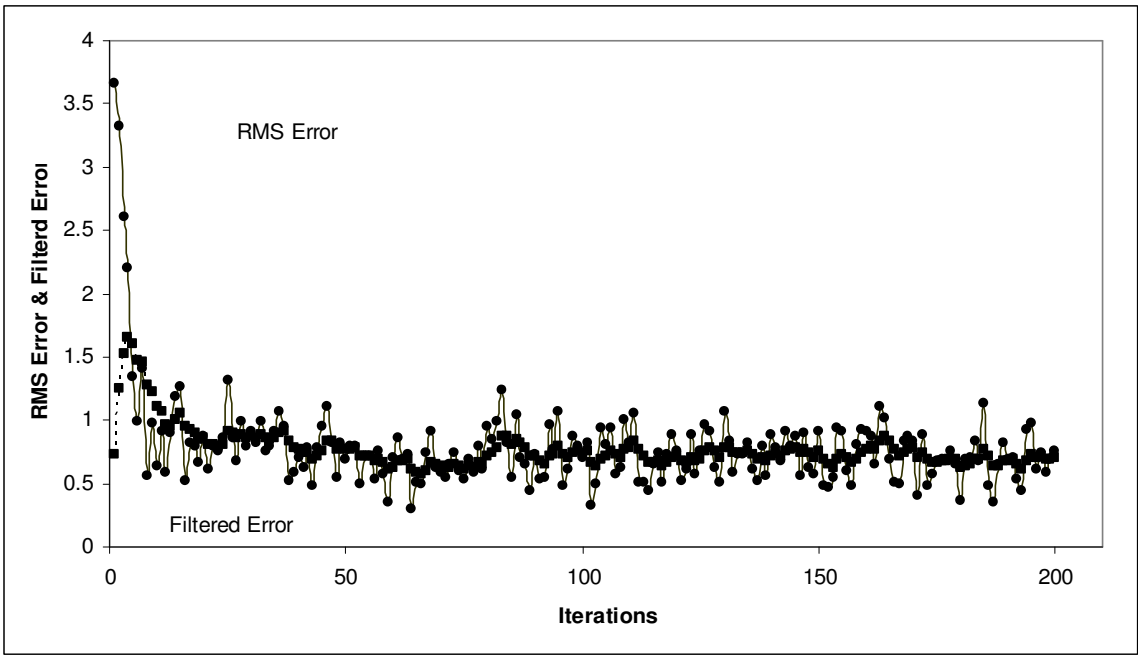


Fig 4.4b RMS Error vs Filtered Error (Case D2)



It is observed that the excessive iterations leads to an improvement of 0.02% from the solution presented by the Steady State Stopping Criterion, and the latter terminated the trial at 30 iterations which indicates that a lot of computation time is saved by the stopping criteria.

Table 4.8: Parameter values for Case (4.1.2.1)

Parameters	with Excessve iterations	Steady State Stopping Criterion
Bias	0.936509028	0.939408105
b-hidden	2.513650257	2.513650257
	0.058786107	0.058786107
x-hidden	4.762602597	4.765501674
	1.388032997	1.388032997
hidden-out	0.486840389	0.486983553
	0.354746384	0.354889548
SSD	2.575424037	2.574843348

Case 4.1.2.2 Optimizer used: Hooke Jeeves algorithm

As before, the initial values of the seven parameters are randomly selected before each trial and the optimizer is run for the requisite number of trials before the best answer is selected to be reported as the global minima.

Table 4.9: Final Optimization results for Case (4.1.2.2)

Bias	b-hidden	x-hidden	hidden-out	SSD
1.225656	1.551861	3.634135	0.654645	2.59205
	-0.24638	0.837713	0.411149	

The Steady State Stopping Criterion is then tested by executing one trial of the optimizer with the stopping criteria and repeating the trial with the same initial values, but tiheout the stopping criteria and letting the optimizer run for the whole 200 iteration limit.

Fig 4.5a RMS Error vs Filtered Error (Case E1)

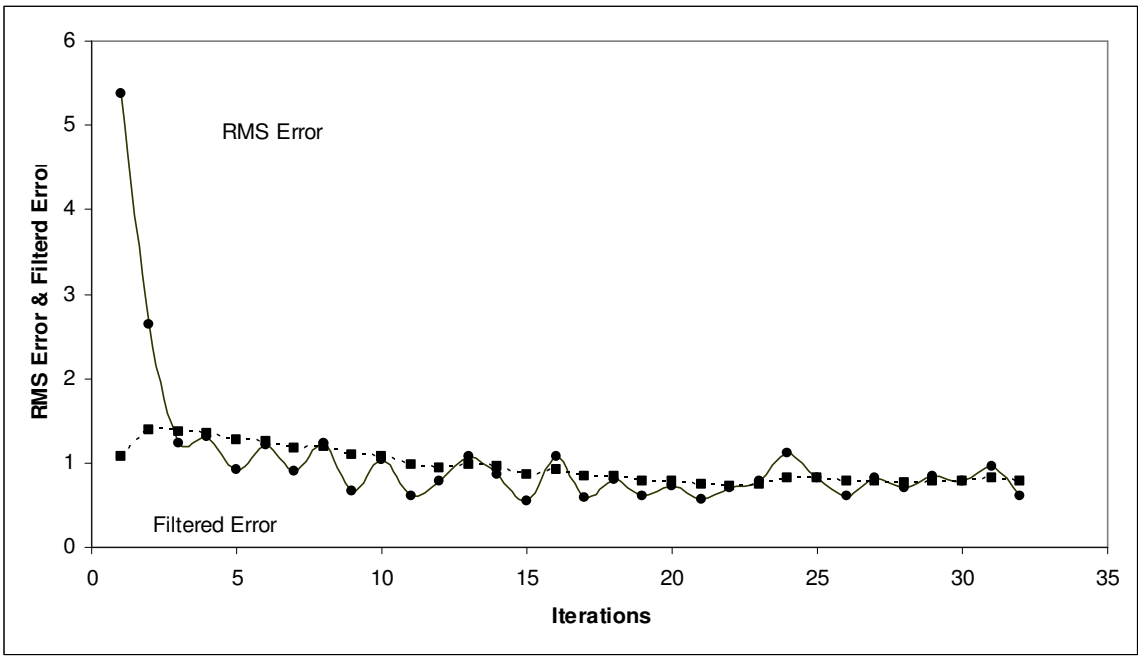


Fig 4.5b RMS Error vs Filtered Error (Case E2)

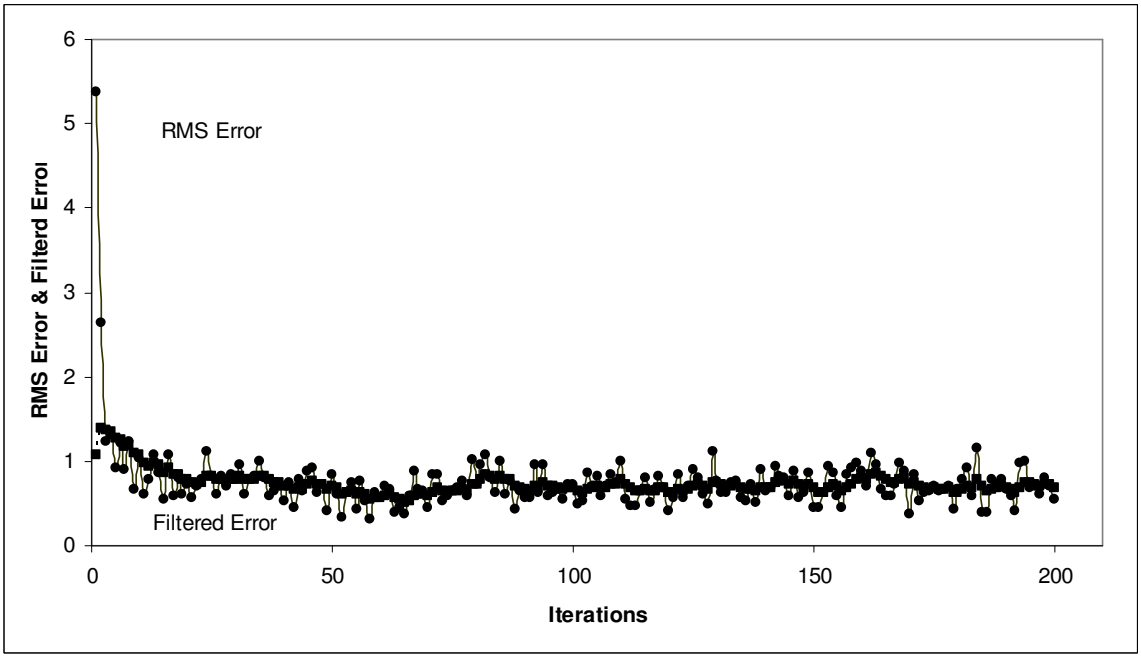


Table 4.10: Parameter values for Case (4.1.2.2)

Parameters	with Excessive iterations	Steady State Stopping Criterion
Bias	0.772053957	1.033696079
b-hidden	-6.109157372	-0.481925535
	2.412784433	-0.2510396
x-hidden	-4.607173777	-2.191750145
	1.679784012	-4.218929434
hidden-out	3.590732861	-1.392207956
	4.326978636	0.542894363
SSD	2.70101234	3.04521071

It is observed that there is a 12% improvement in the solution in this case. It can be attributed to the occurrence of Type-II errors.

Case 4.1.2.3 Optimizer used: BFGS algorithm

As has been done before, the initial values are selected at the start of each new trial using the "*rnd*" function in VBA. The number of trials is determined based on the best fraction and confidence desired by the user and the best result among the trials is reported as the final answer to the requirement.

Table 4.11: Final Optimization results for Case (4.1.2.3)

Bias	b-hidden	x-hidden	hidden-out	SSD
-1.181747	39.73161	70.82894	0.1546973	2.463505
	-1.237346	3.087534	0.7797951	

The stopping criteria is then tested by making one trial without using the Steady State Stopping Criterion and another trial using the same initial guess values and using the stopping criteria to terminate the run when steady state is attained.

Fig 4.6a RMS Error vs Filtered Error (Case F1)

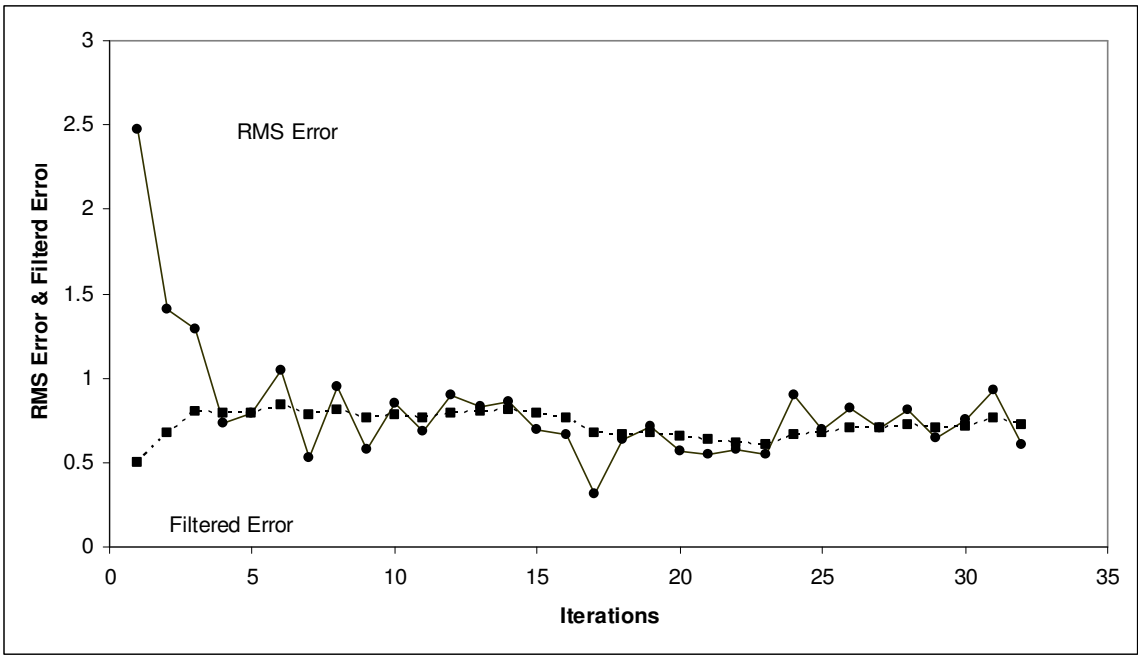


Fig 4.6b RMS Error vs Filtered Error (Case F2)

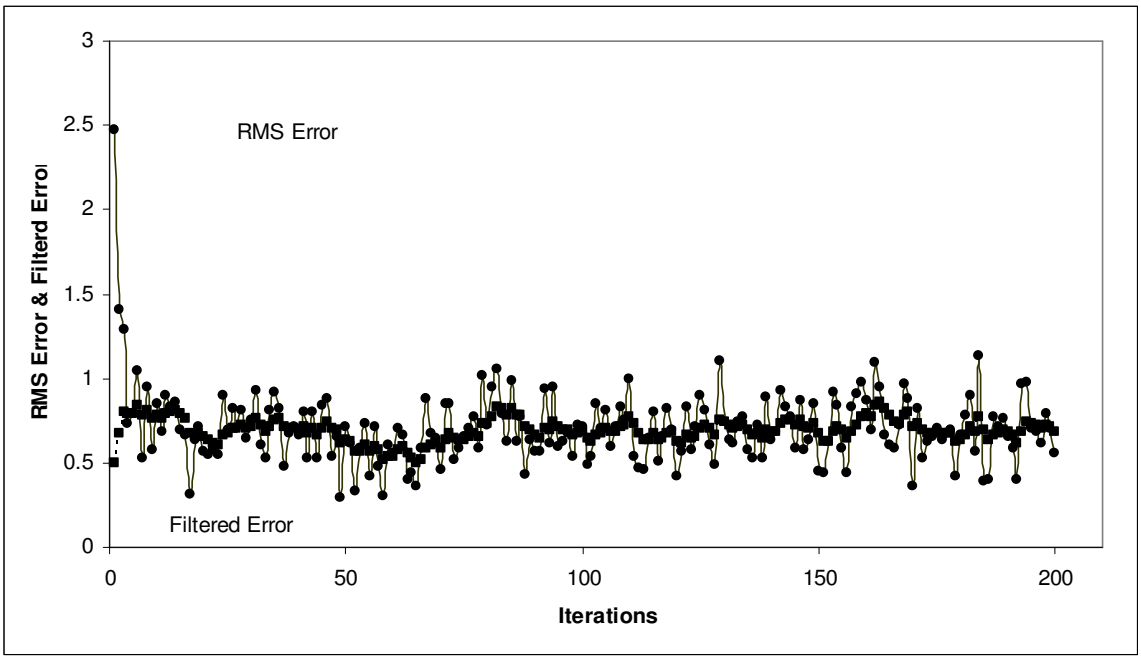


Table 4.12: Parameter values for Case (4.1.2.3)

Parameters	with Excessve iterations	Steady State Stopping Criterion
Bias	-1.976906984	0.32308939
b-hidden	-1.53899143	-4.161999479
	0.071812319	-6.689022244
x-hidden	6.132983145	-2.037950908
	-1.353032068	1.647607143
hidden-out	0.430599932	-1.216941135
	-0.422728368	0.566541194
SSD	2.544638865	2.744895701

It is observed that there is a 7% improvement in the SSD and this can be attributed to Type-II errors.

Another factor that was observed in the BFGS algorithm was that there was a large amount of computation time involved in most trials. This is because each iteration involves derivative calculations, which in the case of empirical modeling means the calculation of the SSD between the experimental data and the model values based on each change in a parameter. This also indicated that it might be inconvenient to use indirect optimization methods. Also, since numerical differentiation techniques are being used, it wouldn't be prudent to use a higher degree differentiation technique since it would significantly increase the computation time involved.

Optimization of Different models based on Set B:

In Set B, we have errors associated with both the dependent and independent variables. It is a more realistic representation of a process since we have measurement disturbances to take into account too. As in the case of Set A, we set the same parameters for the Weakest-Link-in-the-Chain formula, i.e. a 90% confidence that one of the best 10% of the answers will be reported each time. In this series, the threshold value of the $R_{statistic}$ in the Steady State Stopping Criterion is kept at 1. The intention is to see if there are any problems that might arise which might not have been noticed in Set A.

4.1.3 Model used: Third degree polynomial equation

Case 4.1.3.1 Optimization algorithm used: RRR's optimizer

The initial values of the four parameters in Equation (4.3) are randomly selected with each trial using the "*rnd*" function in Visual Basic for Applications, and the optimization was run for the required number of trials. The solution reported by the optimizer is given in Table 4.13 along with the SSD to give the reader an idea of the goodness of the fit.

Table 4.13: Final Optimization results for Case (4.1.3.1)

Parameter	a	b	C	D	SSD
Value	0.264408	0.3377009	-2.206826	-0.3796403	1.8214876

The procedure is repeated for a single trial with the Steady State Stopping Criterion, and the same initial guess of 2 for each parameter is used to run the trial again without the stopping criteria. Again the value of 2 has no special significance.

Fig 4.7a RMS Error vs Filtered Error (Case G1)

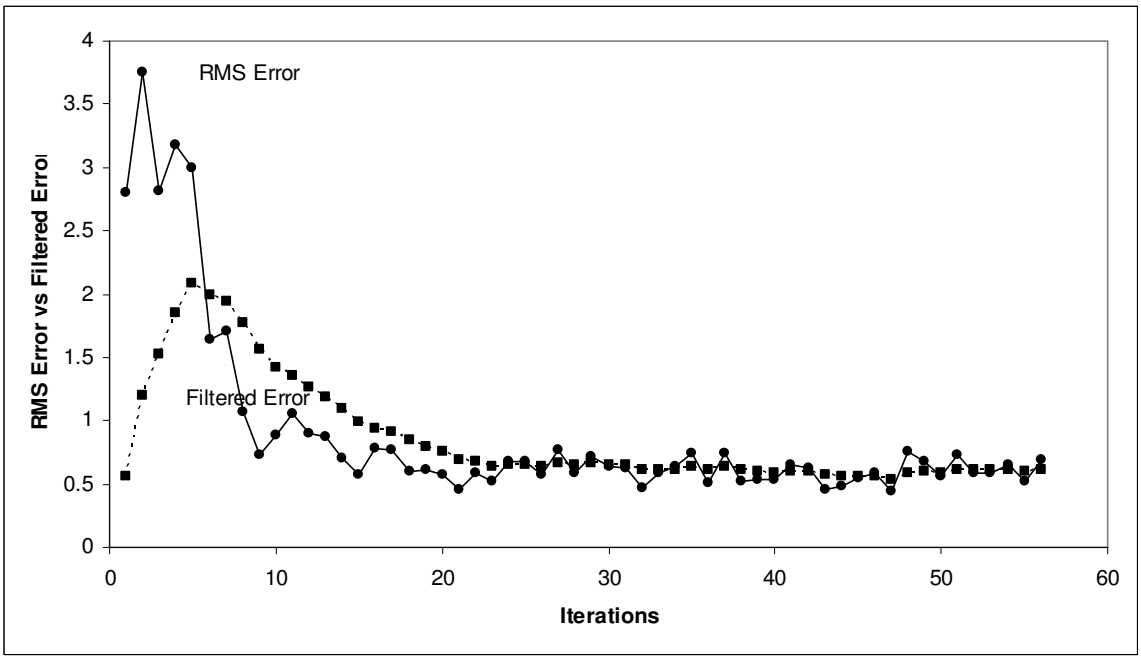
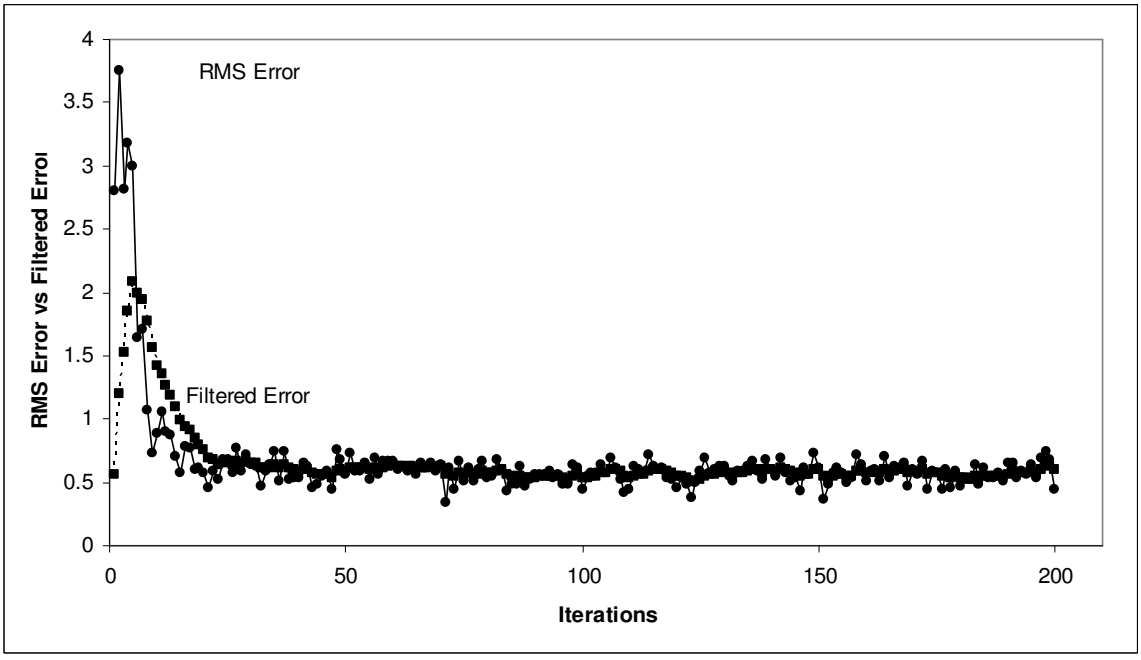


Fig 4.7b RMS Error vs Filtered Error (Case G2)



The final results of the two runs are shown below.

