Non-linear least squares estimation with an iteratively updated data set
by Marc Leonard Keppler

A thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering
Montana State University

Abstract:
This thesis presents a nonlinear least squares approach that iteratively updates the parameter estimate while stepping through the data set. The approach presented proceeds by first computing the parameter estimate for a reduced portion of the data set and then adding consecutive data samples and recomputing the parameter estimate for the new data set. This process repeats until a final parameter estimate for the entire data set is obtained. This approach was developed to estimate the parameters of systems that can be modeled by differential equations. Two routines, referred to collectively as the continuation methods, that utilize this approach are examined. One method steps through the data space using an arbitrary schedule supplied by the user. The other routine computes the number of consecutive data samples to add on each iteration. The continuation methods were validated by comparing their performance to an implementation of Levenburg-Marquardt. Experimental results obtained from a wide variety of functions and dynamic data obtained from a practical fuel cell power system demonstrated that the continuation methods have significantly larger regions of convergence than the traditional Levenburg-Marquardt approach.

NON-LINEAR LEAST SQUARES ESTIMATION WITH AN ITERATIVELY UPDATED DATA SET

by

Marc Leonard Keppler

A thesis submitted in partial fulfillment
of the requirements for the Degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

September 2003

APPROVAL

of a thesis submitted by

Marc Leonard Keppler

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Dr. Steven R. Shaw _____ 9-16-2003
(Signature) Date

Approved for the Department of Electrical and Computer Engineering

Dr. Jim Peterson _____ 9/18/2003
(Signature) Date

Approved for the College of Graduate Studies

Dr. Bruce McLeod _____ 9-23-03
(Signature) Date

STATEMENT OF PERMISSION OF USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Marie L Kepple

Date 9/12/2003

ACKNOWLEDGMENTS

I would like to thank Professor Steven Shaw for his support, contributions, and patience. Professor Shaw's input and advice over the last few years are greatly appreciated. Professor Shaw was a motivating and inspiring advisor.

TABLE OF CONTENTS

LIST OF TABLES vii

LIST OF FIGURES viii

GLOSSARY x

ABSTRACT xi

1 INTRODUCTION 1

2 LEAST SQUARES ESTIMATION 3

LINEAR LEAST SQUARES 3
NON-LINEAR LEAST SQUARES 4
AN EXAMPLE OF LOCAL MINIMA 6

3 THE CONTINUATION METHODS 9

THE APPROACH 9
THE SIMPLE CONTINUATION METHOD 11
Updating the Parameter Estimate 12
Updating the Data Set 13
THE ADAPTIVE CONTINUATION METHOD 14
Updating the Parameter Estimate 15
Updating the Data Set 16
RELATION TO THE KALMAN FILTER 19
The Kalman Filter 19
The Extended Kalman Filter 20

4 VALIDATIONS WITH SIMULATED DATA SETS 22

EXPERIMENTAL PROCEDURES 22
Test Functions 22
Testing the Performance of the Continuation Methods 24
Testing the Robustness of each Method 24
Testing the SCM 25
Testing the ACM 26
VALIDATION RESULTS FOR THE SINGLE SINE WAVE FUNCTIONS 26
Two Parameter Sine Wave with a Fixed Initial Guess 28
Two Parameter Sinewave with Fixed System Parameters 33

TABLE OF CONTENTS – CONTINUED

Four Parameter Sine Wave with a Fixed Initial Guess 33
Four Parameter Sine Wave with Fixed System Parameters 36
VALIDATION RESULTS FOR THE SUM OF TWO SINE WAVES FUNCTION . . . 38
VALIDATION RESULTS FOR THE SUM AND DIFFERENCE OF SINUSOIDS . . . 42
VALIDATION RESULTS FOR THE BOX THREE DIMENSIONAL FUNCTION . . . 44

5 A PRACTICAL IMPLEMENTATION ON A FUEL CELL 47

DYNAMIC FUEL CELL MODEL 47
EXPERIMENTAL SETUP 50
EXPERIMENTAL RESULTS 52

6 CONCLUSIONS 56

THE SIMPLE CONTINUATION METHOD 56
THE ADAPTIVE CONTINUATION METHOD 56
FUTURE RESEARCH 57

REFERENCES 59

LIST OF REFERENCES 60

APPENDICES 63

APPENDIX A: OCTAVE CODE FOR THE SCM 64
APPENDIX B: OCTAVE CODE FOR THE ACM 68
APPENDIX C: OCTAVE CODE FOR LM 76
APPENDIX D: SHARED SUBROUTINES 84
APPENDIX E: UPDATING THE LM PARAMETER 93