Table 4.14: Parameter values for Case (4.1.3.1)

Parameters	with Excessive iterations	Steady State Stopping Criterion
a	0.26405963	0.2649534
b	0.34507112	0.319404045
c	-2.205113	-2.211893244
d	-0.3938873	-0.332135875
SSD	1.82145175	1.821900973

From this we observe that there is a 0.0001% improvement in the SSD when we don't use the Steady State Stopping Criterion. It is also observed that the Stopping criteria terminated the trial at 57 iterations.

Case 4.1.3.2 Optimizer used: Hooke-Jeeves' algorithm

The "*rnd*" function is used again to generate random starting guesses for the optimizer. The optimizer is run for the calculated number of trials and the best answer is reported.

Table 4.15: Final Optimization results for Case (4.1.3.2)

Parameter	a	b	c	d	SSD
Value	0.26407	0.3452001	-2.20517	-0.394158	1.8214517

Again, a single trial is executed using an initial guess of 1 for each parameter and the results are compared with a similar trial with the same initial guess, but without the Steady State Stopping Criterion.

Fig 4.8a RMS Error vs Filtered Error (Case H1)

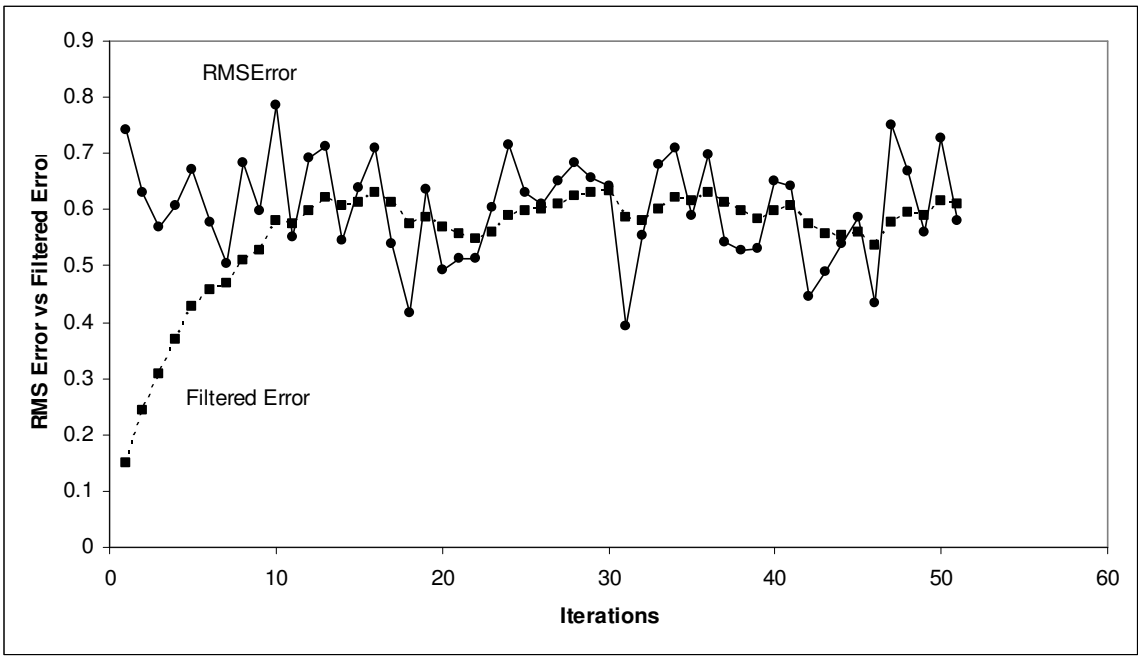
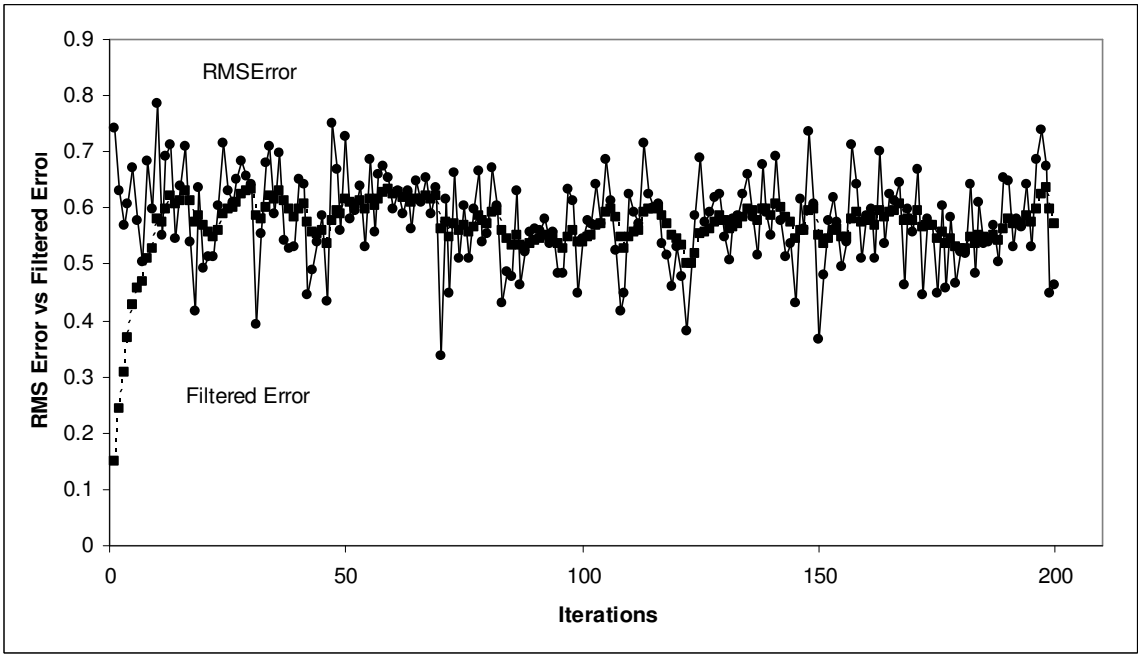


Fig 4.8b RMS Error vs Filtered Error (Case H2)



The final results are shown below.

Table 4.16: parameter values for Case (4.1.3.2)

Parameters	with Excessive iterations	Steady State Stopping Criterion
A	0.26406336	0.264088535
B	0.34533124	0.345412111
C	-2.2051062	-2.205220604
D	-0.3945288	-0.394643259
SSD	1.82145177	1.821451806

From the results, it is observed that there is a 0.00005% difference between the SSD values, and the Steady State Stopping Criterion terminated the search in the fifty second iteration.

The point noted in the case of the Hooke Jeeves' algorithm execution in Set A is noted again in this case. There were cases where the initial guess was inappropriate, and the final SSD reported at the end of the 100 iteration limit was worse than the genera minima reported.

case 4.1.3.3 Optimizer used: Broydon-Fletcher-Goldfarb-Shanno (BFGS) algorithm

The same procedure as before is repeated, where the parameters are randomly selected before each trial and the best result is reported as the global minima.

Table 4.17: Final Optimization results for Case (4.1.3.3)

Parameter	a	b	c	d	SSD
Value	-3.203664	0.4466594	10.568484	-4.1892569	1.8214517

The Steady State Stopping Criterion is evaluated by running a trail with starting values of 2 for each parameter and then running the same trial without the criteria. The results are displayed below:

Fig 4.9a RMS Error vs Filtered Error (Case I1)

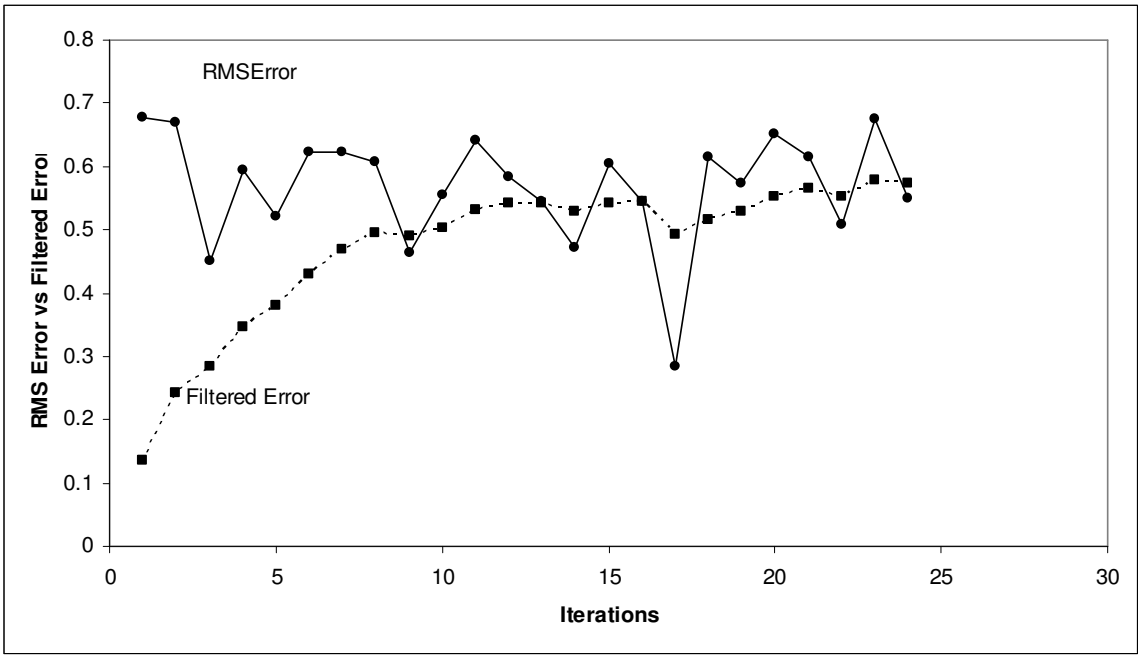


Fig 4.9b RMS Error vs Filtered Error (Case I2)

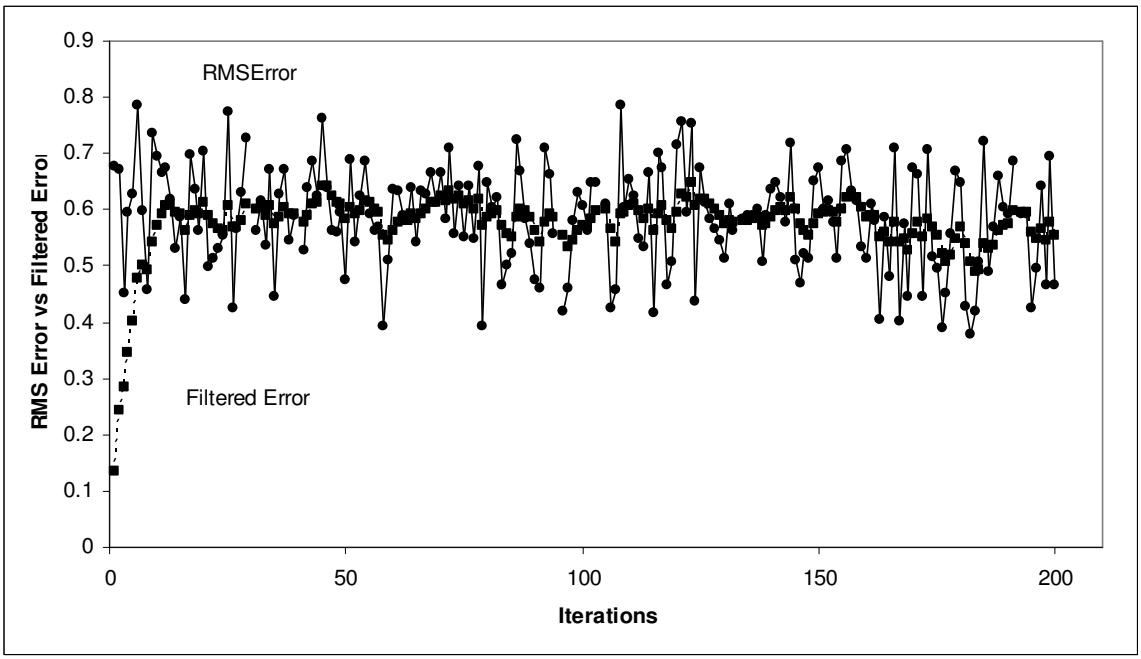


Table 4.18: parameter values for Case (4.1.3.3)

Parameters	with Excessive iterations	Steady State Stopping Criterion
A	0.26396833	0.263968352
B	0.34498261	0.344982613
C	-2.2049263	-2.204926401
D	-0.3937965	-0.393796516
SSD	1.82145183	1.821451823

It is observed that there is almost no difference between the SSD value reported in the two cases, and the stopping criteria terminated the trial at 25 iterations, which for the case of the BFGS optimizer is very advantageous if we consider the computation time required.

4.1.4 Model used: neural network

Case 4.1.4.1 Optimizer used: RRR's Optimizer

The seven parameters of the neural network are randomly selected using the "rnd" function in VBA at the start of each new trial. The optimizer is run for the calculated number of trials and the best solution is reported. The SSD is also reported since it gives us an idea of how close the neural network is to modeling the actual process.

Table 4.19: Final Optimization results for Case (4.1.4.1)

Bias	b-hidden	x-hidden	hidden-out	SSD
0.66989	-2.33702	-5.70278	-1.6659	0.835982
	-0.55894	-1.15353	2.7867	

The Steady State Stopping Criterion is then tested by running one trial of the optimizer and comparing the results using the same initial guess and making the optimizer run for a large number of iterations (in this case 200).

Fig 4.10a RMS Error vs Filtered Error (Case J1)

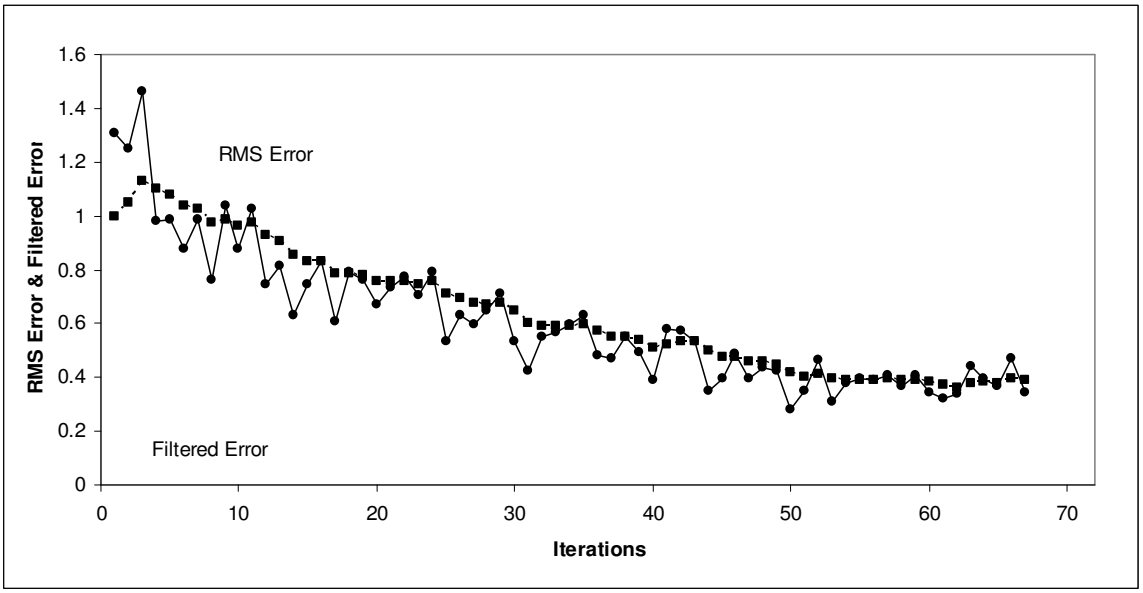


Fig 4.10b RMS Error vs Filtered Error (Case J2)

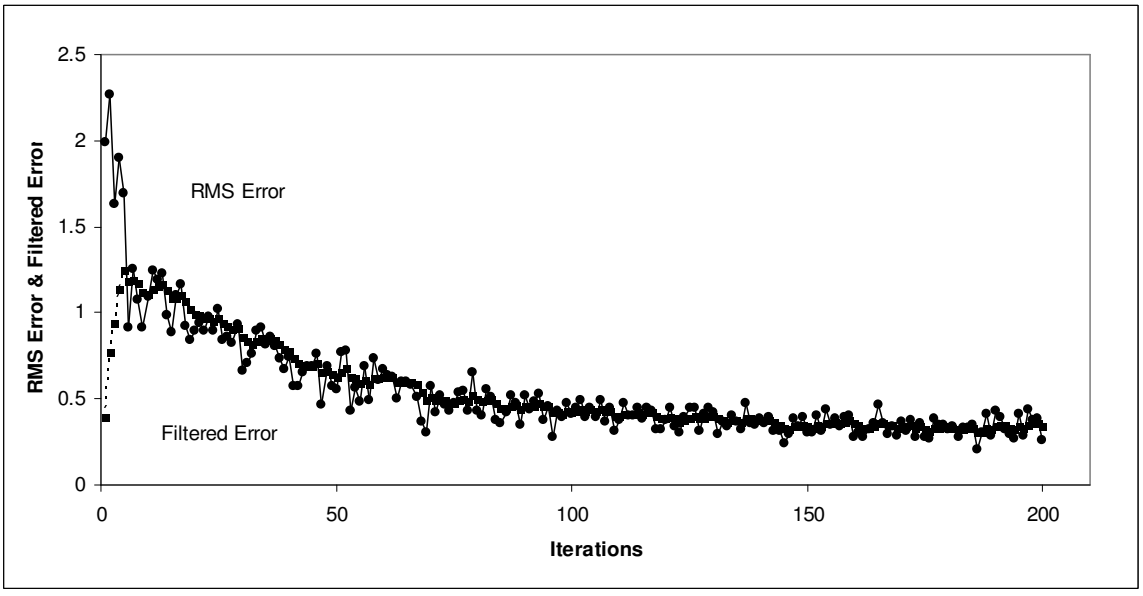


Table 4.20: Parameter values for Case (4.1.4.1)

Parameters	with Excessve iterations	Steady State Stopping Criterion
Bias	0.42395801	0.579606397
b-hidden	3.54516021	2.881790712
	-0.6572138	0.702295161
x-hidden	5.54636768	6.077587936
	-0.8814934	1.239091187
hidden-out	1.80828615	1.710634785
	3.61551556	-2.678422714
SSD	0.59914322	0.869499883

It is observed that the excessive iterations leads to an significant improvement from the solution presented by the Steady State Stopping Criterion. From Figures 4.10a and 4.10b, we note that the steady state identifier doesn't really track the gradual decrease in the errors. This is a typical example of the Type II error, where the null hypethesis is accepted even if it's not true, i.e. the data window being observed by the identifier leads the latter to infer the attainment of steady state even if it has not been attained. One way to reduce Type-II error would be to sample more data for the purposes of steady state identification. Another factor coming into play is the threshold on the value of $R_{statictic}$ which identifies steady state. The value of 1 might be replaced by a lower value (say 0.85).

Case 4.1.4.2 Optimizer used: Hooke Jeeves algorithm

As before, the initial values of the 7 parameters are randomly selected before each trial and the optimizer is run for the requisite number of trials before the best answer is selected to be reported as the global minima.

Table 4.21: Final Optimization for Case (4.1.4.2)

Bias	b-hidden	x-hidden	hidden-out	SSD
0.61757	2.08481	-3.734539	3.38889	0.4359
	-0.69004	1.172703	5.41916	

The Steady State Stopping Criterion is then tested by executing one trial of the optimizer with the stopping criteria and repeating the trial with the same initial values, but tiheout the stopping criteria and letting the optimizer run for the whole 200 iteration limit.

Fig 4.11a RMS Error vs Filtered Error (Case K1)

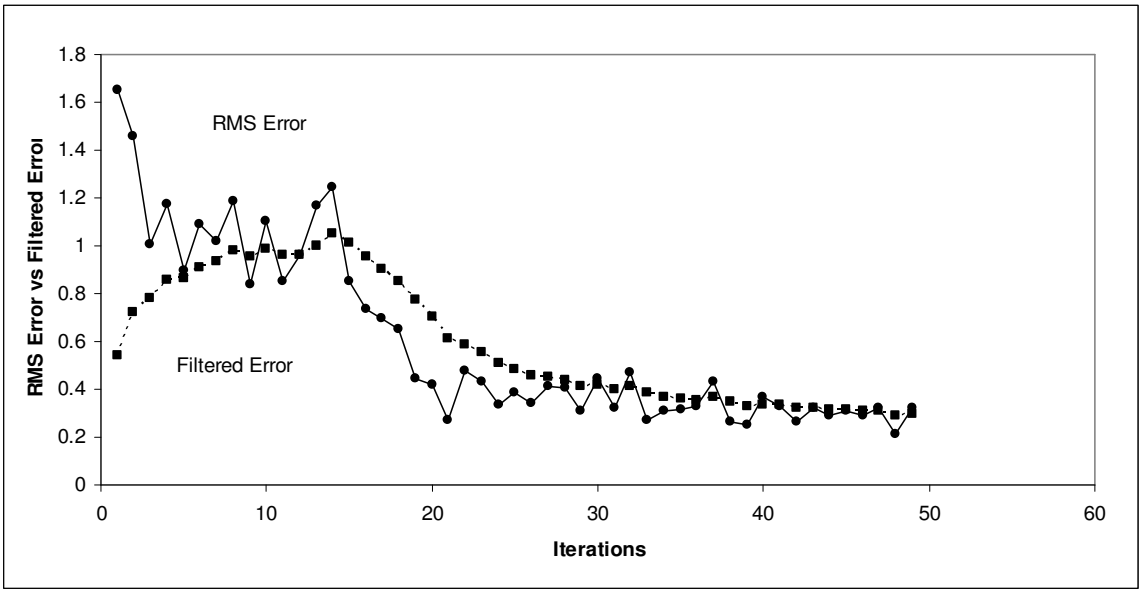


Fig 4.11b RMS Error vs Filtered Error (Case K2)

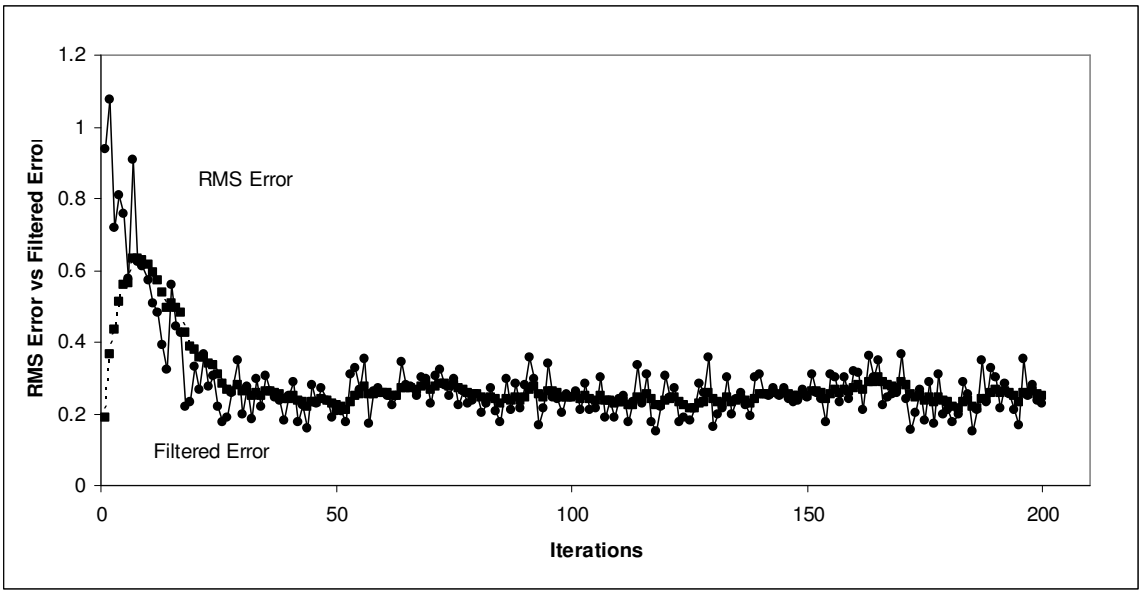


Table 4.22: Parameter values for Case (4.1.4.2)

Parameters	with Excessve iterations	Steady State Stopping Criterion
Bias	0.53955545	-0.425464964
b-hidden	0.57256665	-2.420506859
	2.32747602	1.066876841
x-hidden	-0.8233908	3.634687185
	-3.5987146	-1.47389946
hidden-out	-6.0723837	3.808899975
	2.95178108	5.161466551
SSD	0.34193133	0.435365259

It is observed that there is a 20% improvement in the solution in this case. Figure 4.11a does indicate another case of Type-II error, even though the difference in the solutions is not as significant as in the previous case.