LIST OF TABLES

Table		Page
4.1	Validation results for test function (4.1) with a fixed initial guess. . .	28
4.2	Validation results for test function (4.1) with fixed system parameters.	33
4.3	Validation results for test function (4.2) with a fixed initial guess. . .	38
4.4	Validation results for test function (4.2) with fixed system parameters.	38
4.5	Validation results for test function (4.3) with fixed system parameters.	40
4.6	Validation results for test function (4.4) with a fixed initial guess. . .	42
4.7	Validation results for the Box 3-Dimensional function (4.5)	45
5.1	Validation results for the dynamic fuel cell model (5.9).	53

LIST OF FIGURES

Figure		Page
2.1	An illustration of the relationship between δ_{GN} and δ_{LM}.	5
2.2	Observation set for the single parameter sine wave (2.17) used to illustrate the problem of local minima.	7
2.3	Loss function for (2.17).	7
3.1	Contour plot of the loss function of (2.17). The colors on the contour map indicate the magnitude of the residual norm of the loss function. In order of increasing magnitude the colors are; blue, green, yellow, and red.	10
3.2	Contour plot demonstrating how the SCM updates the parameter estimate as it steps through the data space with $\Delta_N = 20$.	14
3.3	Contour plot demonstrating how the SCM updates the parameter estimate as it steps through the data space with $\Delta_N = 50$.	15
3.4	Contour plot demonstrating how an the ACM updates the parameter estimate while adaptively updating the data set.	18
4.1	The convergence of the SCM as a function of Δ_N when fitting equation (4.1), the initial guess was fixed, and the actual parameters were generated randomly.	29
4.2	Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.1), the initial guess was fixed, and the actual parameters were generated randomly. . . .	30
4.3	Scatter plot of the 2000 actual system parameters used for the experiments with equation (4.1) and a fixed initial guess.	31
4.4	Scatter plot of the actual system parameters for which the SCM did not converge when fitting (4.1), the initial guess was fixed, and the actual parameters were generated randomly.	31
4.5	Scatter plot of the actual parameters for which LM did not converge when fitting (4.1), the initial guess was fixed, and the actual parameters were generated randomly.	32
4.6	Convergence of the SCM as a function of Δ_N when fitting equation (4.1), the actual parameters were fixed, and the initial guesses were generated randomly.	34

LIST OF FIGURES – CONTINUED

Figure Page

4.7 Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.1), the actual parameters were fixed, and the initial guesses were generated randomly. 34

4.8 Scatter plot of the 2000 initial guesses generated for the experiments with equation (4.1) and fixed system parameters. 35

4.9 Scatter plot of the initial guesses for which the SCM did not converge when fitting (4.1), the actual parameters were fixed, and the initial guesses were generated randomly. 35

4.10 Scatter plot of the initial guesses for which LM did not converge when fitting (4.1), the actual parameters were fixed, and the initial guesses were generated randomly. 36

4.11 Convergence of the SCM as a function of Δ_N when fitting equation (4.2), the initial guess was fixed, and the actual parameters were generated randomly. 37

4.12 Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.2), the initial guess was fixed, and the system parameters were generated randomly. . . . 37

4.13 Convergence of the SCM as a function of Δ_N when fitting equation (4.2), the system parameters were fixed, and the initial guesses were generated randomly. 39

4.14 Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.2), the actual parameters were fixed, and the initial guesses were generated randomly. 39

4.15 Convergence of the SCM as a function of Δ_N when fitting equation (4.3), the system parameters were fixed, and the initial guesses were generated randomly. 41

4.16 Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.3), the system parameters were fixed, and the initial guesses were generated randomly. 41

4.17 Convergence of the SCM as a function of Δ_N when fitting equation (4.4), the initial guess was fixed, and the system parameters generated randomly. 43

LIST OF FIGURES – CONTINUED

Figure		Page
4.18	Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.4), the initial guess was fixed, and the system parameters were generated randomly. . . .	43
5.1	Circuit analogy for the temperature response of a fuel cell.	49
5.2	Location of K-type thermocouple in SR-12 stack a) and location of thermocouple in the fuel cell cartridge b).	49
5.3	Schematic of Experimental Setup.	51
5.4	Measured output current of the SR-12 PEM fuel cell used to model the output voltage.	55
5.5	Predicted and actual voltage response of the SR-12. The dark line is the response predicted by the SCM.	55

GLOSSARY

local minimum	Minimum of a function on a closed interval
global minimum	Minimum of the entire function
residual	Difference between the observed data and estimated data.
region of convergence	Range of initial parameter estimates for which the routine converges to the actual value.
iteration	One complete cycle through the continuation methods with a fixed number of data points.
loss function	A mathematical expression that computes how the parameter estimate changes the residual.
convergence	Percentage of total tests for which the routine converged to the actual parameters.
test problem	The problem of estimating the parameters of a single data set using a single initial guess.
trust region	Region of the loss function that is not expected to contain a local minima.
sensible heat	Heat energy stored in a substance as a result of an increase in its temperature.
eps	Machine precision
$\vec{N}$	A vector that completely defines the steps taken in data space by the simple continuation method.
Δ_N	Size of the step taken in the data space
N_k	Number of data samples evaluated by the continuation methods on the k'th iteration
N_{max}	Maximum number of samples in a measured data set
$\|\| * \|\|$	The l_2 vector norm of the quantity $*$

ABSTRACT

This thesis presents a nonlinear least squares approach that iteratively updates the parameter estimate while stepping through the data set. The approach presented proceeds by first computing the parameter estimate for a reduced portion of the data set and then adding consecutive data samples and recomputing the parameter estimate for the new data set. This process repeats until a final parameter estimate for the entire data set is obtained. This approach was developed to estimate the parameters of systems that can be modeled by differential equations. Two routines, referred to collectively as the continuation methods, that utilize this approach are examined. One method steps through the data space using an arbitrary schedule supplied by the user. The other routine computes the number of consecutive data samples to add on each iteration. The continuation methods were validated by comparing their performance to an implementation of Levenburg-Marquardt. Experimental results obtained from a wide variety of functions and dynamic data obtained from a practical fuel cell power system demonstrated that the continuation methods have significantly larger regions of convergence than the traditional Levenburg-Marquardt approach.

INTRODUCTION

Least squares estimation is a common solution to the problem of determining unknown system parameters from a measured data set. Least squares fitting estimates the actual parameters of a modeled system by minimizing the sum of the squared errors between the estimated and measured data. Two major categories of least squares fitting are linear least squares and nonlinear least squares (NLS) estimation. Linear least squares can be used when the function relating the system model is linear from the parameters to the output. Otherwise, NLS is used [1].

NLS estimation routines generally update an initial parameter estimate $\hat{\theta}_0$ iteratively to obtain the final estimate $\hat{\theta}$. These routines generally converge only for a limited range of initial guesses. The continuation methods presented in this thesis were developed to increase the region of convergence (ROC) provided by least squares estimation. Development of the continuation methods was motivated by the desire to estimate the parameters of an induction motor from transient stator current measurements. A general approach for estimating the parameters of these motors based on increasing amounts of data was proposed in [2] and the continuation methods presented in this thesis are an extension of this research. The continuation methods are based on the NLS approach and were developed to estimate the parameters of systems that can be modeled by differential equations.

Systems that can be modeled by differential equations represent a broad class of useful problems. The continuation methods developed in this thesis are beneficial whenever an accurate initial guess for a system that can be modeled by differential equations is not available or easy to obtain. Two interesting specializations of this problem class are noninvasive system diagnostics and identification of fuel cell model parameters. One example of noninvasive system diagnostics is the Nonintrusive Load Monitor (NILM) [3, 4, 5, 6, 7, 8]. This device is installed at a centralized location in a

power distribution network and monitors electrical transients to determine the condition of the individual loads in the network. This requires the NILM to identify a wide range of systems without an accurate initial guess for every transient. Another interesting specialization of this problem class is the identification of fuel cell parameters. Much research has focused on the development of steady-state [9, 10, 11, 12, 13] and dynamic [14, 15, 16, 17, 18] models for fuel cells. System identification using these models can improve the design of individual fuel cells and fuel cell power systems.