Case 4.1.4.3 Optimizer used: BFGS algorithm

As it has been done before, the initial values are selected at the start of each new trial using the "*rnd*" function in VBA. The number of trials is determined based on the best fraction and confidence desired by the user and the best result among the trials is reported as the final answer to the requirement.

Table 4.23: Final Optimization results using for Case (4.1.4.3)

Bias	b-hidden	x-hidden	hidden-out	SSD
-0.29778	-1.82963	-1.499854	-7.07405	0.315093
	3.553234	3.019008	-5.38718	

The stopping criteria is then tested by making one trial without using the Steady State Stopping Criterion and another trial using the same initial guess values and using the stopping criteria to terminate the run when steady state is attained.

Fig 4.12a RMS Error vs Filtered Error (Case L1)

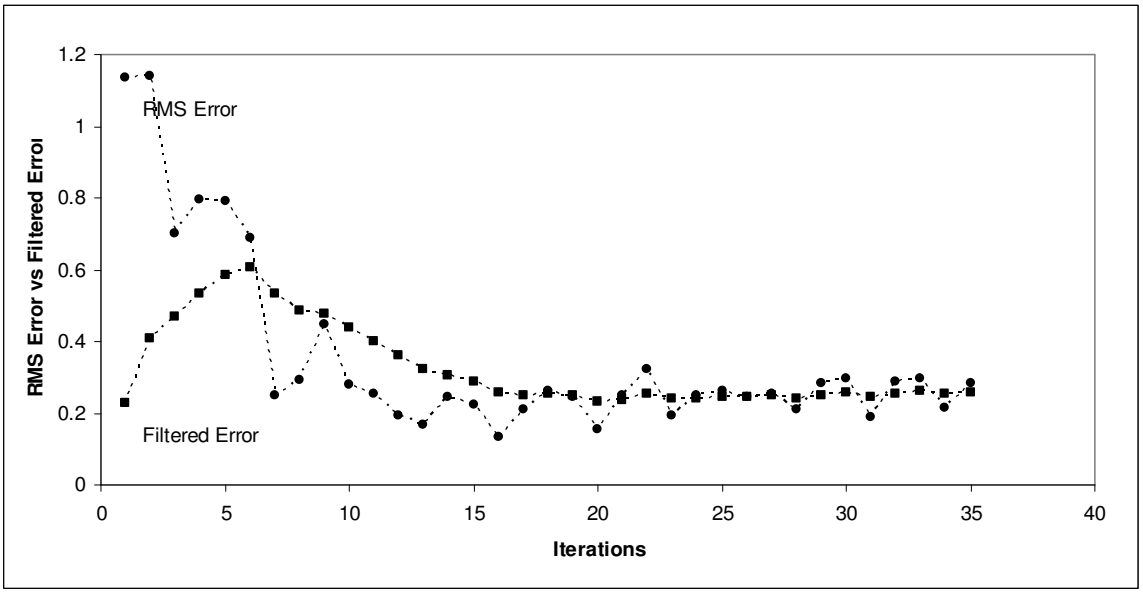


Fig 4.12b RMS Error vs Filtered Error (Case L2)

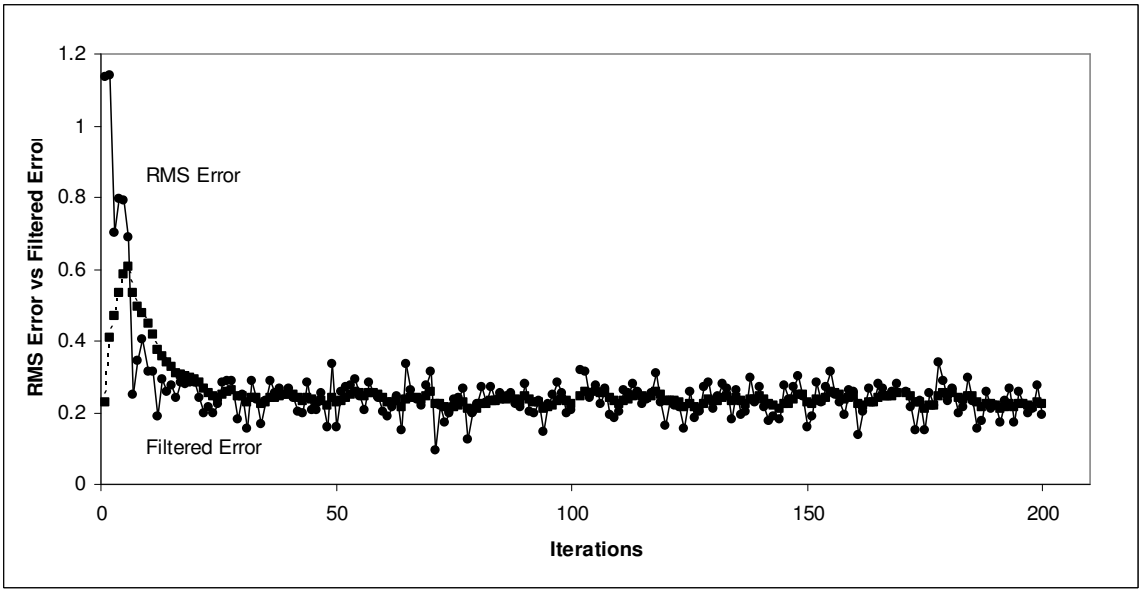


Table 4.24: Parameter values for Case (4.1.4.3)

Parameters	with Excessve iterations	Steady State Stopping Criterion
Bias	0.26327861	0.308302404
b-hidden	2.18081727	1.617539959
	3.79800038	3.595572715
x-hidden	-1.5764121	-1.366225855
	-2.8347884	-3.175522809
hidden-out	-8.2699408	-6.409117966
	6.57475695	4.566487341
SSD	0.3102657	0.316846743

It is observed that there is a 7% improvement in the SSD, though it is not significant considering the actual numbers generated.

4.2. Results from Experimental Data

Two-phase flow is the simultaneous flow of a gas and a liquid in a pipe or tube. This is a very commonly observed phenomenon in chemical engineering unit operations, such as distillation columns, evaporators, reactors, condensers etc. in this study, we consider the two-phase flow of water and air in a vertical pipe. The fluid flow rates are measured using rotameters in coordination with orifice meters. And a control system is used to control the flow in the system. The CAMILE software is used to monitor and operate the control system. Pressure transducers measure the pressure at the top and bottom of the vertical column. All the data is assimilated by CAMILE, and reported in text files. The experimental data used are shown in Appendix D.

There are various methods used to model the pressure drop in two-phase flow. In this study, the Lockhart-Martinelli correlation is used to determine the same. A sample calculation is shown in Appendix B

The Lockhart-Martinelli Correlation constant, C is readily available from the literature. The values change depending on the flow characteristics. And are shown in table

Table 4.25: Flow Patterns of Fluid based on Reynolds number

Flow Pattern	Reynolds number
Laminar	Re<2000
Turbulent	3000<Re<50000

Table 4.26: Lockhart-Martinelli correlation constants for different vapor-liquid flow patterns.

Liquid	Vapor	C
Laminar	Laminar	5
Turbulent	Laminar	10
Laminar	Turbulent	12
Turbulent	Turbulent	20

The value of C is evidently dependent on the flow patterns of both the liquid and the vapor. To effectively correlate this in the calculation of the correlation constant by the optimizer, the following model is used:

$$C = a\,\mathrm{Re}^{b}\,\mathrm{Re}^{c} \qquad (4.2)$$

The flow data obtained (presented in Appendix B) is classified into four groups based on the flow patterns of the liquid and the vapor. The objective of the present set of cases is to make the Lockhart-Martinelli model best predict the experimentally measured pressure drop for each of the four laminar-turbulent cases. The three coefficients, a, b, and c, are the DVs in each optimization. The effectiveness of the stopping criteria is tested by running the optimizer once and repeating the trial with the same initial guess, but without the stopping criteria. The maximum limit of 200 iterations is assumed to be adequate to ensure steady state. The goodness of the model itself is checked by plotting the experimental pressure drop values against the pressure drop values predicted by the model. The RRR's Optimizer is used in the presentation of the cases. The classification and the results obtained in each case are discussed below.

Case 4.2.1 Liquid Flow – Laminar

Vapor Flow – Laminar

The values of a, b, c and the SSD for Laminar-Laminar flow is given in table 4.?

Table 4.27: Final Optimization results for Laminar-Laminar Flow

Parameter	a	b	C	SSD
Value	-1.59013	-1.67572	1.685488	0.28827

Fig 13: Experimental Pressure Drop vs Model Pressure Drop for Laminar-Laminar Flow

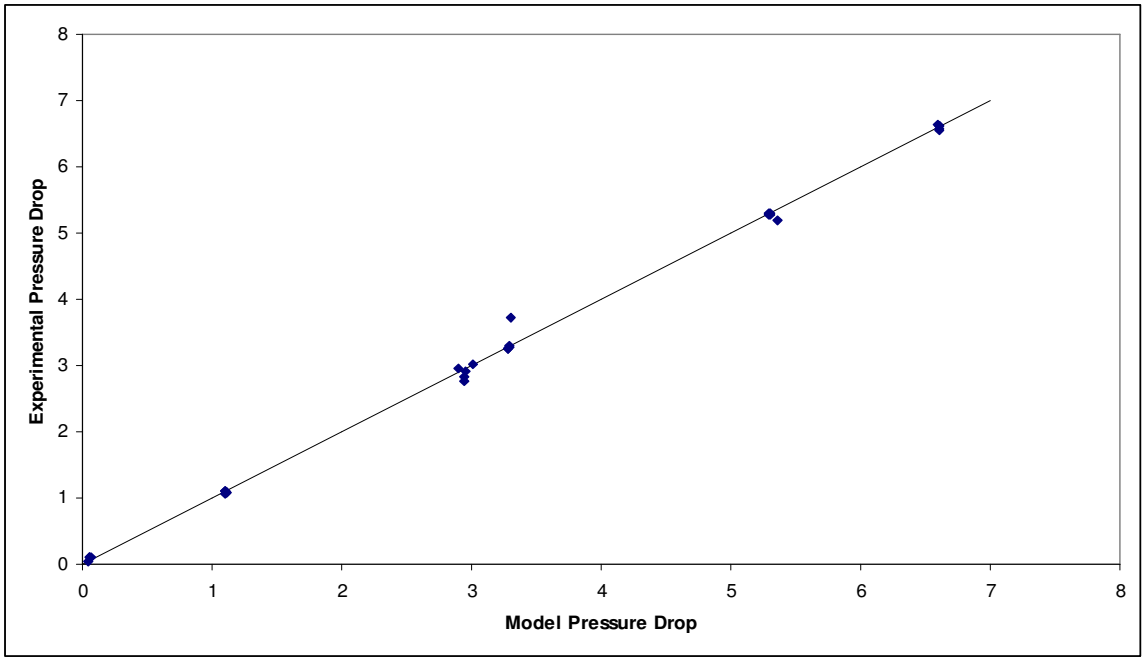


Fig 4.14a: RMS Error vs Filtered Error for Laminar-Laminar Flow with Steady State Stopping Criterion

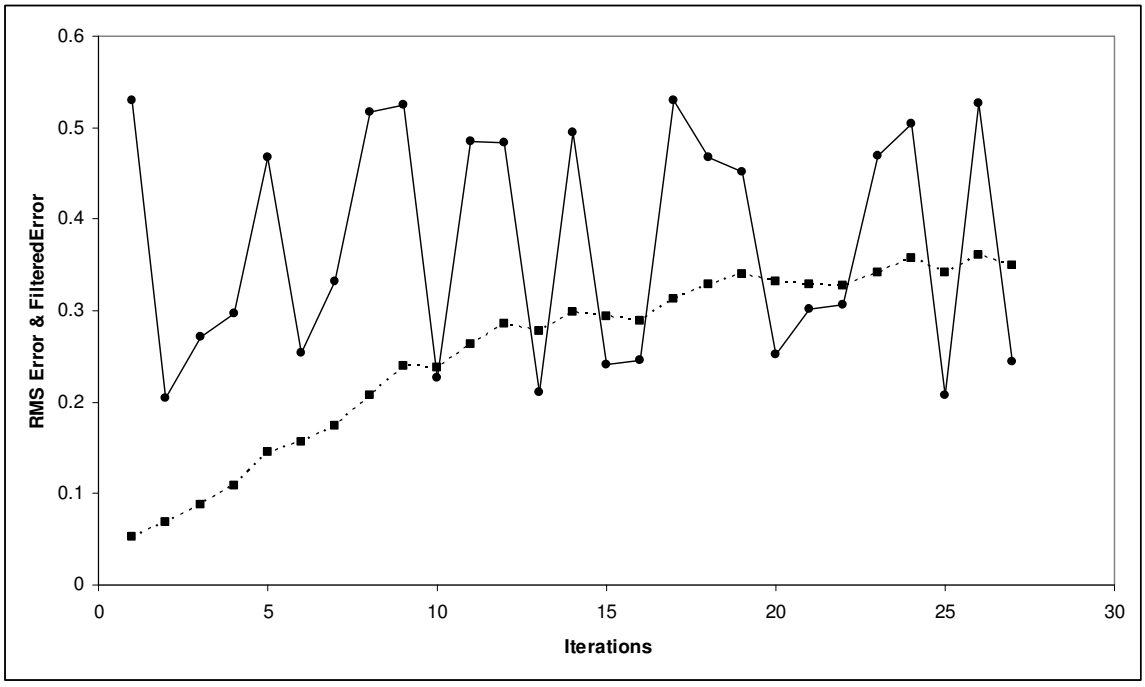


Fig 4.14b: RMS Error vs Filtered Error for Laminar-Laminar Flow with Excessive Iterations

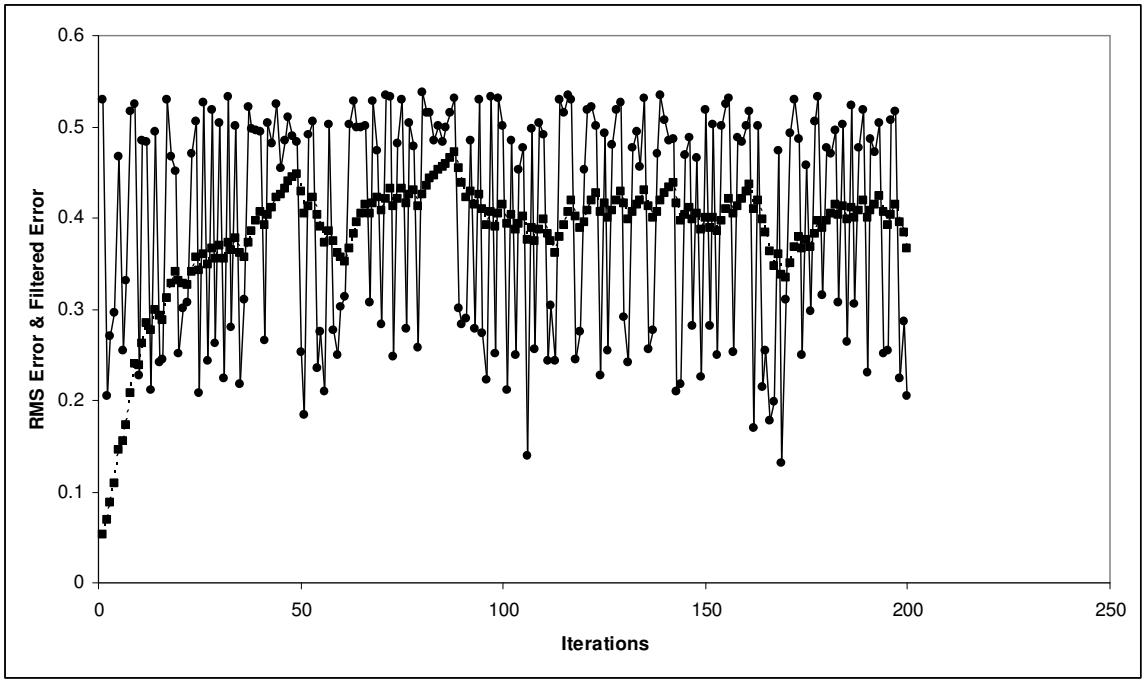


Table 4.28: Parameter values for Laminar-Laminar Flow

Parameter	With Excessive Iterations	With SS
a	-0.61653	-1.45648
b	0.447043	0.557754
c	-0.33146	-0.57551
SSD	0.292015	0.292455

Case 4.2.2 Liquid Flow – Turbulent

Vapor Flow – Laminar

Table 4.29: Final Optimization results for Turbulent-Laminar Flow

Parameter	a	b	c	SSD
Value	0.033814	1.501337	-0.97511	2.351466

Fig 4.15: Experimental Pressure Drop vs Model Pressure Drop for Turbulent-Laminar Flow

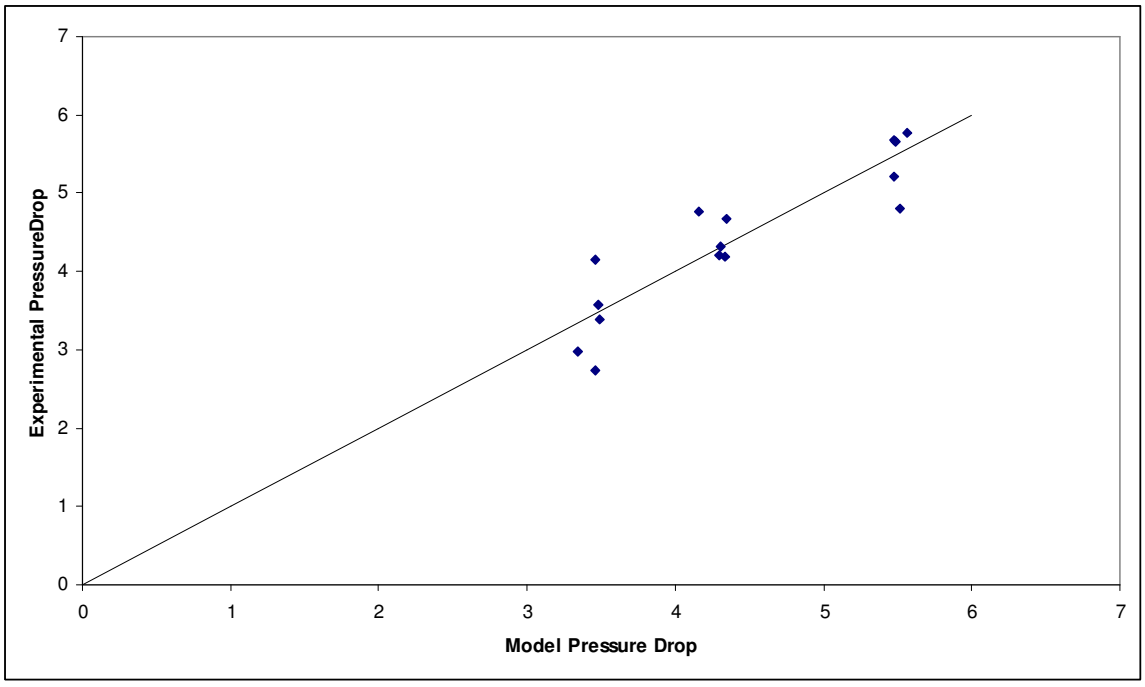


Fig 4.16a: RMS Error vs Filtered Error for Turbulent-Laminar flow with Steady State Stopping Criterion

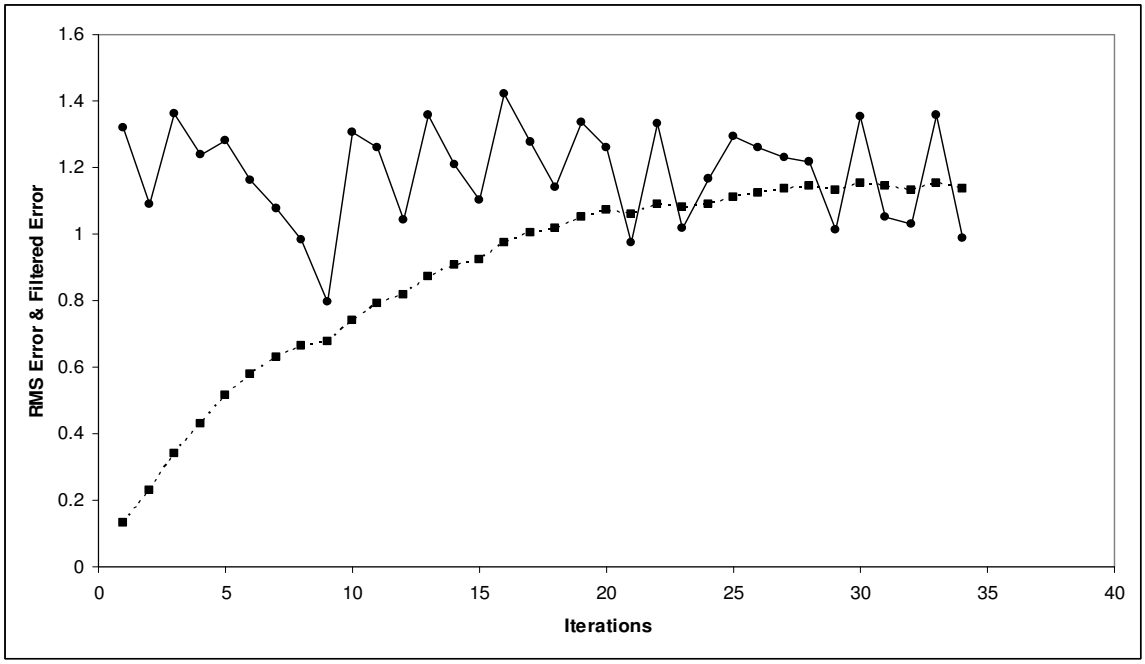


Fig 4.16b: RMS Error vs Filtered Error for Turbulent-Laminar Flow with Excessive Iterations

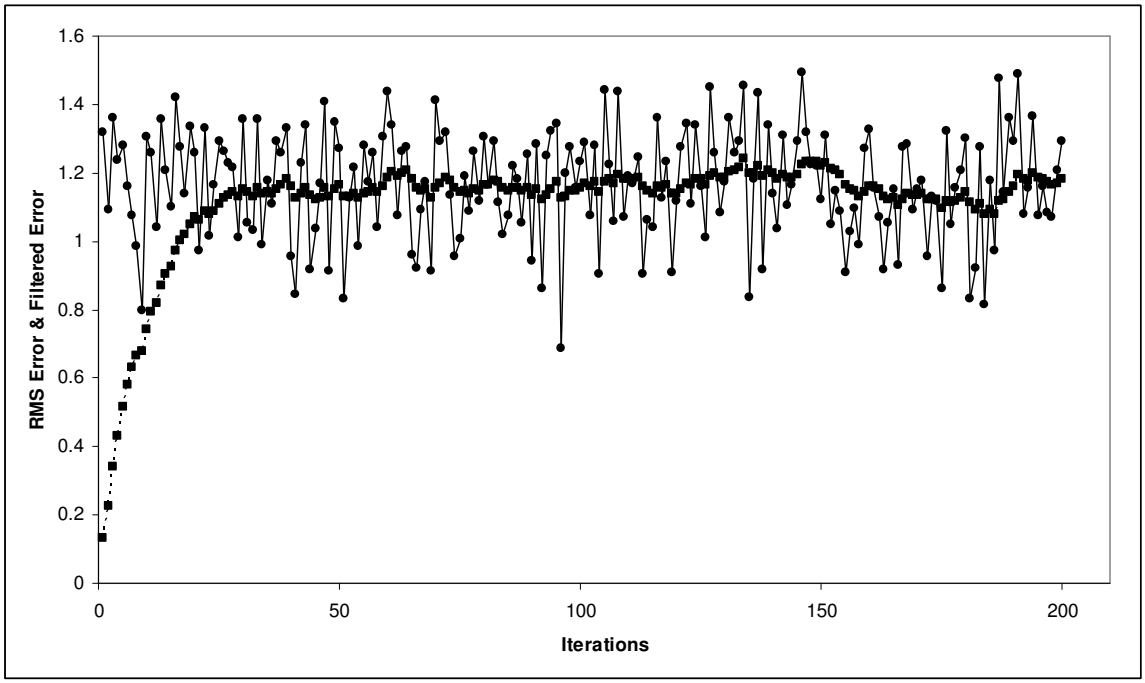


Table 4.30: Parameter values for Turbulent-Laminar Flow

Parameter	With Excessive Iterations	With SS
a	4.543739	0.935582
b	0.282529	0.794484
c	-0.21111	-0.59364
SSD	2.355028	2.355656

Case 4.2.3 Liquid Flow – Laminar

Vapor Flow – Turbulent

Table 4.31: Final Optimization results for Laminar-Turbulent Flow

Parameter	a	b	c	SSD
Value	0.901514	-1.42825	1.169466	0.008928544

Fig 4.17 Experimental Pressure Drop vs Model Pressure Drop for Laminar-Turbulent Flow

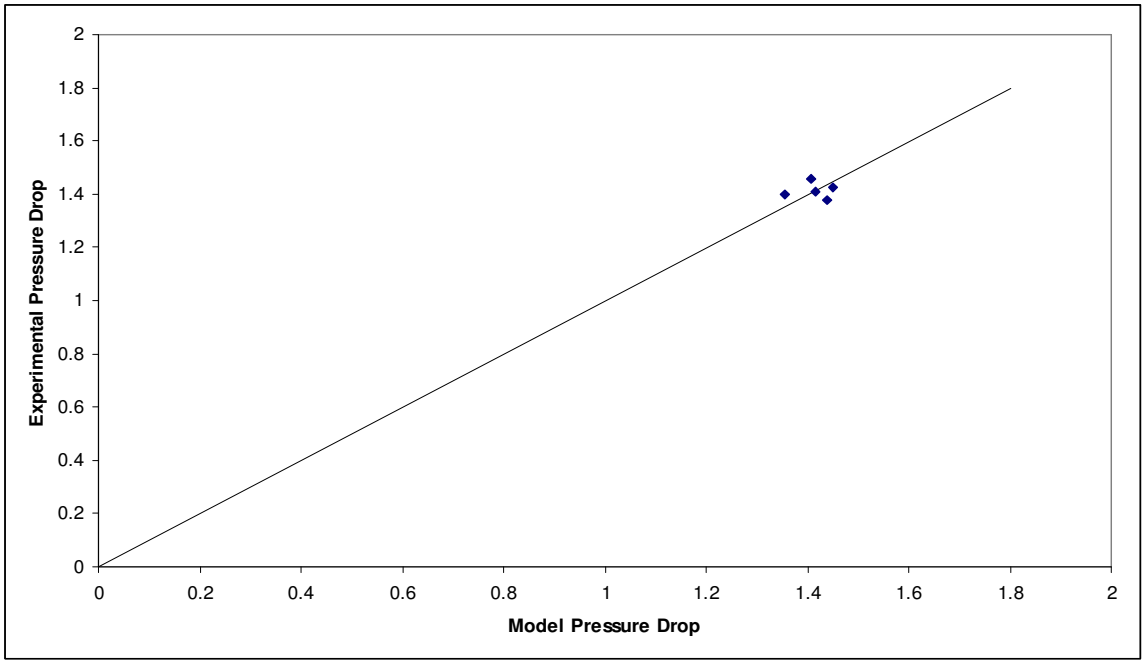


Fig 4.18a: RMS Error vs Filtered Error for Laminar-Turbulent flow with Steady State Stopping Criterion

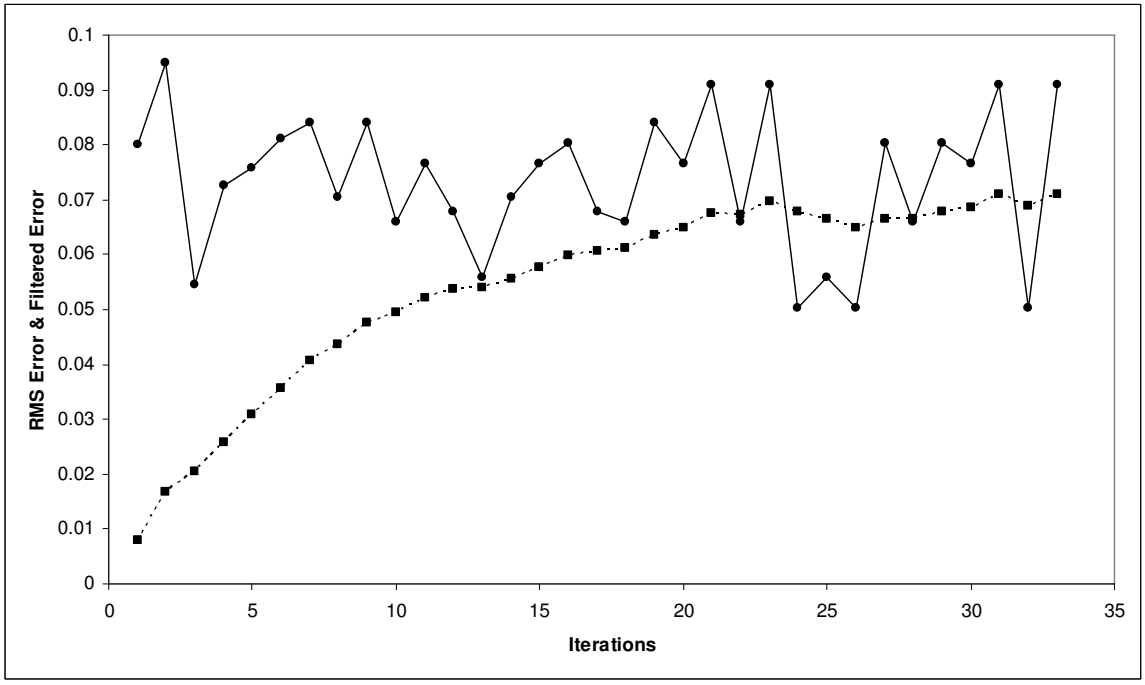


Fig 4.18b: RMS Error vs Filtered Error for Laminar-Turbulent Flow with Excessive Iterations

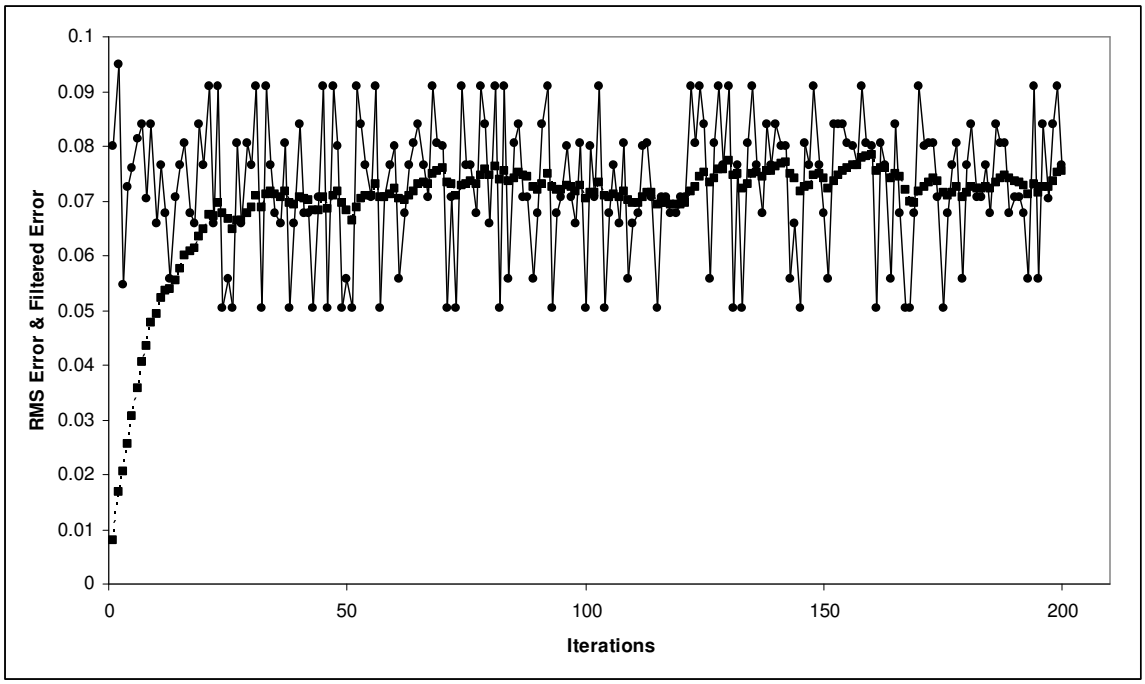


Table 4.32: Parameter values for Laminar-Turbulent Flow

Parameter	With Excessive Iterations	With SS
a	0.74480862985	0.649
b	0.807835642	0.828768
c	-0.677895135	-0.67923

SSD	0.008940903	0.008941

Case 4.2.4 Liquid Flow – Turbulent

Vapor Flow – Turbulent

Table 4.33: Final Optimization results for Laminar-Laminar Flow

Parameter	a	b	c	SSD
Value	0.076989	1.160383	-0.51798	2.805532

Fig 4.19: Experimental Pressure Drop vs Model Pressure Drop for Turbulent-Turbulent Flow

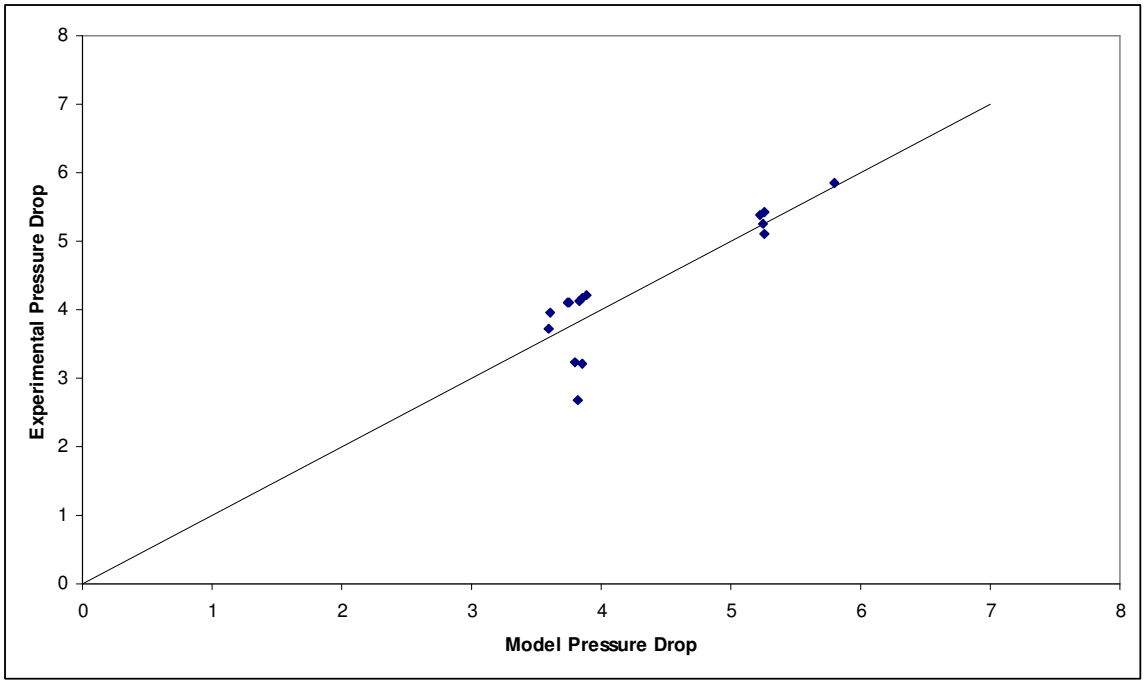


Fig 4.20a: RMS Error vs Filtered Error for Turbulent-Turbulent flow with Steady State Stopping Criterion

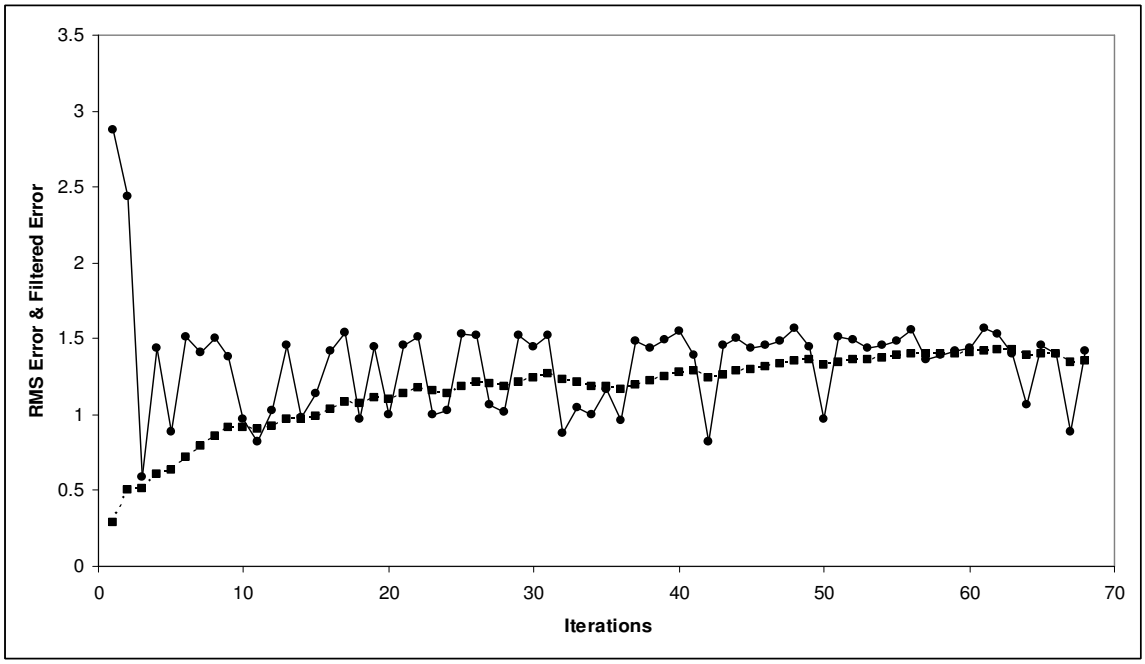


Fig 4.20b: RMS Error vs Filtered Error for Turbulent-Turbulent Flow with Excessive Iterations

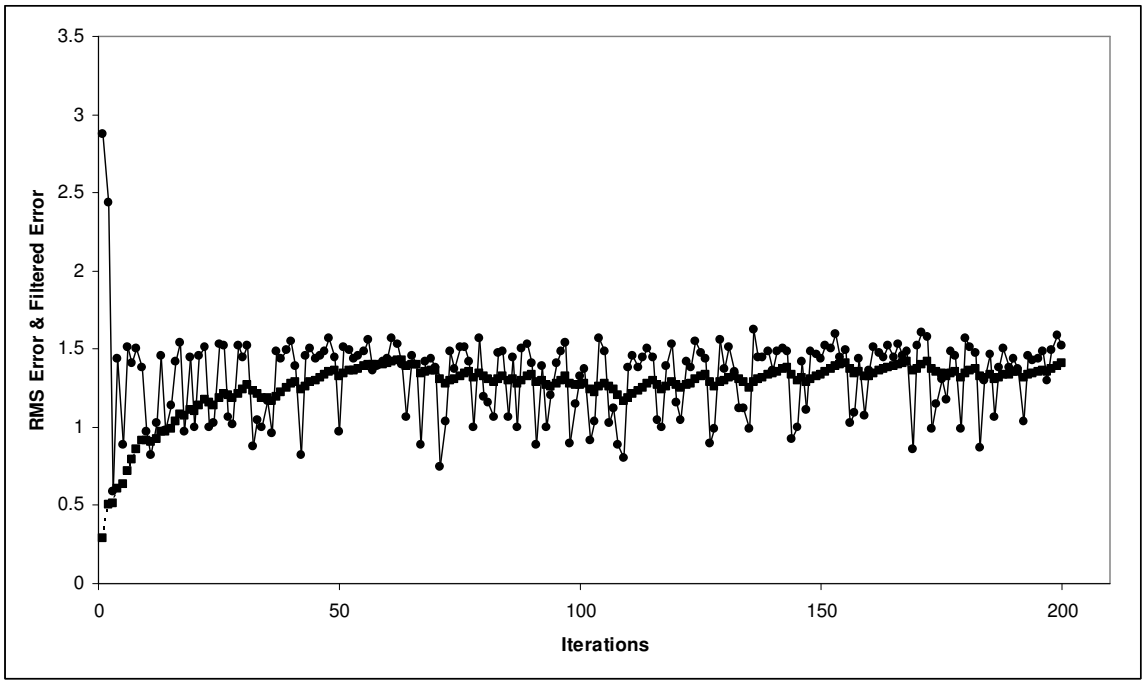


Table 4.34: Parameter values for Laminar-Laminar Flow

Parameter	With Excessive Iterations	With SS
a	0.62626503185	0.51116580305
b	0.793496239	0.79279523
c	-0.394532864	-0.371494198
SSD	2.800624	2.811018

4.3. Results of the Best-of-N Analysis

The Best-of-N analysis had been described in the previous chapter, and Equation (2.1.2) is used in the present study to calculate the number of trials the optimizer runs in order to determine the global optimum, i.e. the best possible model. It is also evident that the analysis is dependent on the stopping criteria terminating a trial at the local optimum. The previous sections can lead to a conclusion that the Steady State Stopping criterion does in fact do so, and one can progress with the analysis of the Best-of-N formula.

The present analysis of the Best-of-N formula is done based on running the optimizers on different models, and datasets, for a large number of trials (in this case 10,000) each starting with a random initial set of values. The analysis algorithm described in Section 3.4 is implemented on each set of points thus obtained. The algorithm is programmed in VBA and is reproduced in Appendix (C). The final result of the algorithm gives us the confidence with which the Best-of-N formula can predict the optimum within the predetermined best fraction of the results generated by a specific optimization algorithm. The testing algorithm also generates a cumulative distribution for the data, which is used in determining the higher limit for the required best fraction of the results. For the present set of discussed cases, it is required to be 90% confident that one of the best 10% of the results will be reported each time.