This thesis begins with an overview of least squares estimation in Chapter 2. The description in Chapter 2 includes general least squares estimation along with the Gauss-Newton and Levenberg-Marquardt methods. Chapter 3 introduces the continuation methods and examines the computations involved with each method. The continuation methods are compared to a traditional Levenberg-Marquardt routine using simulated data sets in Chapter 4 and using dynamic data obtained from a practical fuel cell power system in Chapter 5. Chapter 6 summarizes the experimental results and provides suggestions for future research.

LEAST SQUARES ESTIMATION

This chapter summarizes the mathematical computations involved with least squares estimation and the Gauss-Newton and Levenberg-Marquardt iterations. The overview provided is intended to facilitate a description of the continuation methods and the problem of local minima from a mathematical point of view. This review is not exhaustive. More rigorous treatments of these topics are given in [19, 20, 21].

The primary purpose of least squares is to determine an m dimensional parameter vector $\hat{\theta}$ that minimizes the error between an n dimensional measurement vector y and an n dimensional estimate vector $\hat{y}$ given an p dimensional input vector u and a function relating these quantities

$$\hat{y} = f(\hat{\theta}, u). \tag{2.1}$$

The error measure for least squares is the loss function

$$V(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{2}r'r, \tag{2.2}$$

where r is the n dimensional residual vector defined by

$$r = y - \hat{y}. \tag{2.3}$$

The loss function (2.2) produces a maximum likelihood estimate when the data has normal, identically distributed, and independent errors [20].

Linear Least Squares

Linear least squares is applied to systems that are linear in the parameters. These systems can be expressed

$$y(u) = \sum_{i=1}^{m}\theta_i f_i(u), \tag{2.4}$$

where f_i represents the i'th component of f in (2.1) except f is no longer a function of θ. Linear systems can also be expressed using the linear matrix equation

$$y = A\theta, \tag{2.5}$$

where y is linearly related to the parameters θ. If the system is linear in the parameters, it is possible to obtain the final least squares estimate in a single iteration. If $m = n$ then the solution can be found by inverting A as follows

$$\hat{\theta} = A^{-1}y \tag{2.6}$$

If $n > m$ then the pseudo-inverse must be utilized. When $n > m$ the linear least squares problem is often solved using the singular value decomposition (SVD) [20, 22].

Non-Linear Least Squares

If a system is not linear in the parameters nonlinear least squares must be applied. Nonlinear least squares is applied to systems that are expressed

$$y(u) = f(\theta, u), \tag{2.7}$$

where the parameters θ are inside the nonlinear function f. These systems require an iterative approach to minimize the loss function in (2.2). Many methods exist that solve the NLS problem. However, the Levenberg-Marquardt method [23, 1, 24] and its variations [25, 26] are most commonly used [20].

The Levenberg-Marquardt method is an adaptation of the Gauss-Newton method. The Gauss-Newton iteration is

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + \delta_{GN}^{(k)}, \tag{2.8}$$

$$\delta_{GN} = (J^T J)^{-1} J^T r, \tag{2.9}$$

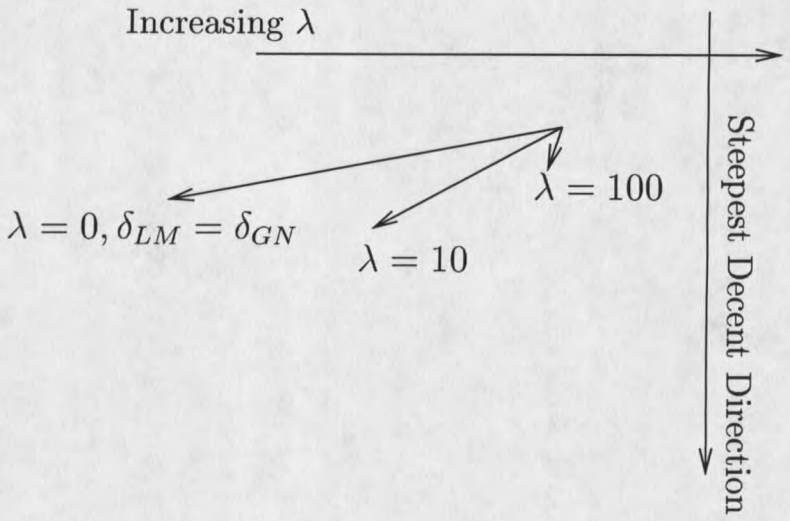

Fig. 2.1. An illustration of the relationship between δ_{GN} and δ_{LM}.