It has to be noted that the number of trials (10,000) though notably large, is not the same as an infinite number of runs, and consequently, the probabilities involved in Equation

(2.3) would not be absolutely accurate, and consequently, a fair degree of accuracy is assumed to be associated with them. In mathematical terms, we can say that the confidence in the results (as predicted by the above mentioned algorithm) is normally distributed, with

$$\mu = n.p \tag{4.3.1}$$

$$\sigma^2 = n.p.q \tag{4.3.2}$$

where μ is the mean, n is the number of sets involved, and p is the required probability, and $q = (1-p)$.

Considering the present situation, Equation (2.3) gives us 22 trials, and we have 10,000 points. This gives us $(10000/22) \approx 454$ sets, i.e. n; p is 0.9 based on the required confidence, and q is 0.1.

From Equations (4.3.1) and (4.3.2), we get μ = 409.1, and σ = 6.3957. If we were to consider $\mu \pm 3 \cdot \sigma$, a result between 94.336 % and 85.88% cannot be rejected.

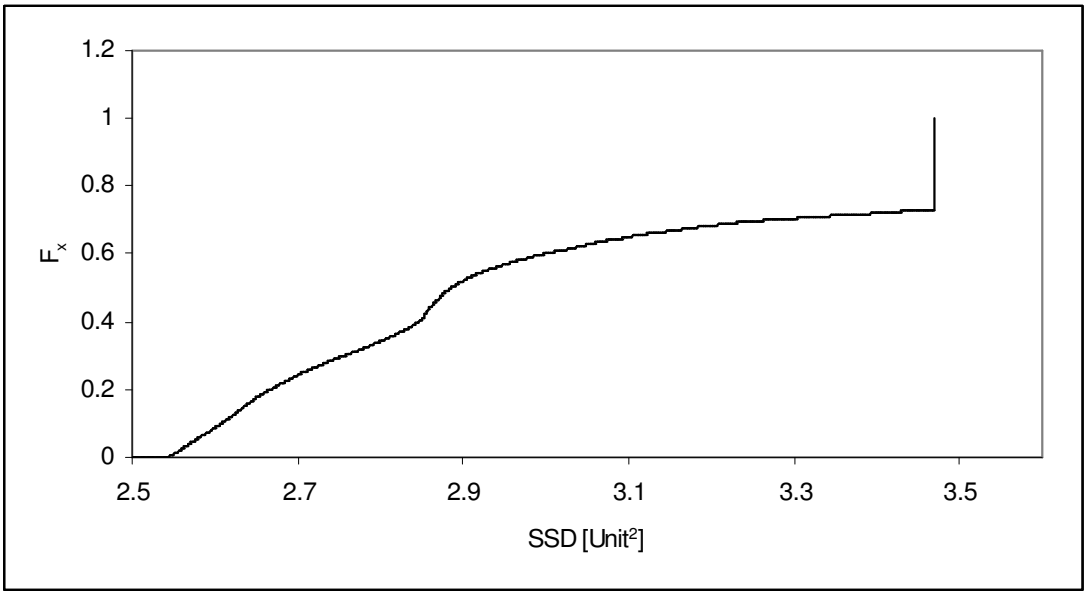
Case 4.3.1 Model Used: Neural Network

Optimizer: RRR's Algorithm

Dataset: Set A

The cumulative distribution of the 10000 datapoints is given in Figure 4.21. the testing algorithm reveals that the best 10% of the answers are reported 89.8% of the times when 22 sets of points are considered as predicted by the Best-of-N formula.

Fig 4.21: Cumulative Distribution for Case (4.3.1)



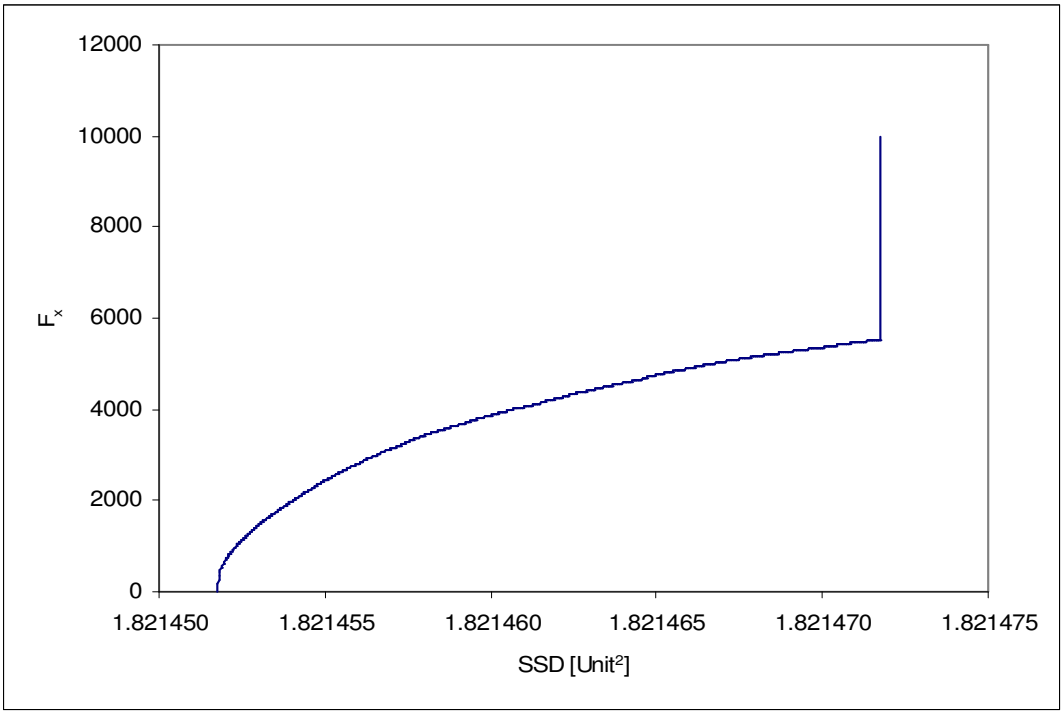
Case 4.3.2 Model Used: Third degree Polynomial Equation

Optimizer: RRR's Algorithm

Dataset: Set A

The cumulative distribution of the 10000 datapoints is given in Figure 4.22. the testing algorithm reveals that the best 10% of the answers are reported 89.4% of the times when we consider 22 sets of points as predicted by the Best-of-N formula.

Fig 4.22: Cumulative Distribution for Case (4.3.2)



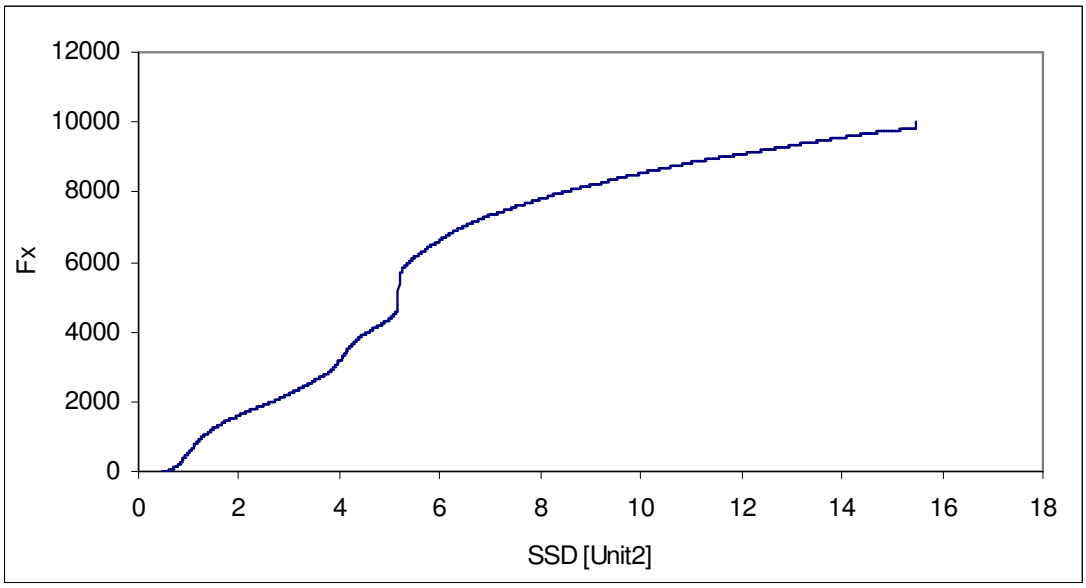
Case 4.3.3 Model Used: Neural Network

Optimizer: RRR's Algorithm

Dataset: Set B

The cumulative distribution of the 10000 datapoints is given in fig 4.23. the testing algorithm reveals that the best 10% of the answers are reported 87.6% of the times when we consider 22 sets of points as predicted by the Best-of-N formula. From the discussion presented in the beginning of this section, this would be in the range where the formula can not be rejected.

Fig 4.23: Cumulative Distribution for Case (4.3.3)



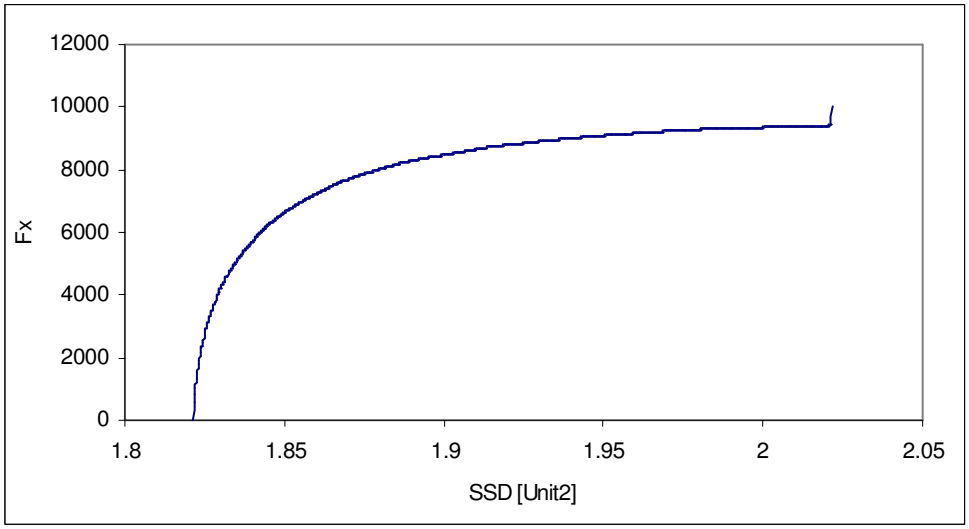
Case 4.3.4 Model Used: Third Degree Polynomial

Optimizer: RRR's Algorithm

Dataset: Set B

The cumulative distribution of the 10000 datapoints is given in fig 4.??. the testing algorithm reveals that the best 10% of the answers are reported 92.9% of the times when we consider 22 sets of points as predicted by the Best-of-N formula. This is under the range that was calculated for a normal distribution.

Fig 4.24: Cumulative Distribution for Case (4.3.4)



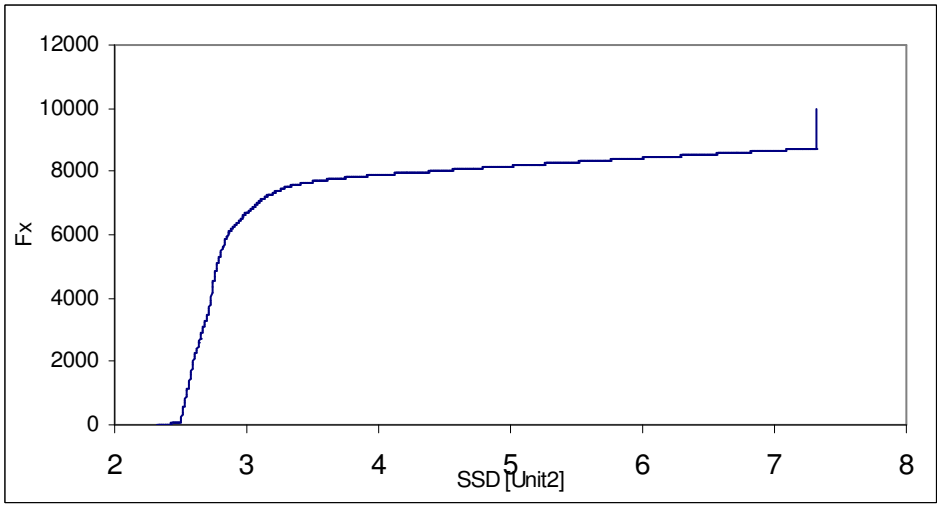
Case 4.3.5 Model Used: Neural Network

Optimizer: Hooke Jeeves Algorithm

Dataset: Set A

The cumulative distribution of the 10000 datapoints is given in fig 4.??. the testing algorithm reveals that the best 10% of the answers are reported 91.4% of the times when we consider 22 sets of points as predicted by the Best-of-N formula. This falls within the range of the normal distribution for the given data, and the formula can not be rejected.

Fig 4.25: Cumulative Distribution for Case (4.3.5)



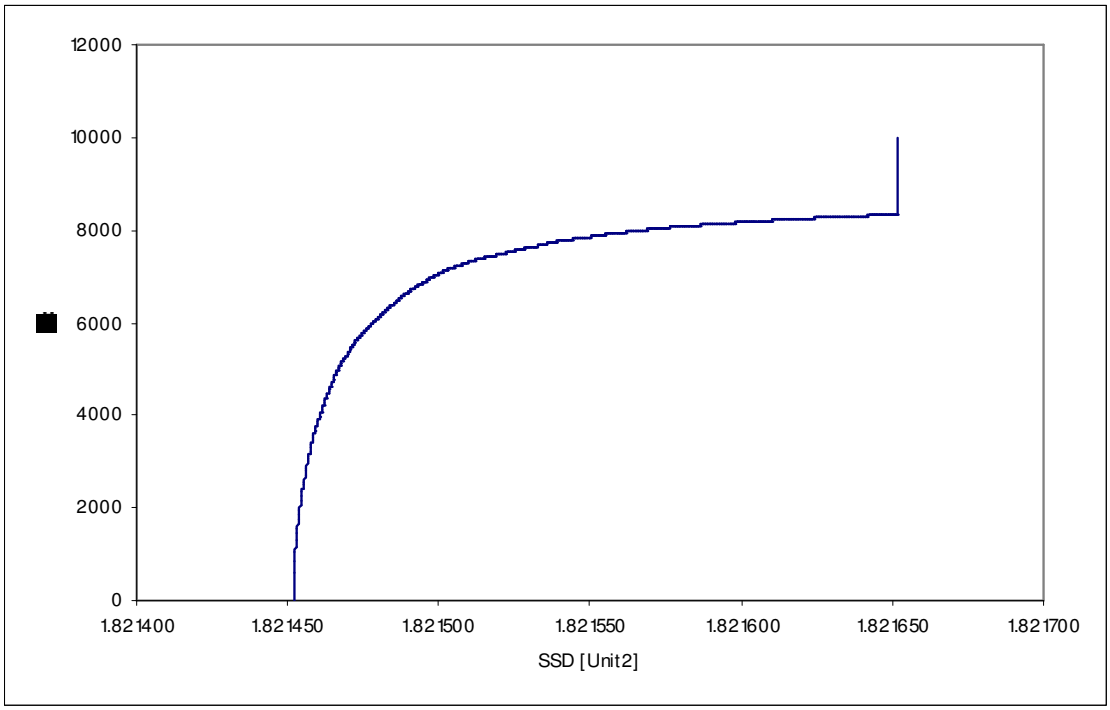
Case 4.3.6 Model Used: Neural Network

Optimizer: Hooke Jeeves Algorithm

Dataset: Set B

The cumulative distribution of the 10000 datapoints is given in fig 4.??. the testing algorithm reveals that the best 10% of the answers are reported 90.6% of the times when we consider 22 sets of points as predicted by the Best-of-N formula.

Fig 4.26: Cumulative Distribution for Case (4.3.6)



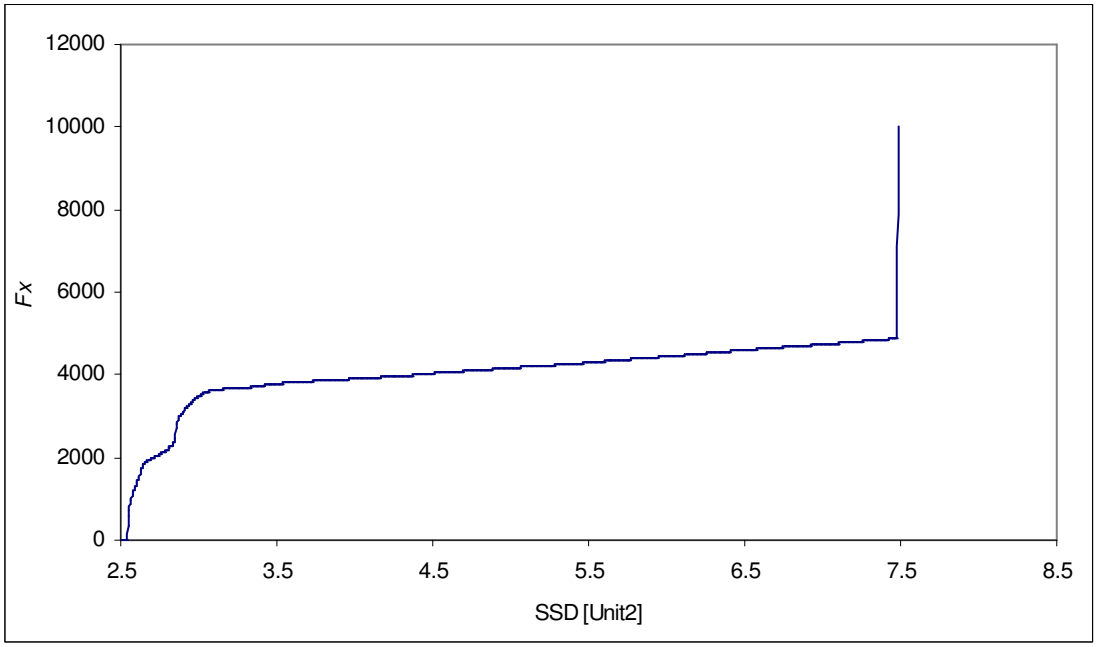
Case 4.3.7 Model Used: Neural Network

Optimizer: BFGS Algorithm

Dataset: Set A

The cumulative distribution of the 10000 datapoints is given in fig 4.??. the testing algorithm reveals that the best 10% of the answers are reported 92.4% of the times when we consider 22 sets of points as predicted by the Best-of-N formula.

Fig 4.27: Cumulative Distribution for Case (4.3.7)



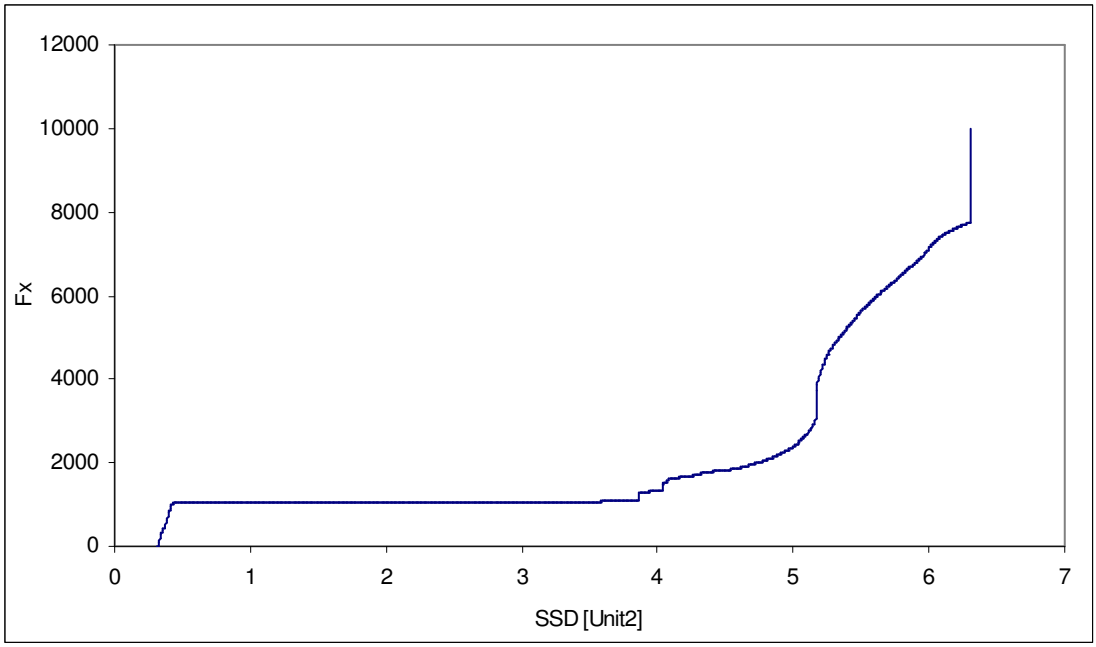
Case 4.3.8 Model Used: Neural Network

Optimizer: BFGS Algorithm

Dataset: Set B

The cumulative distribution of the 10000 datapoints is given in fig 4.??. the testing algorithm reveals that the best 10% of the answers are reported 91.0% of the times when we consider 22 sets of points as predicted by the Best-of-N formula.

Fig 4.28: Cumulative Distribution for Case (4.3.8)



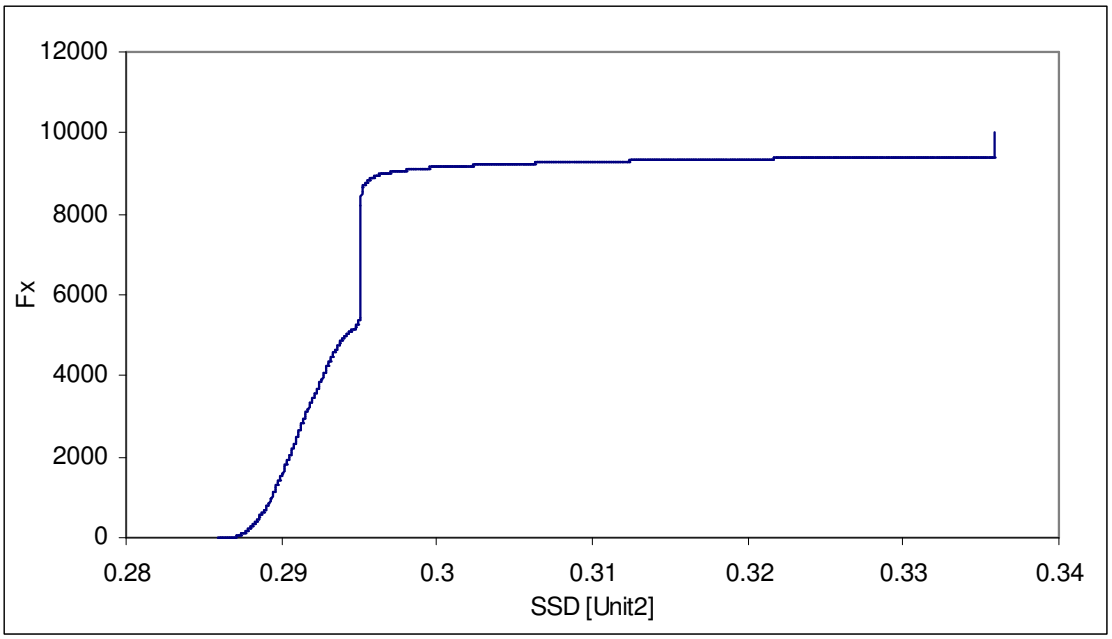
Case 4.3.9 Model Used: Lockahrt-Martinelli

Optimizer: RRR Algorithm

Dataset: Pressure Drop data for Laminar-Laminar Flow

The cumulative distribution of the 10000 datapoints is given in fig 4.??. the testing algorithm reveals that the best 10% of the answers are reported 94.2% of the times when we consider 22 sets of points as predicted by the Best-of-N formula.

Fig 4.29: Cumulative Distribution for Case (4.3.8)



to summarize the results of this section, the following tables are presented. Table 3?a is based on a 90% confidence that the best 10% of the solutions will be reported and Table 3?b is based on a 95% confidence that the best 5% of the solutions will be reported.

Table 4.35a Results of Best-of-N analysis: Percentage of occurrence of the best 10% of the solutions.

Data Set	Model	RRR	HJ	BFGS
A	NN	89.8	91.4	92.4
	Poly	89.4	-	-
B	NN	87.6	90.6	91
	Poly	92.9	-	-
2-Phase PD	L-M	94.2	-	-

Table 4.35a Results of Best-of-N analysis: Percentage of occurrence of the best 10% of the solutions.

Data Set	Model	RRR	HJ	BFGS
A	NN	95.6	94.5	97.5
	Poly	98.1	-	-
B	NN	93.7	95.5	96.1
	Poly	95.3	-	-
2-Phase PD	L-M	99	-	-

4.4 Discussions

The Steady State Stopping Criterion has been applied in earlier work [3,2] in neural network training, and in other examples of nonlinear optimization. At the same time, the Best-of-N analysis has been used exclusively in neural network training [4].

The algorithm combining the two ideas is observed to be successful in modeling noisy data, and in modeling pressure drop data for a two-phase flow system. the algorithm is shown to be capable of reporting the desired globally optimum model for given process data.

The stopping criterion is seen to provide successful results in most cases. The stopping criterion is observed to terminate the trial fairly quickly, and in most cases the excessive iterations do not generate significantly better answers. This also reinforces the results obtained in the testing of the Best-of-N starting method. Considering the former, it has been observed that keeping the threshold on $R_{statistic}$ to be 0.85 has its advantages over the

previously used value of 1. The studies on Set B also indicate that the stopping criteria can sometimes fail at detecting steady state because of the occurrence of Type-II errors. In the specific case of Figures 4.12a and 4.12b, it can be visually confirmed that a lower $R_{statictic}$ threshold can improve the performance of the stopping criteria. At the same time, one has to appreciate that the use of the Best-of-N formula helps these situations because the results are still generated with the same confidence irrespective of the set threshold in the stopping criteria. The issues of the type-II errors arising in the steady state identification require more scrutiny, and the use of a lower threshold for $R_{statistic}$ also warrants more detailed study.

Previous studies have claimed that no *a priori* information is required in selecting Fw(a) and Fx(a)to determine N. However we do not guarantee that the method will give us the desired results every time. A counter example to this effect can be a surface with shallow optima all over, and one global minimum located at a very narrow valley, i.e. there is only a 1% chance of ever hitting the global minimum. In this case, choosing the best 10% of the results will not yield a good optimum, and a choice of the best 0.5% might give an N large enough that the global minimum could be found.

The Best-of-N method has one distinct disadvantage akin to most multisart optimizers, i.e. they consume a considerable amount of computation time. Snyman and Fatti, have developed an approach to determine N based on Baysean statistics. In this, the optimizer is started ‘n’ times initially. The CDF of the RMS results (the OF) provides information about the distribution of the OF values, and this, along with a user specified confidence,

helps determine the value of N needed. In effect, instead of doing unnecessarily excessive runs, the algorithm looks at each new solution as it is generated to determine (or to update) how many runs will it be necessary to generate the global optima. Further work can be carried out in this regard, where the logic could replace the Best-of-N criteria or the two ideas could be combined.

Another point to be noted in the exercise as a whole is the calculation of the SSD between the model and the experimental data. The present work uses a simple definition of the error in the calculation, but there are more accurate methods being studied. The VBA code used in the present study is effective, but it takes up a lot of computational burden in the process. The code can be streamlined by reevaluation of the necessary calculations. This can also help in the application of the logic in more indirect methods involving the evaluation of the derivatives of the objective function, which, in the case of empirical modeling, can be extremely time consuming.

CHAPTER V

CONCLUSION

The Best-of-N analysis originally developed to determine the number of random starts required in neural network training has been extended to more generic empirical modeling applications. It has been combined with the previously studied Steady State Stopping Criterion to develop a global optimization logic for nonlinear empirical modeling.

The combined logic has been tested on a variety of modeling objectives, and applications. The steady state stopping criterion successfully determines the point of termination in each individual trial, and the Best-of-N analysis is analyzed to prove that the user defined confidence in finding the global minimum is met. It can thus be concluded that the combined logic, as a whole, gives successful and efficient results.

Further research is warranted in the removal of Type-II errors that may occur in the identification of steady state, and in determining the optimum threshold for the $R_{statistic}$ in the steady state identifier. The Best-of-N starting methods can also be studied further in attempts to reduce the number of trials involved in obtaining a specific objective. The present algorithm is effective in its execution, but the code can be streamlined with respect to the calculations involving the computation of the SSD.

The logic can be applied in commercial modeling applications subject to the dissemination of the above findings and the streamlining of the computational burden involved in the modeling process.

REFERENCES

1. T. F. Edgar, D. M. Himmelbalu, L. S. Ladson, Optimization of Chemical Processes, McGraw-Hill NY, 2001
2. V. Padmanabhan, "A Study of a Novel Stopping Criterion for Optimization," A thesis in Chemical Engineering, Oklahoma State University, 2005
3. S. Cao, R. R. Rhinehart, "An efficient method for on-line identification of steady state," Journal of Process Control, Vol. 5, No. 6, pp.363-374, 1995
4. M. S. Iyer, R. R. Rhinehart, "A method to determine the required number of neural network training repetitions," IEEE Transactions on Neural Networks, Vol. 10, No. 2, pp. 427-432, 1999
5. R.R. Rhinehart, "Best-of-N training paper explanation," School of Chemical Engineering, Oklahoma State University, private communication
6. V. Padmanabhan, "A novel termination criterion for optimization," School of Chemical Engineering, Oklahoma State University
7. A.V. Balakrishnan, M. Thomas, Lecture notes in Control and Information sciences, Springer-Verlag, NY, 1982
8. R. M. Betha, R. R. Rhinehart, Applied Engineering Statistics, Marcel Dekker, NY, 1991
9. G. E. P. Box, G. M. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, 1976

APPENDIX A

CONTRIVED DATA

The first set of contrived data is based on the following model equation:

$$y = 5\exp\left(\frac{20}{x}\right)$$

to make the data reflect an actual process, a normally distributed random error is introduced with a variance of 1 unit. A set of 30 data points are selected for the study.

Table A.1: Contrived Data with errors incorporated in the dependent variable (Set A)

Serial No.	x	y
1	1	0.361161
2	4	-0.38911
3	7	0.578485
4	10	0.67641
5	13	0.583247
6	16	0.944263
7	19	1.342423
8	22	2.017296
9	25	2.233575
10	28	2.318014
11	31	3.023673
12	34	2.981573
13	37	2.704862

Serial No.	x	y
14	40	3.498605
15	43	2.965337
16	46	2.785793
17	49	3.486505
18	52	3.809234
19	55	3.587298
20	58	3.163685
21	61	3.996238
22	64	3.26411
23	67	3.782867
24	70	4.056131
25	73	3.730098
26	76	3.401893
27	79	3.678942
28	82	3.547743
29	85	3.626967
30	88	4.330047

The second set of contrived data attempts to realize an actual process more accurately. There are errors associated with both the dependent and independent variables. The original data is based on the following model:

$$y = \exp\left(-30\left(x - 0.5\right)^2\right)$$

Both the dependent and independent variables have normally distributed random errors incorporated in them. The base value of *x* used in the table below is the basis of the calculation of both the x and y values, both of which have errors with a variance of 0.4 and 1 associated with them respectively. Here, the base x refers to the nominal value believed to be true by the experiment, and x is the actual but unknowable value. Y is thus measured from x (which is already noisy) and has it's own noise incorporated too.

Table A.2: Contrived Data with errors associated with both dependent and independent variables (Set B)

Serial No.	Base x	x	y
1	0	0.050184	-0.05865
2	0.025	0.053933	0.027031
3	0.05	0.121355	0.043433
4	0.075	0.05717	0.014812
5	0.1	0.117694	0.040712
6	0.125	0.12364	0.038379
7	0.15	0.139239	0.024217
8	0.175	0.207925	0.070216
9	0.2	0.224697	0.054074
10	0.225	0.216106	0.143239
11	0.25	0.296369	0.113829

Serial No.	Base x	x	y
12	0.275	0.256715	0.210443
13	0.3	0.32142	0.285091
14	0.325	0.33107	0.383834
15	0.35	0.365156	0.522981
16	0.375	0.306388	0.587964
17	0.4	0.415347	0.754327
18	0.425	0.475975	0.829305
19	0.45	0.436924	0.878163
20	0.475	0.478253	0.940252
21	0.5	0.495903	0.965767
22	0.525	0.521928	1.025498
23	0.55	0.552716	0.932712
24	0.575	0.607042	0.868465
25	0.6	0.590964	0.747658
26	0.625	0.565694	0.596703
27	0.65	0.612117	0.515776
28	0.675	0.630977	0.422942
29	0.7	0.662263	0.342466
30	0.725	0.753676	0.275027
31	0.75	0.777691	0.165545
32	0.775	0.763522	0.079979
33	0.8	0.772853	0.032768
34	0.825	0.87461	-0.01089
35	0.85	0.788645	-0.00213
36	0.875	0.908301	0.007084
37	0.9	0.896038	-0.01642
38	0.925	0.859823	0.005293
39	0.95	0.943584	0.020634

Serial No.	Base x	x	y
40	0.975	1.035687	0.040919
41	1	0.996513	0.004479

APPENDIX B

PRESSURE DROP DATA

AND

EXAMPLE CALCUALTIONS FOR PRESSURE DROP

IN TWO-PHASE FLOW

S. no.	Delta_Pr.	dP_STF	large air flow Fl_1_Filt (ft^3/min)	small air flow Fl_2_Filt (ft^3/min)	liquid flow rate Fl_3_Filt (kg/hr)	Water Ht. (m) W_Ht_Filt
1	0.0507	0.0508	1.3498	0.0510	91.1077	0.0334
2	0.0688	0.0688	1.5193	0.0516	92.5997	0.0371
3	0.0479	0.0479	1.5942	0.0475	90.4832	0.0334
4	0.0515	0.0515	1.6495	0.0509	92.2022	0.0335
5	0.0381	0.0381	1.6680	0.0513	89.0564	0.0244
6	4.3754	4.3164	24.9847	0.0544	519.5394	3.0113
7	4.2531	4.2510	24.9838	0.0537	520.4920	3.0191
8	4.1076	4.2632	24.9760	0.0518	513.5342	3.0412
9	4.4256	4.4681	24.9957	0.0557	525.4731	3.0210
10	6.6422	4.9260	12.1766	0.0635	295.2589	3.8030
11	6.6132	6.6132	1.3310	0.0477	88.4521	4.6495
12	6.5460	6.5460	1.3907	0.0479	88.3032	4.6502
13	6.6422	6.6422	1.6260	0.0499	90.5244	4.6483
14	6.5842	6.5842	1.5733	0.0496	89.6316	4.6484
15	6.6224	6.6224	1.3945	0.0481	89.0713	4.6542
16	4.1848	3.9782	1.5374	1.0012	497.4645	2.8596
17	4.6788	4.1810	1.6143	1.0012	496.4514	2.8711
18	4.2142	4.0723	1.6453	1.0011	494.1385	2.8379
19	5.3232	4.1241	1.5262	1.0010	484.4626	2.8457
20	3.7591	4.0752	1.4381	0.7254	417.8737	2.7753
21	3.0983	2.9573	6.9099	0.0535	506.7354	2.0248
22	3.1356	3.1676	6.9465	0.0548	518.5857	2.0634
23	2.2391	2.9537	6.9189	0.0524	526.2402	2.0251
24	3.2055	2.8058	6.8981	0.0526	514.9471	2.1135
25	2.6775	2.9935	7.0390	0.0532	505.1873	2.0855

S. no.	Delta_Pr.	dP_STF	large air flow Fl_1_Filt (ft^3/min)	small air flow Fl_2_Filt (ft^3/min)	liquid flow rate Fl_3_Filt (kg/hr)	Water Ht. (m) W_Ht_Filt
26	4.8066	5.0599	1.4871	0.5015	516.2134	3.6626
27	5.6527	5.2805	1.5952	0.5011	517.7882	3.6403
28	5.1833	5.1967	1.6012	0.5010	517.1487	3.6385
29	4.7757	5.3291	1.3940	0.5000	519.9964	3.6841
30	5.2051	5.2203	1.7096	0.5006	515.9110	3.6371
31	5.2872	5.2872	1.4949	0.0504	102.3022	3.7237
32	5.3022	5.3022	1.5885	0.0518	101.6442	3.7253
33	5.2826	5.2826	1.5451	0.0517	101.3847	3.7269
34	5.3060	5.3060	1.4748	0.0511	101.0812	3.7267
35	5.1835	5.1835	1.6051	0.0503	100.5808	3.7672
36	3.0200	3.0019	1.5660	0.5015	99.8789	2.1164
37	2.9130	2.9137	1.3943	0.5012	99.9950	2.0793
38	2.8286	2.8286	1.5553	0.5014	99.9650	2.0742
39	2.7656	2.8260	1.3057	0.5004	99.8528	2.0707
40	2.9677	2.9228	1.3522	0.1949	100.9842	2.0342
41	3.2543	3.2543	1.2259	0.0480	99.8881	2.3033
42	3.2596	3.2596	1.2069	0.0498	100.2032	2.3040
43	3.3017	3.3017	1.4645	0.0517	101.5342	2.3096
44	3.2772	3.2772	1.2805	0.0473	99.9780	2.3085
45	3.7305	3.7305	1.4216	0.0521	101.1915	2.3161
46	4.1507	3.4547	1.3397	1.0008	297.1745	2.3546
47	2.8382	3.3660	1.3290	1.0005	299.4958	2.3747
42	3.2596	3.2596	1.2069	0.0498	100.2032	2.3040
43	3.3017	3.3017	1.4645	0.0517	101.5342	2.3096
44	3.2772	3.2772	1.2805	0.0473	99.9780	2.3085
45	3.7305	3.7305	1.4216	0.0521	101.1915	2.3161
46	4.1507	3.4547	1.3397	1.0008	297.1745	2.3546
47	2.8382	3.3660	1.3290	1.0005	299.4958	2.3747
48	3.6234	3.5409	1.3920	1.0002	298.5453	2.3681
49	2.7378	3.3277	1.5377	1.0003	304.5260	2.3544
50	2.9877	3.2834	1.3748	0.6925	241.8319	2.2926
51	3.1209	2.7576	7.0204	0.0509	496.4620	1.9407
52	2.7202	2.7473	7.0880	0.0537	506.1500	2.1037
53	3.2052	2.7254	7.0773	0.0531	502.9768	2.1059
54	1.9616	2.5210	6.9554	0.0512	502.3859	1.9336
55	3.1072	2.7301	7.0119	0.0547	504.0659	2.0360
56	1.4118	1.4118	7.0717	0.0510	99.3628	0.9594
57	1.4246	1.4246	6.9956	0.0514	100.1572	0.9835
58	1.3984	1.3984	6.9280	0.0481	98.4197	0.9171
59	1.3768	1.3768	6.9778	0.0514	99.0677	0.9765

S. no.	Delta_Pr.	dP_STF	large air flow FI_1_Filt (ft^3/min)	small air flow FI_2_Filt (ft^3/min)	liquid flow rate FI_3_Filt (kg/hr)	Water Ht. (m) W_Ht_Filt
60	1.4579	1.4579	6.9697	0.0499	98.8602	0.9545
61	1.1030	1.1030	1.5693	0.0492	99.2359	0.7712
62	1.1041	1.1041	1.7420	0.0491	98.4339	0.7674
63	1.0864	1.0864	1.4532	0.0503	98.4255	0.7726
64	1.0923	1.0923	1.4741	0.0511	98.3083	0.7760
65	1.0722	1.0722	1.4127	0.0511	100.2193	0.7680

Example Calculation:

Density of Air

The density of air at ambient conditions can be found from the ideal gas law which requires pressure (P), and molecular weight (MW), the gas constant (R), and temperature (T):

$$\rho_g = \frac{MWP_{avg}}{RT_{avg}} \qquad \text{(i)}$$

For example:

$$\rho_g = \frac{24.9\frac{lb_m}{lbmol} * 742.2 mmHg}{998.9\frac{mmhg.ft^3}{lbmol.K} * 293.15K} = 0.06313\frac{lb_m}{ft^3} = 1.0135\frac{kg}{m^3}$$

In this work, the pressure represents the average pressure in the two-phase flow column, and the temperature represents the water temperature. The molecular weight of 24.9 lbm/lbmole represents that of saturated air at the water temperature.

Density of Water

$$\rho = 28.282 \frac{kg}{ft^3} = 998.77 \frac{kg}{m^3}$$

Void Fraction and Two-Phase Density

The void fraction is calculated based on the height of the liquid in the column and the height of the column.

$$\varepsilon_g = \frac{Vol_g}{Vol_{total}} = \frac{h_v}{h} \qquad \text{(ii)}$$

$$\varepsilon_g = \frac{2.6021m}{5.44m} = 0.4783$$

The two-phase density is then calculated using,

$$\rho_{TP} = \varepsilon_g . \rho_g + \left(1 - \varepsilon_g\right) . \rho_l \qquad \text{(iii)}$$

For example:

$$\rho_{TP} = 0.4783 * 1.0135 \frac{kg}{m^3} + \left(1 - 0.4783\right) * 998.77 \frac{kg}{m^3} = 521.5133 \frac{kg}{m^3}$$

Reynolds' Number

The Reynolds' number for the liquid is defined as:

$$\mathrm{Re}_l = \frac{D\dot{m}_l}{A\mu_l} \qquad \text{(iv)}$$

Where,

D = Diameter of pipe or tube

$\dot{m}_l$ = mass flow rate of liquid

A = Cross sectional area of pipe or tube

μ_l = viscosity of liquid

For example:

$$\mathrm{Re}_l = \frac{0.026m * 0.1372 \frac{kg}{s}}{5.57 \times 10^{-4} m^2 * 0.00109 \frac{kg}{ms}} = 5878.1117$$

Similarly for the gas:

$$\mathrm{Re}_g = \frac{D\dot{m}_g}{A\mu_g} \qquad \text{(v)}$$

For example:

$$\mathrm{Re}_l = \frac{0.026m * 0.00123 \frac{kg}{s}}{5.57 \times 10^{-4} m^2 * 3.23E-05 \frac{kg}{ms}} = 1780.3518$$

Observing the Reynolds numbers in our example, the Liquid is in turbulent flow, and the gas is in laminar flow for this example. Hence, the Lockhart-Martinelli constant is given by the following Equation [2]:

$$C_i = a_i \, \mathrm{Re}_l^{\,b_i} \, \mathrm{Re}_g^{\,c_i} \qquad \text{(vi)}$$

$$C = 0.26464 * 5878.1117^{0.7549} * 1780.3518^{-0.3664} = 11.9417$$

Mass Fraction, x_g

The mass fraction of the gas can be calculated as shown below. The mass fraction of the liquid can be easily determined by taking the difference of x_g from 1. this is taken into account in the subsequent equations.

$$x_g = \frac{m_g}{m_l + m_g} = \frac{0.00123}{0.00123 + 0.1372} = 0.00889$$

Friction Factor, f

The friction factor for the fluid flow can be given by the following relation.

$$f_l = \frac{64}{\mathrm{Re}_l} = \frac{64}{5878.1117} = 0.01088$$

$$f_g = \frac{64}{\mathrm{Re}_g} = \frac{64}{1780.3518} = 0.03594$$

Note that the fluids are both in laminar flow. If the liquid is in turbulent flow, the following relation can be used:

The Martinelli multiplier is calculated as follows.

$$X^2 = \frac{\left(\frac{\Delta P_f}{L}\right)_l}{\left(\frac{\Delta P_f}{L}\right)_g} = \frac{f_l \left(1 - x_g\right)^2 \rho_g}{f_g \left(x_g\right)^2 \rho_l} \qquad \text{(vii)}$$

$$X^2 = \frac{0.01088 * \left(1 - 0.00889\right)^2 * 1.0135}{0.03594 * \left(0.00889\right)^2 * 998.77} = 3.8130$$

$$X = 1.9526$$

The frictional multiplier that results from the Lockhart-Martinelli correlation is then given by

$$\left(\phi_g\right)^2 = 1 + CX + X^2 \qquad \text{(viii)}$$

$$\left(\phi_g\right)^2 = 1 + 11.9417 * 1.9526 + 3.8130 = 28.3399$$

The single phase frictional pressure drops for the gas phase is given by:

$$-\left(\frac{\Delta P_f}{L}\right)_g = \frac{2.f_g.\left(\frac{m}{A}\right)^2.\left(x_g\right)^2}{\rho_g.D} = \frac{2 * 0.03594 * \left(\frac{0.00123}{5.57E-04}\right)^2 * 0.00889^2}{1.0135 * 0.026} = 13.3550\frac{Pa}{m}$$

The hydrostatic head is thus calculated by:

$$\Delta P = \rho_{TP}.g = 521.5133 * 9.8 = 5204.7034\frac{Pa}{m}$$

The two-phase frictional pressure drop is given by the following relation:

$$-\left(\frac{\Delta P_f}{L}\right)_{TP} = -\left(\frac{\Delta P_f}{L}\right)_g.\left(\phi_g\right)^2 = 13.3550 * 28.3399 = 375.699\frac{Pa}{m}$$

Thus the total pressure drop per unit length is obtained by combining the hydrostatic head and the two-phase pressure drop.

$$-\left(\frac{\Delta P_f}{L}\right) = 5204.7034 + 375.699 = 5580.4029\frac{Pa}{m}$$

Multiplying the above with the height of the column, we obtain the pressure prop for a two-phase system.

$$-\left(\Delta P\right) = 5580.4029\frac{Pa}{m} * 5.44m = 30357.3922Pa = 4.4018Psi$$

APPENDIX C

COMPUTER PROGRAMS

All the programming is done on Visual Basic for Applications based on MS EXCEL. The three main programs involved are generic enough that minor modifications are required when a different function is used.