where J is the Jacobian matrix [19] with elements

$$J_{i,j} = -\frac{\partial r_i}{\partial \theta_j}\bigg|_{\hat{\theta}}. \tag{2.10}$$

The Levenberg-Marquardt step δ_{LM} follows δ_{GN} with an additional weighting term that allows the step to interpolate between the steepest decent direction and δ_{GN}. The Levenberg-Marquardt step is given by

$$\delta_{LM} = (J^T J + \lambda D^T D)^{-1} J^T r, \tag{2.11}$$

where λ and sometimes D are changed on each iteration [27]. The term λ is often referred to as the Levenberg-Marquardt parameter and D is a weighting matrix that accounts for the scaling of the problem. Implementations of Levenberg-Marquardt vary primarily in how D and λ are computed [24]. Notice that δ_{LM} approaches δ_{GN} as $\lambda \to 0$ and δ_{LM} reduces to zero and approaches the steepest-descent direction as λ increases with $D = I$ [20]. The relationship between δ_{GN} and δ_{LM} is illustrated in Fig. 2.1.

The disadvantage of most NLS estimation algorithms, and Gauss-Newton and Levenberg-Marquardt in particular, is that they evaluate the loss function using local

information. Thus, they converge to the minimum closest to the initial parameter estimate. This is a significant problem because the loss function of a nonlinear system can have many local minima. Routines that evaluate the loss function using local information are susceptible to local minima because the derivative of the loss function is zero at both global and local extrema. The derivative of the loss function at any extrema is

$$\nabla V(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(\frac{\partial}{\partial\theta_1} \quad \frac{\partial}{\partial\theta_2} \quad \cdots \quad \frac{\partial}{\partial\theta_m}\right) r_i^2 = 0. \tag{2.12}$$

Expanding this derivative yields

$$\nabla V(\theta) = \sum_{i=1}^{n}\left(\frac{\partial r_i}{\partial\theta_1} \quad \frac{\partial r_i}{\partial\theta_2} \quad \cdots \quad \frac{\partial r_i}{\partial\theta_m}\right) r_i. \tag{2.13}$$

Noticing that the term

$$\left(\frac{\partial r_i}{\partial\theta_1} \quad \frac{\partial r_i}{\partial\theta_2} \quad \cdots \quad \frac{\partial r_i}{\partial\theta_m}\right) \tag{2.14}$$

is a row of the Jacobian matrix defined in (2.10), it follows that [24]

$$\nabla V(\theta) = \sum_{i=1}^{n}\left(\frac{\partial r_i}{\partial\theta_1} \quad \frac{\partial r_i}{\partial\theta_2} \quad \cdots \quad \frac{\partial r_i}{\partial\theta_m}\right) r_i = -(J^T r)^T = 0, \tag{2.15}$$

$$\delta_{LM} = (J^T J + \lambda D^T D)^{-1} J^T r = 0. \tag{2.16}$$

The Levenberg-Marquardt step δ_{LM} is zero at a local minimum and the iteration does not improve the parameter estimate.

An Example of Local Minima

Estimating the frequency of a sine wave illustrates the problem of local minima. Traditional NLS routines that utilize local information generally require an accurate initial guess to estimate the frequency of a sine wave. Suppose the observed data is given by the function