This is the list of Public variables used in the entire set of programs.

```
'Prithwijit Ghoshal
'List of Public variables used between the optimization routines
Public zip As Integer

Public Xe() As Double   'acutal X
Public Ye() As Double   'acutal Y
Public Xs() As Double   'x scaled
Public Ys() As Double   'y scaled
'used in scaling the contrived data
'definitions are obvious from the var. names
Public Xmax As Double
Public Xmin As Double
Public Ymin As Double
Public Ymax As Double
Public Xmid As Double
Public Ymid As Double

Public NumTrials As Integer 'number of trials
Public nt As Integer        'counter for output
```

```
Public Npoints As Integer   'number of data points
Public Nrand As Integer     'number of random picks.. % of Npoints

'variables defined for the SS stopping criteria
Public Nf, Df, Xf, Sumold

'used in the actual optimization routine
'to track changes in the x values
Public X(20) As Double
Public xo(20) As Double
Public dX(20) As Double
```

Subroutines: these routines are common to all the three optimization routines with minor modifications for BFGS, which are shown later.

This routine takes the data and scales it between -0.8 and 0.8. These scaled values are used in the actual calculations.

```
Sub Initial_Calculations()
'Prithwijit Ghoshal
'to be called by the main HRo routine once and stores the restuls in a globally
defined array set
   ActiveWorkbook.Sheets("Sheet1").Activate
Dim I As Integer


   For I = 1 To Npoints
      Xe(I) = Cells(12 + I, 3).Value
      Ye(I) = Cells(12 + I, 4).Value
```

```
    Next I
    'finding the max and min values of x and y
    'will be used to scale them
    Xmax = Xe(1)
    Xmin = Xe(1)
    Ymax = Ye(1)
    Ymin = Ye(1)

    For I = 2 To Npoints
        If Xmax < Xe(I) Then: Xmax = Xe(I)
        If Xmin > Xe(I) Then: Xmin = Xe(I)
        If Ymax < Ye(I) Then: Ymax = Ye(I)
        If Ymin > Ye(I) Then: Ymin = Ye(I)
    Next I
    Xmid = (Xmin + Xmax) / 2
    Ymid = (Ymin + Ymax) / 2
    'scaling X and Y and performing the rest of the calculations
    For I = 1 To Npoints
        'scaling X and Y
        Xs(I) = 0.8 * (Xe(I) - Xmid) / (Xmax - Xmid)
        Ys(I) = 0.8 * (Ye(I) - Ymid) / (Ymax - Ymid)
        'output
        Cells(12 + I, 5).Value = Xs(I)
        Cells(12 + I, 6).Value = Ys(I)

    Next I
End Sub
```

This routine is the one which is subject to change dependent on the function being used. Here, the model and the actual data are compared and the SSD is evaluated.

```
Sub Calculations(xp() As Double, _
        sqdev() As Double, _
        SSD As Double)
'Prithwijit Ghoshal
'performs the calculations required to find the SSD between model and data

'variable declaration
   Dim I As Integer

   Dim Ys_Model() As Double
   Dim Y_Model() As Double

   ReDim Ys_Model(1 To Npoints)
   ReDim Y_Model(1 To Npoints)

   ActiveWorkbook.Sheets("Sheet1").Activate
'reinitializing the value of SSD
SSD = 0#
'    'scaling X and Y and performing the rest of the calculations
'
   For I = 1 To Npoints
      Ys_Model(I) = FF(xp(1), xp(2), xp(3), xp(4), Xs(I))

      'converting to unscaled
      Y_Model(I) = Ymid + Ys_Model(I) * (Ymax - Ymid) / 0.8

      sqdev(I) = (Y_Model(I) - Ye(I)) ^ 2
```

```
    SSD = SSD + sqdev(I)
  Next I

  'output section
  For I = 1 To Npoints
    Cells(12 + I, 7).Value = Ys_Model(I)
    Cells(12 + I, 8).Value = Y_Model(I)
    Cells(12 + I, 9).Value = sqdev(I)
  Next I
  Cells(7, 9).Value = SSD



End Sub
```

This is the Steady State Stopping Criterion. It picks out a random set of the deviations (without repetitions in the random selection) and uses the data to calculate an RMS value that is compared to a filtered value of the error to determine steady state.

```
Sub Steady_State(sqdev() As Double, SS As String)
'R Russell Rhinehart
'Modified: Prithwijit Ghoshal
'Steady State Stopping Criterion
' selection with out replacement.
Dim Index() As Integer
ReDim Index(1 To Nrand)
Sum = 0
SS = "N"
Call RANDOM(Index())
```

```
For L = 1 To Nrand
   Sum = Sum + sqdev(Index(L))
   Cells(L, 35) = Index(L)
Next L
Sum = Sqr(Sum)
"Cells(zip + 1, 39) = Sum



Nf = 0.2 * (Xf - Sum) ^ 2 + 0.8 * Nf
Df = 0.2 * (Sum - Sumold) ^ 2 + 0.8 * Df
Sumold = Sum
Xf = 0.2 * Sum + 0.8 * Xf
RStatistic = 1.8 * Nf / Df

"Cells(zip + 1, 40) = Xf
If RStatistic < 0.85 Then
   SS = "Y"
   Cells(nt + 1, 15) = Nf
   Cells(nt + 1, 16) = Df
   Cells(nt + 1, 17) = Xf
   Cells(nt + 1, 13) = RStatistic
   Cells(nt + 1, 14) = SS
End If
Cells(6, 12) = Nf
Cells(7, 12) = Df
Cells(8, 12) = Xf
Cells(4, 12) = RStatistic
Cells(5, 12) = SS

End Sub
```

This is a small program that was created to select random numbers without repetitions and assign them to an array of specified size.

```
Sub RANDOM(A() As Integer)
'Prithwijit Ghoshal
'finds a set of random numbers without repititions
'set stored and transferred in array A()

'variable declaration

   Dim I As Integer    'loop counter
   Dim K As Integer    'loop counter
For I = 1 To Nrand
   A(I) = Int(Rnd() * (Npoints) + 1)
   For K = 1 To I - 1
      If A(K) = A(I) Then
         A(I) = Int(Rnd() * (Npoints) + 1)
         K = 0
      End If
   Next K
Next I

End Sub
```

This is another program used to make the code more generic. This finds the number of data points the program will be required to handle.

```
Sub Find_Points()
'Prithwijit Ghoshal
'finds the number of data points provided for the modeling procedure

   ActiveWorkbook.Sheets("Sheet1").Activate
```

```
    Do While (Cells(13 + Npoints, 3).Value <> "")
       Npoints = Npoints + 1
    Loop

End Sub
```

The next program is used at the end of all the trials. It finds the smallest SSD value among the ones found and reports the corresponding model parameters.

```
Sub FInal_Pick()
'Prithwijit Ghoshal
'picking the lowest of the set and reporting it..
Dim locate As Integer   'location of lowest SSD
Dim Min As Double   'lowest SSD
Dim I As Integer    'loop ocunter
Dim xp(1 To 4) As Double
Dim sqdev(1 To 100) As Double
Dim SSD As Double

Min = 100000#
locate = 0#

'find and locate the minimum..
For I = 1 To 22 'NumTrials
   If Min > Sheet1.Cells(1 + I, 32) Then
      Min = Sheet1.Cells(1 + I, 32)
      locate = I
   End If
Next I

'outputthe result
```

```
Sheet1.Cells(2, 2) = Sheet1.Cells(1 + locate, 19)
Sheet1.Cells(2, 3) = Sheet1.Cells(1 + locate, 20)
Sheet1.Cells(2, 4) = Sheet1.Cells(1 + locate, 21)
Sheet1.Cells(2, 5) = Sheet1.Cells(1 + locate, 22)
For I = 1 To 4
   xp(I) = Sheet1.Cells(2, 1 + I)
Next I
Call Calculations(xp(), sqdev(), SSD)



End Sub
```

Main Program (RRR's Heuristic Optimizer)

The program is based on the algorithm described in Chapter 3.

```
Sub HRO()
'R Russell Rhinehart, Prithwijit Ghoshal
'Heuristic random number based optimizer formulated by RRR
'incorporates the Weakest-Link-in-the-Chain strategy for global optimization
'incorporates Steady State Stopping Criterion
'Modified
'Oct 15, 2007
'Oct 16, 2007

'variable declaration
   Dim Yold As Double
   Dim Y As Double

   Dim SS As String
```

```
    Dim SQRDev() As Double

    Dim Trial_timer As Double
    Dim Total_timer As Double


    Total_timer = Timer
    ActiveWorkbook.Sheets("Sheet1").Activate
N = 4  'decision variables
M = 200  'number of iterations

zip = 1

Call Find_Points

ReDim Xe(1 To Npoints)
ReDim Ye(1 To Npoints)
ReDim Xs(1 To Npoints)
ReDim Ys(1 To Npoints)
ReDim SQRDev(1 To Npoints)

Nrand = Round(Cells(3, 8).Value * Npoints / 100)

Call Initial_Calculations

Expand_Factor = 1.5
Contract_Factor = -0.5 / Expand_Factor


'to run one trial
conf = Cells(1, 9) / 100   '90
```

```
bestfract = Cells(1, 11) / 100      '10
NumTrials = 1
   If conf < 1 And conf > 0 And bestfract < 1 And bestfract > 0 Then NumTrials =
Int(0.5 + Log(1 - conf) / Log(1 - bestfract))
   Cells(5, 2) = NumTrials
''check phase
'numtrials = 1
'input
 For nt = 1 To NumTrials
   Range(Cells(1, 38), Cells(201, 40)).Clear

  'random start using a range of +5 to -5
   'xo(1) = Rnd * 4 - 2

   For K = 1 To N
      xo(K) = Rnd * 8# - 4#
   Next K
      'data echo
      'Cells(2, 2).Value = xo(1)
         For K = 0 To N - 1
            Cells(2, 2 + K) = xo(K + 1)
            'Cells(3, 2 + K) = xo(4 + K)
         Next K

      For I = 1 To N
        X(I) = xo(I)
        dX(I) = 0.1
      Next I
      'Worksheets("Neural Network").Calculate
      Call Calculations(X(), SQRDev(), Yold)
       Cells(7, 9) = Yold
```

```
    Y = Yold

  For J = 1 To M              'limit of 100 iterations
    For I = 1 To N                'N is the number of decision variables
      Cells(1, 1) = J
      Cells(2, 1) = I
      X(I) = xo(I) + dX(I)        'xo(i) is the base point, dx(i) is the proposed
change

      'output
      'Cells(2, 2).Value = X(1)

      For K = 0 To 3
        Cells(2, 2 + K) = X(1 + K)
        'Cells(3, 2 + K) = X(4 + K)
      Next K
      Call Calculations(X(), SQRDev(), Y)
        Cells(7, 9) = Y

      If Y < Yold Then
        xo(I) = X(I)
        dX(I) = dX(I) * Expand_Factor
        Yold = Y

      Else
        X(I) = xo(I)
        dX(I) = Contract_Factor * dX(I)       '0.5 is the contraction factor. You
could use another number
        Call Calculations(X(), SQRDev(), Y)
      End If
      'output
```

```
            'Cells(2, 2).Value = X(1)
            For K = 0 To 3
               Cells(2, 2 + K) = X(K)
               'Cells(3, 2 + K) = X(4 + K)
            Next K
            Cells(zip + 1, 38) = zip



         Next I
         Cells(J + 1, 38) = J
         zip = J
         'introducing the Steady State Stopping Criterion after each set of iterations
          completes
         Call Steady_State(SQRDev(), SS)
            If SS = "Y" Then
               Cells(1 + nt, 18).Value = nt
               For K = 0 To 3
                  Cells(1 + nt, 19 + K).Value = xo(K + 1)
               Next K


               Cells(1 + nt, 32).Value = Yold
               Cells(1 + nt, 33).Value = Timer - Trial_timer
               'time required for each iteration
               'get out of the trial
               GoTo 101
            End If
      Next J

101
Call Calculations(xo(), SQRDev(), Y)
      'zip = 0
```

```
Next nt
Cells(1 + nt, 33) = Timer - Total_timer

Call FInal_Pick

End Sub
```

Main Program: Hooke Jeeves Optimizer

The program is split into two sections. The first is the main program described below, where we have the Best-of-N formula repeating the trials, and consequently reporting the model with the smallest SSD.

```
Sub Hooke_Jeeves()
' Prithwijit Ghoshal
'Modified
'Oct 15, 2007
'Oct 16, 2007

'variable declaration
   Dim Yold As Double
   Dim Y As Double



   Dim Trial_timer As Double
   Dim Total_timer As Double
```

```
    Total_timer = Timer
    ActiveWorkbook.Sheets("Sheet1").Activate
N = 7  'decision variables
M = 200  'number of iterations

zip = 1

Call Find_Points

ReDim Xe(1 To Npoints)
ReDim Ye(1 To Npoints)
ReDim Xs(1 To Npoints)
ReDim Ys(1 To Npoints)
ReDim SQRDev(1 To Npoints)

Nrand = Round(Cells(3, 8).Value * Npoints / 100)

Call Initial_Calculations

Expand_Factor = 1.5
Contract_Factor = -0.5 / Expand_Factor



'to run one trial
conf = Cells(1, 9) / 100   '90
bestfract = Cells(1, 11) / 100      '10
NumTrials = 1
    If conf < 1 And conf > 0 And bestfract < 1 And bestfract > 0 Then NumTrials =
Int(0.5 + Log(1 - conf) / Log(1 - bestfract))
```

```
    Cells(5, 2) = NumTrials
For nt = 1 To NumTrials
    Range(Cells(1, 38), Cells(201, 40)).Clear

   'random start using a range of +5 to -5
    xo(1) = Rnd * 4 - 2

    For K = 1 To 7
       xo(K) = Rnd * 4 - 2#
    Next K
       'data echo
       Cells(2, 2).Value = xo(1)
          For K = 1 To 3
             Cells(2, 2 + K) = xo(1 + K)
             Cells(3, 2 + K) = xo(4 + K)
          Next K
    HookeJeevesD 0.0001, 200, 7, 0.1, True, False, 1, xo(), Y, 1


             Cells(1 + nt, 19).Value = xo(1)
             Cells(1 + nt, 20).Value = xo(2)
             Cells(1 + nt, 24).Value = xo(3)
             Cells(1 + nt, 28).Value = xo(4)
             Cells(1 + nt, 21).Value = xo(5)
             Cells(1 + nt, 25).Value = xo(6)
             Cells(1 + nt, 29).Value = xo(7)
             Cells(1 + nt, 32).Value = Y
             Cells(1 + nt, 33).Value = Timer - Trial_timer
Next nt

End Sub
```

This is the second subroutine which is based on the Hooke Jeeves' algorithm described in Chapter 3.

```
Sub HookeJeevesD(dEpsilon As Double, lngMaxIter As Long, iDim As Integer, _
        dAlpha As Double, bUserPatt As Boolean, bDebug As Boolean, _
        lngFun As Long, aX() As Double, dFXFinal As Double, _
        lngTotFunCall As Long)
'T Judson Wooters,Prithwijit Ghoshal
Dim arrXSolve() As Double
Dim arrXCurr() As Double
Dim arrXPast() As Double
Dim arrXDel() As Double
Dim arrFXSolve(1 To 4) As Double
Dim dFXPast As Double
Dim dMin As Double
Dim iMin As Long
Dim iCount As Integer
Dim lngFunCall As Long
Dim lngPts As Long
Dim lngActPts As Long
Dim bPattern As Boolean
Dim bFoundMin As Boolean
Dim K As Long
Dim J As Long
Dim N As Long
Dim P As Long

bPattern = False
bFoundMin = False
lngTotFunCall = 0
```

```
ReDim arrXCurr(1 To iDim + 7)
ReDim arrXPast(1 To iDim)
ReDim arrXDel(1 To iDim)
ReDim arrXSolve(1 To 4, 1 To iDim)

For K = 1 To iDim
   arrXSolve(2, K) = aX(K)
Next K

lngPts = lngPts + 1
lngActPts = lngActPts + 1

For K = 1 To iDim
   arrXCurr(K) = arrXSolve(2, K)
Next K

Call Calculations(arrXCurr(), SQRDev(), arrFXSolve(2))

For K = 0 To lngMaxIter
   zip = K + 1
   Cells(zip + 1, 38) = zip

   If Not bPattern Then
      For N = 1 To iDim
         arrXSolve(1, N) = arrXSolve(2, N)
      Next N
      arrFXSolve(1) = arrFXSolve(2)
   End If
```

```
    iCount = 0

    For J = 1 To iDim
        DiscExplore arrXSolve(), arrFXSolve, dAlpha, iDim, J, lngFun, lngFunCall,
_
                arrXDel()


        dMin = arrFXSolve(2)
        iMin = 2
        For N = 2 To 4
            If dMin > arrFXSolve(N) Then
                dMin = arrFXSolve(N)
                iMin = N
            End If
        Next N

        If iMin <> 2 Then
            For N = 1 To iDim
                arrXSolve(2, N) = arrXSolve(iMin, N)
            Next N
            arrFXSolve(2) = arrFXSolve(iMin)

        End If
        If J < iDim Then
            lngFunCall = 0
        End If
    Next J
    'steady state check
    Call Steady_State(SQRDev(), SS)
    If SS = "Y" Then
```

```
    bFoundMin = True
    Exit For
End If

If arrFXSolve(1) = arrFXSolve(2) Then
    If dAlpha < dEpsilon Then
        bFoundMin = True
        Exit For
    End If
    dAlpha = dAlpha / 2

    If bPattern Then
        bPattern = False
    End If
ElseIf (arrFXSolve(1) - arrFXSolve(2)) < dEpsilon And _
        (arrFXSolve(1) - arrFXSolve(2)) > 0 Then
    If dAlpha < dEpsilon Then
        bFoundMin = True
        Exit For
    Else
        dAlpha = dAlpha / 2
    End If
ElseIf arrFXSolve(1) - arrFXSolve(2) > 0 And bUserPatt Then
    For N = 1 To iDim
        arrXPast(N) = arrXSolve(1, N)
    Next N
    dFXPast = arrFXSolve(1)
    For N = 1 To iDim
        arrXSolve(1, N) = arrXSolve(2, N)
    Next N
    arrFXSolve(1) = arrFXSolve(2)
```

```
        'pattern jump
        For N = 1 To iDim
            arrXSolve(2, N) = arrXSolve(2, N) + (arrXSolve(2, N) - arrXPast(N))
        Next N
        For N = 1 To iDim
            arrXCurr(N) = arrXSolve(2, N)
        Next N
        Call Calculations(arrXCurr(), SQRDev(), arrFXSolve(2))

        bPattern = True
    ElseIf bUserPatt Then
        For N = 1 To iDim
            arrXSolve(2, N) = arrXSolve(1, N)
        Next N
        arrFXSolve(2) = arrFXSolve(1)
        bPattern = False
    End If

    lngTotFunCall = lngTotFunCall + lngFunCall
    lngFunCall = 0

Next K

If Not bFoundMin And bDebug Then
    For N = 1 To iDim
        Sheet2.Cells(6, 9 + N).Value = arrXSolve(2, N)
    Next N
    Sheet2.Cells(7, 9).Value = "May not have found minimum"
End If

For K = 1 To iDim
```

```
   aX(K) = arrXSolve(2, K)
Next K
dFXFinal = arrFXSolve(2)

End Sub
Sub DiscExplore(arrXSolve() As Double, arrFXSolve() As Double, dAlpha As
Double, _
            iDim As Integer, iIndex As Long, lngFun As Long, lngFunCall As
Long, _
            arrXDel() As Double)
'T Judson Wooters,Prithwiit Ghoshal
Dim arrXCurr() As Double
Dim K As Long
Dim J As Long

ReDim arrXCurr(1 To iDim + 7)

For K = 3 To 4
   For J = 1 To iDim
      arrXSolve(K, J) = arrXSolve(2, J)
   Next J
Next K

arrXSolve(3, iIndex) = arrXSolve(2, iIndex) + dAlpha
For K = 1 To iDim
   arrXCurr(K) = arrXSolve(3, K)
Next K
Call Calculations(arrXCurr(), SQRDev(), arrFXSolve(3))
arrXSolve(4, iIndex) = arrXSolve(2, iIndex) - dAlpha
For K = 1 To iDim
   arrXCurr(K) = arrXSolve(4, K)
```

```
Next K
Call Calculations(arrXCurr(), SQRDev(), arrFXSolve(4))

End Sub
```

Main Program: Broydon-Fletcher-Goldfarb-Shanno (BFGS)

This is again split into a number of sections. The first is using the Best-of-N formula to repeat trials, and consequently finds the lowest SSD for the solution.

```
Sub B_F_G_S()
'Prithwijit Ghoshal
'Modified
'Feb 24, 2008
'

'variable declaration
   Dim Yold As Double
   Dim Y As Double



   Dim Trial_timer As Double
   Dim Total_timer As Double



   Total_timer = Timer
   ActiveWorkbook.Sheets("Sheet1").Activate
N = 4  'decision variables
```

```
M = 200  'number of iterations

zip = 1

Call Find_Points

ReDim Xe(1 To Npoints)
ReDim Ye(1 To Npoints)
ReDim Xs(1 To Npoints)
ReDim Ys(1 To Npoints)
ReDim SQRDev(1 To Npoints)

Nrand = Round(Cells(3, 8).Value * Npoints / 100)

Call Initial_Calculations

Expand_Factor = 1.5
Contract_Factor = -0.5 / Expand_Factor



'to run one trial
conf = Cells(1, 9) / 100   '90
bestfract = Cells(1, 11) / 100      '10
NumTrials = 1
   If conf < 1 And conf > 0 And bestfract < 1 And bestfract > 0 Then NumTrials =
Int(0.5 + Log(1 - conf) / Log(1 - bestfract))
   Cells(5, 2) = NumTrials
   NumTrials = 1
For nt = 1 To NumTrials
   Range(Cells(1, 38), Cells(201, 40)).Clear
```

```
    'initializations
    Nf = 0#
    Df = 0#
    Xf = 0#

   'random start using a range of +5 to -5
    xo(1) = Rnd * 4 - 2

    For K = 1 To 7
       xo(K) = Rnd * 4 - 2#
    Next K
       'data echo
       Cells(2, 2).Value = xo(1)
          For K = 1 To 3
             Cells(2, 2 + K) = xo(1 + K)
             Cells(3, 2 + K) = xo(4 + K)
          Next K
    'HookeJeevesD 0.0001, 200, 4, 0.1, True, False, 1, xo(), Y, 1
    BFGS 0.0001, 200, 13, 1, 0.0001, False, True, 1, 100, 0.001, xo(), Y, 1
             Cells(1 + nt, 19).Value = xo(1)
             Cells(1 + nt, 20).Value = xo(2)
             Cells(1 + nt, 24).Value = xo(3)
             Cells(1 + nt, 28).Value = xo(4)
             Cells(1 + nt, 32).Value = Y
             Cells(1 + nt, 33).Value = Timer - Trial_timer
Next nt
Call Final_Pick
End Sub
```

This is used to find the derivatives of the required objective function based on a forward difference.

```
Function fF_Der1FD(aX() As Double, lngIndex As Long, iDim As Integer, _
                   dStep As Double, lngFun As Long, lngFunCall As Long, _
                   aXDel() As Double) As Double
' T. Judson Wooters 29-MAR-2007
‘Modified: Prithwijit Ghoshal
' Function used to determine derivative using 4 data points and central difference
' Inputs:       aX()        array of current iteration x locations
'               lngIndex    determines which element to base derivative on
'               iDim        # of dimensions
'               dStep       stepsize for finite difference
'               lngFun      function number corresponding to function in Newton
Interface Module
'               lngDerCall  keeps track of derivative calls
'               aXDel()     dummy variable, used with other programs which access
fFX function
' Output:       fF_Der1FD   Derivative


Dim aFX(1 To 5) As Double          ' function evaluation for all 5 points
Dim aXCurr() As Double             ' temporary location for x locations
Dim K As Integer                   ' counter variable

ReDim aXCurr(1 To iDim)

For K = 1 To iDim
   aXCurr(K) = aX(K)               ' Load temporary x's
Next K
Call Calculations(aX(), SQRDev(), aFX(3))
```

```
'   X(Curr+1)
aXCurr(lngIndex) = aX(lngIndex) + dStep
Call Calculations(aXCurr(), SQRDev(), aFX(4))
End Function
```

This is the main routine which is based on the BFGS algorithm described in Chapter 3.

```
Sub BFGS(dEpsilon As Double, lngMaxIter As Long, iDim As Integer, _
        dLambda As Double, dStep As Double, bDebug As Boolean, _
        bLineNR As Boolean, lngFun As Long, lngMaxIterNR As Long, _
        dEpsilonNR As Double, aX() As Double, dFXFinal As Double, _
        lngTotFunCall As Long)
' T. Judson Wooters, 29-MAR-2007
' Main BFGS (Quasi-Newton) program to find minimum, inputs explained in
RunBFGS sub

Dim aXNext() As Double          ' new set of x values based on iteration
Dim aXPrev() As Double          ' old set of x values used to reset algorithm
Dim aI() As Double              ' identity matrix
Dim aSearch() As Double         ' search direction vector
Dim aBDelInv() As Double        ' inverse B difference matrix
Dim aBNegInv() As Double        ' negative inverse B matrix
Dim aBInv() As Double           ' inverse B matrix
Dim aF_Der1() As Double         ' vector of 1st derivatives
Dim aF_Der1Prev() As Double     ' previous vector of 1st derivatives
Dim aPosDef() As Double         ' intermediate array in determining if B inverse is
pos def
Dim aXDel() As Double           ' used in line searching by Newton module (stores
search dir)
Dim aXDiff() As Double          ' difference in x between iterations
```

```
Dim aGDiff() As Double          ' difference in 1st derivatives between iterations
Dim aInt1() As Double           ' intermediate array in BFGS
Dim aInt2() As Double           ' intermediate array in BFGS
Dim aInt3() As Double           ' intermediate array in BFGS
Dim aInt4() As Double           ' intermediate array in BFGS
Dim aInt5() As Double           ' intermediate array in BFGS
Dim dFX As Double               ' function evaluation
Dim dFXNext As Double           ' function evaluation based on next x values
Dim dLambdaIn As Double         ' default step for multiplication with search
direction
Dim dPosDef As Double           ' intermediate value in determining if B inverse is
pos def
Dim dSearch As Double           ' intermediate value in determining if search
direction is improving
Dim dInt4 As Double             ' intermediate value in BFGS
Dim dInt5_1 As Double           ' intermediate value in BFGS
Dim dInt5_2 As Double           ' intermediate value in BFGS
Dim dMagGrad As Double          ' magnitude of 1st derivatives, used with
stopping criteria
Dim lngFunCall As Long          ' number of function calls per iteration
Dim lngDerCall As Long          ' number of derivative calls per iteration
Dim bCauchy As Boolean          ' True = current iteration is steepest decent
Dim bPosDef As Boolean          ' True = positive definate
Dim bMinFound As Boolean        ' True = algorithm finished meeting the
stopping criteria
Dim K As Long                   ' counting variable
Dim J As Long                   ' counting variable
Dim N As Long                   ' counting variable

' ----- INITIALIZE VARIABLES -----
dLambdaIn = dLambda
```

```
bCauchy = True
bMinFound = False
dFXFinal = 0
lngTotFunCall = 0

ReDim aXNext(1 To iDim)
ReDim aI(1 To iDim, 1 To iDim)
ReDim aBInv(1 To iDim, 1 To iDim)
ReDim aBNegInv(1 To iDim, 1 To iDim)
ReDim aBDelInv(1 To iDim, 1 To iDim)
ReDim aF_Der1(1 To iDim)
ReDim aPosDef(1 To iDim)
ReDim aSearch(1 To iDim)
ReDim aXDel(1 To iDim)
ReDim aXDiff(1 To iDim)
ReDim aGDiff(1 To iDim)
ReDim aF_Der1Prev(1 To iDim)
ReDim aXPrev(1 To iDim)
ReDim aInt1(1 To iDim)
ReDim aInt2(1 To iDim, 1 To iDim)
ReDim aInt3(1 To iDim, 1 To iDim)
ReDim aInt4(1 To iDim, 1 To iDim)
ReDim aInt5(1 To iDim, 1 To iDim)

For K = 1 To iDim
   For J = 1 To iDim
      If K = J Then
         aI(K, J) = 1
      Else
         aI(K, J) = 0
      End If
```

```
    Next J
Next K

Call Calculations(aX(), SQRDev(), dFX)



' ----- BEGIN BFGS ALGORITHM -----
For K = 0 To lngMaxIter
    zip = K + 1
    Cells(zip + 1, 38) = zip

    ' Find array of first derivatives
    For J = 1 To iDim
        aF_Der1(J) = fF_Der1FD(aX(), J, iDim, dStep, lngFun, lngFunCall,
aXDel())
    Next J

    ' If steepest decent iteration, B inverse is the identity matrix
    If bCauchy Then
        For J = 1 To iDim
            For N = 1 To iDim
                aBInv(J, N) = aI(J, N)
            Next N
        Next J
    ' If BFGS step, find B inverse using update calculations
    Else
        For J = 1 To iDim
            aXDiff(J) = aX(J) - aXPrev(J)
            aGDiff(J) = aF_Der1(J) - aF_Der1Prev(J)
        Next J
```

```
    MatVect aGDiff(), aBInv(), aInt1(), iDim
    For J = 1 To iDim
       aInt1(J) = aXDiff(J) - aInt1(J)
    Next J
    VectVectT aInt1(), aXDiff(), aInt2(), iDim
    VectVectT aXDiff(), aInt1(), aInt3(), iDim
    dInt4 = fVectTVect(aGDiff(), aXDiff(), iDim)
    If dInt4 = 0 Then Exit For
    For J = 1 To iDim
       For N = 1 To iDim
          aInt4(J, N) = (aInt2(J, N) + aInt3(J, N)) / dInt4
       Next N
    Next J
    dInt5_1 = fVectTVect(aInt1(), aGDiff(), iDim)
    dInt5_2 = (fVectTVect(aGDiff(), aXDiff(), iDim)) ^ 2
    VectVectT aXDiff(), aXDiff(), aInt5(), iDim
    For J = 1 To iDim
       For N = 1 To iDim
          aInt5(J, N) = (dInt5_1 / dInt5_2) * aInt5(J, N)
          aBDelInv(J, N) = aInt4(J, N) - aInt5(J, N)
          aBInv(J, N) = aBInv(J, N) + aBDelInv(J, N)
       Next N
    Next J
End If

' Negative of B inverse matrix
For J = 1 To iDim
    For N = 1 To iDim
       aBNegInv(J, N) = -aBInv(J, N)
    Next N
Next J
```

```
' Determine if search direction is improving
MatVect aF_Der1(), aBNegInv(), aSearch(), iDim
dSearch = fVectTVect(aF_Der1(), aSearch(), iDim)

' Determine if B inverse is positive definate
VectTMat aX(), aBInv(), aPosDef(), iDim
dPosDef = fVectTVect(aPosDef(), aX(), iDim)

If dPosDef >= 0 And dSearch < 0 Then
   bPosDef = True
Else
   bPosDef = False
End If

' If matrix is positive definate, find new x values
If bPosDef Then
   ' Find best 10% of dlambda to make f(x) decrease with 95% confidence
   ' interval using RRR paper on neural network training
   For N = 1 To 30
      If bLineNR Then
         ' newton raphson line search
         Newton dLambda, dEpsilonNR, lngMaxIterNR, "Min," dStep, aX(), _
               aSearch(), False, -1, iDim, lngFun, lngFunCall
      End If
      For J = 1 To iDim
         aXNext(J) = aX(J) + dLambda * aSearch(J)
      Next J
      'dFXNext = fFX(0, aXNext(), aXDel(), 0, lngFun, iDim, lngFunCall)
      Call Calculations(aXNext(), SQRDev(), dFXNext)
```

```
        If dFXNext < dFX Then Exit For
        If bLineNR Then
            dLambda = 10 ^ (RandomNum(-6, 1))
        End If
    Next N
    bCauchy = False
' If matrix not positive definate, repeat this iteration using steepest decent
Else
    bCauchy = True
End If



' If the current iteration is positive definate (calculated new x values),
' check for termination
'steady state check
If bPosDef Then
    Call Steady_State_1(SQRDev(), SS)
    If SS = "Y" Then
        bFoundMin = True
        Exit For
    End If
End If

' Replace current step with next step
For J = 1 To iDim
    aF_Der1Prev(J) = aF_Der1(J)
    aXPrev(J) = aX(J)
    aX(J) = aXNext(J)
Next J
dFX = dFXNext
```

```
    lngTotFunCall = lngTotFunCall + lngFunCall
    lngFunCall = 0
    lngDerCall = 0

    ' Reset BFGS variables
    For J = 1 To iDim
        aInt1(J) = 0
        For N = 1 To iDim
            aInt2(J, N) = 0
            aInt3(J, N) = 0
            aInt4(J, N) = 0
            aInt5(J, N) = 0
            aBDelInv(J, N) = 0
        Next N
    Next J
Next K

' If the minimum was not found by meeting stopping criteria, raise a flag
If Not bMinFound And bDebug Then
    MsgBox "Min may not have been found"
    For J = 1 To iDim
        'shtBFGS.Cells(6, 9 + J).Value = aX(J)
    Next J
    'shtBFGS.Cells(6, 16).Value = dFX
End If

dFXFinal = dFXNext

End Sub
```

The following is a set of routines created for matrix manipulations.

```
Sub VectTMat(aVect() As Double, aMat() As Double, aVectOut() As Double,
iDim As Integer)
' T Judson Wooters, 29-MAR-2007
' Subprogram to multiply a transposed vector with a matrix
' Inputs:       aVect()     transposed vector
'               aMat()      matrix
'               iDim        number of dimensions
' Outputs:      aVectOut()  resulting vector

Dim dSum As Double
Dim J As Long
Dim N As Long

   For J = 1 To iDim
      dSum = 0
      For N = 1 To iDim
         dSum = dSum + aVect(N) * aMat(N, J)
      Next N
      aVectOut(J) = dSum
   Next J

End Sub

Sub MatVect(aVect() As Double, aMat() As Double, aVectOut() As Double,
iDim As Integer)
' T Judson Wooters, 29-MAR-2007
' Subprogram to multiply a matrix with a vector
' Inputs:       aVect()     vector
'               aMat()      matrix
```

```
'              iDim        number of dimensions
' Outputs:      aVectOut()  resulting vector

Dim dSum As Double
Dim J As Long
Dim N As Long

   For J = 1 To iDim
      dSum = 0
      For N = 1 To iDim
         dSum = dSum + aVect(N) * aMat(J, N)
      Next N
      aVectOut(J) = dSum
   Next J

End Sub

Sub VectVectT(aVect1() As Double, aVect2() As Double, aMatOut() As Double,
iDim As Integer)
' T Judson Wooters, 29-MAR-2007
' Subprogram to multiply a vector with a transposed vector
' Inputs:       aVect1()    vector
'               aVect2()    transposed vector
'               iDim        number of dimensions
' Outputs:      aMatOut()   resulting matrix

Dim dSum As Double
Dim J As Long
Dim N As Long

   For J = 1 To iDim
```

```
        For N = 1 To iDim
            aMatOut(J, N) = aVect1(J) * aVect2(N)
        Next N
    Next J

End Sub

Function fVectTVect(aVect1() As Double, aVect2() As Double, iDim As Integer)
As Double
' T Judson Wooters, 29-MAR-2007
' Function to multiply a transposed vector with a vector
' Inputs:       aVect1()    transposed vector
'               aVect2()    vector
'               iDim        number of dimensions
' Outputs:      fVectTVect   resulting scaler

Dim J As Long

    For J = 1 To iDim
        fVectTVect = fVectTVect + aVect1(J) * aVect2(J)
    Next J

End Function
```

This is a modified Subroutine that looks at a complete selection of the deviations and not a random one.

```
Sub Steady_State_1(sqdev() As Double, SS As String)
‘Prithwijit Ghoshal
‘modified SS criteria to accommodate the BFGS routine. Without random picks
' selection with out replacement.
```

```
Dim Index() As Integer
ReDim Index(1 To Nrand)
Sum = 0
SS = "N"
Call RANDOM(Index())
For L = 1 To Nrand
        Sum = Sum + sqdev(Index(L))
        Cells(L, 35) = Index(L)
Next L
Sum = Sqr(Sum)
Cells(zip + 1, 39) = Sum
Nf = 0.2 * (Xf - Sum) ^ 2 + 0.8 * Nf
Df = 0.2 * (Sum - Sumold) ^ 2 + 0.8 * Df
Sumold = Sum
Xf = 0.2 * Sum + 0.8 * Xf
RStatistic = 1.8 * Nf / Df

Cells(zip + 1, 40) = Xf
If RStatistic < 1 Then
   SS = "Y"
   Cells(nt + 1, 15) = Nf
   Cells(nt + 1, 16) = Df
   Cells(nt + 1, 17) = Xf
   Cells(nt + 1, 13) = RStatistic
   Cells(nt + 1, 14) = SS
End If
Cells(6, 12) = Nf
Cells(7, 12) = Df
Cells(8, 12) = Xf
Cells(4, 12) = RStatistic
Cells(5, 12) = SS
```

```
End Sub
```

Functions:

This can be used by changing the **Calculations** subroutine to use the function "AA" and by changing the number of decision variables involved in the required optimizer.

The first function is the Neural Network created by Dr R. Russell Rhinehart.

```
Function AA(X, B, w11, w12, w13, w14, w21, w22, w23, w24, w1, w2, w3, w4)
'   R. Russell Rhinehart   Neural Network Demo Program
'   School of Chemical Engineering, Oklahoma State university
'   rrr@okstate.edu
'   Last revised November 2005
'
'   Program computes the NN output for a 1-input-1-output NN with input bias,
'   three hidden layer neurons, and one output neuron.  NN transfer function
'   is bi-polar sigmoidal.  Training is by EXCEL Solver add in.
'

Dim WIH(2, 4)   'Weights on hidden layer
Dim WHO(4)      'weights on output layer
Dim N(4)        'Neuron Output

'
'   Get values of weights from spreadsheet
'

WIH(1, 1) = w11
WIH(1, 2) = w12
```

```
WIH(1, 3) = w13
WIH(1, 4) = w14
WIH(2, 1) = w21
WIH(2, 2) = w22
WIH(2, 3) = w23
WIH(2, 4) = w24
WHO(1) = w1
WHO(2) = w2
WHO(3) = w3
WHO(4) = w4

For J = 1 To 2      'for each of the hidden neurons
   z = B * WIH(1, J) + X * WIH(2, J)                  'calculate weighted input
   N(J) = (Exp(z) - Exp(-z)) / (Exp(z) + Exp(-z))      'calculate neuron output
Next J

   z = N(1) * WHO(1) + N(2) * WHO(2) + N(3) * WHO(3) + N(4) * WHO(4)
'calculate weighted input for output neuron
   AA = (Exp(z) - Exp(-z)) / (Exp(z) + Exp(-z))        'calculate neuron output for
output neuron

End Function
```

This is a simple third order polynomial function

```
Function FF(A As Double, B As Double, C As Double, D As Double, X As
Double) As Double
'Prithwijit Ghoshal
'November 07, 2007
'Polynomial function

FF = A + B * X + C * X ^ 2 + D * X ^ 3

End Function
```

APPENDIX D

CASE INDEX FOR STEADY STATE VS EXCESSIVE ITERATIONS ANALYSIS

Case	Model Used	Optimizer	Steady State (Y/N)	Data Set (A/B)
A1	Polynomial	RRR	Y	A
A2	Polynomial	RRR	N	A
B1	Polynomial	HJ	Y	A
B2	Polynomial	HJ	N	A
C1	Polynomial	BFGS	Y	A
C2	Polynomial	BFGS	N	A
D1	Neural Network	RRR	Y	A
D2	Neural Network	RRR	N	A
E1	Neural Network	HJ	Y	A
E2	Neural Network	HJ	N	A
F1	Neural Network	BFGS	Y	A
F2	Neural Network	BFGS	N	A
G1	Polynomial	RRR	Y	B
G2	Polynomial	RRR	N	B
H1	Polynomial	HJ	Y	B
H2	Polynomial	HJ	N	B
I1	Polynomial	BFGS	Y	B
I2	Polynomial	BFGS	N	B
J1	Neural Network	RRR	Y	B
J2	Neural Network	RRR	N	B
K1	Neural Network	HJ	Y	B
K2	Neural Network	HJ	N	B
L1	Neural Network	BFGS	Y	B
L2	Neural Network	BFGS	N	B

VITA

Prithwijit Ghoshal

Candidate for the Degree of

Master of Science

Thesis: STUDY OF AN INITIALIZATION METHOD AND STOPPING CRITERIA FOR NONLINEAR OPTIMIZATION

Major Field: Chemical Engineering

Biographical:

Personal Data: Born on January 04, 1985
to Timir Baran Ghoshal and Bharati Ghoshal

Education: Graduated High School from the Assembly of God Church School, Kolkata in 2002. Completed Bachelor of Engineering in Chemical Engineering from Siddaganga Institute of Technology, affiliated to Visveswaraya Technological University, Karnataka India. Completed the requirements for the Master of Science or Arts in Chemical Engineering at Oklahoma State University, Stillwater, Oklahoma in July, 2008

Experience: Worked as Teaching Assistant for ENGR 1412 (Computer Programming in Visual Basic for Applications) from August 2006 to May 2008). Worked as teaching Assistant for CHE 5723 (Optimization Applications) from August 2007 to December 2007. Working as Research Assistant from August 2006 to present

Professional Memberships:

Golden Key Honor Society	2007 to present
American Chemical Society	June 2008 to present

Name: Prithwijit Ghoshal Date of Degree: July, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: STUDY OF AN INITIALIZATION METHOD AND STOPPING CRITERIA FOR NONLINEAR OPTIMIZATION

Pages in Study: 169 Candidate for the Degree of Master of Science

Major Field: Chemical Engineering

Scope and Method of Study: An initialization method previously used in neural network training is combined with a novel steady state stopping criterion and used in the empirical modeling optimization of various applications and modeling objectives. The effectiveness of both the initialization method and the stopping criterion are tested using direct and indirect optimization algorithms, on contrived data, and on experimental data from a two-phase flow system. The study attempts to create global optimization logic for nonlinear modeling.

Findings and Conclusions: The global optimization logic developed for empirical modeling optimization is scale independent, requires no a priori knowledge to stop a trial, and is found to be robust while handling noisy data. The logic defines the number of random starts that will find a user defined "best" percentage of possible model solutions from process data with a desired confidence. The logic can use a variety of nonlinear local optimizers, and can be incorporated in future optimization software as a modeling tool.

ADVISER'S APPROVAL: Dr R. Russell Rhinehart