$$y = \sin(6t), \tag{2.17}$$

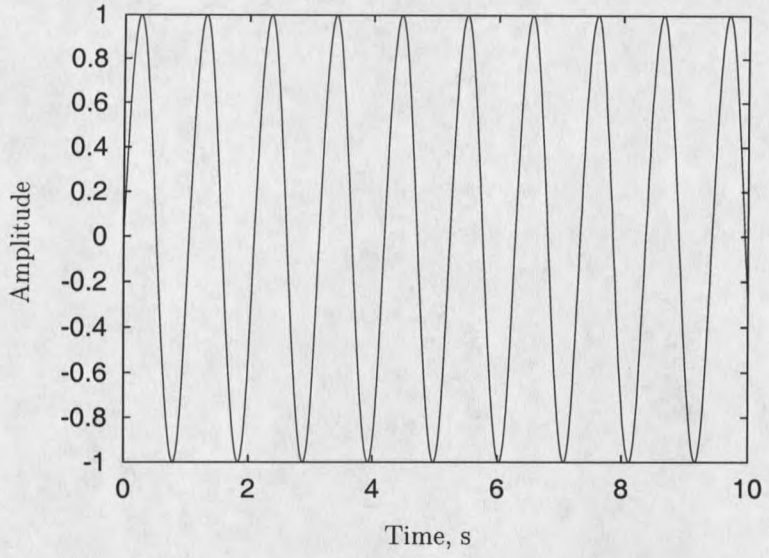

Fig. 2.2. Observation set for the single parameter sine wave (2.17) used to illustrate the problem of local minima.

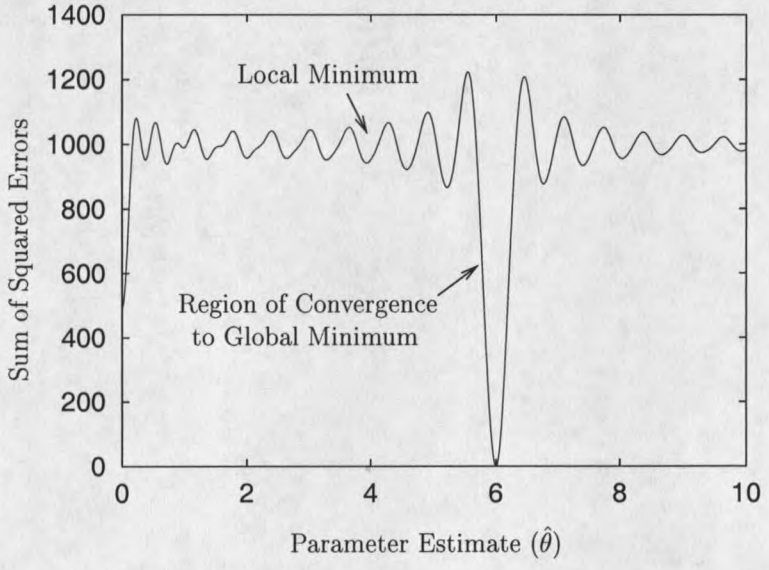

Fig. 2.3. Loss function for (2.17).

where t is a time vector on the interval $[0, 10]$ seconds with a sample rate of 100 samples/sec (see Fig. 2.2). The loss function for this observation set is shown in Fig. 2.3. Many local minima exist in this loss function. Traditional least squares approaches will only converge to the actual frequency if the initial guess is within the concave-up region directly surrounding the global minimum. For this example, traditional least squares routines will converge to the actual parameter estimate only if the initial guess is within approximately 7.5% of the actual value over the interval $[0, 10]$. This is a very small region of convergence. The disadvantage of NLS routines that use local information is that their convergence completely depends on the initial guess and the shape of the loss function.

THE CONTINUATION METHODS

The Approach

Before the specific computations of the continuation methods are presented it is necessary to understand the fundamental approach of the routines. The continuation methods avoid local minima and increase the ROC by stepping iteratively through the data space and solving the NLS problem on each iteration. One iteration through the continuation methods corresponds to one minimization of the loss function (2.2) for a particular number of data points N_k. The continuation methods update the data set by adding consecutive data samples. That is, they begin with the first data point d_1 and add data points following the scheme

$$(d_1 \quad d_2 \quad d_3 \quad \cdots \quad d_{N_k} \quad \cdots \quad d_{N_{k+1}} \quad \cdots \quad d_{N_{max}})$$

where d_i represents the i'th data point, N_k is the number of data points evaluated on the k'th iteration and N_{max} is the maximum number of data points in the observation set. The general approach taken by the continuation methods is

1) Obtain the initial data points to evaluate $N_{k=0}$
2) Minimize the loss function over N_k
3) Increase the data set to N_{k+1}
4) Repeat 2) and 3) until $N_k = N_{max}$ and
   the convergence criteria is met.

Both continuation methods follow the general approach outlined above. However, they differ in how they obtain the initial number of data points $N_{k=0}$, how the data space is updated on each iteration, and how the parameter estimate is computed. The specific differences between these routines are examined in the following sections. For

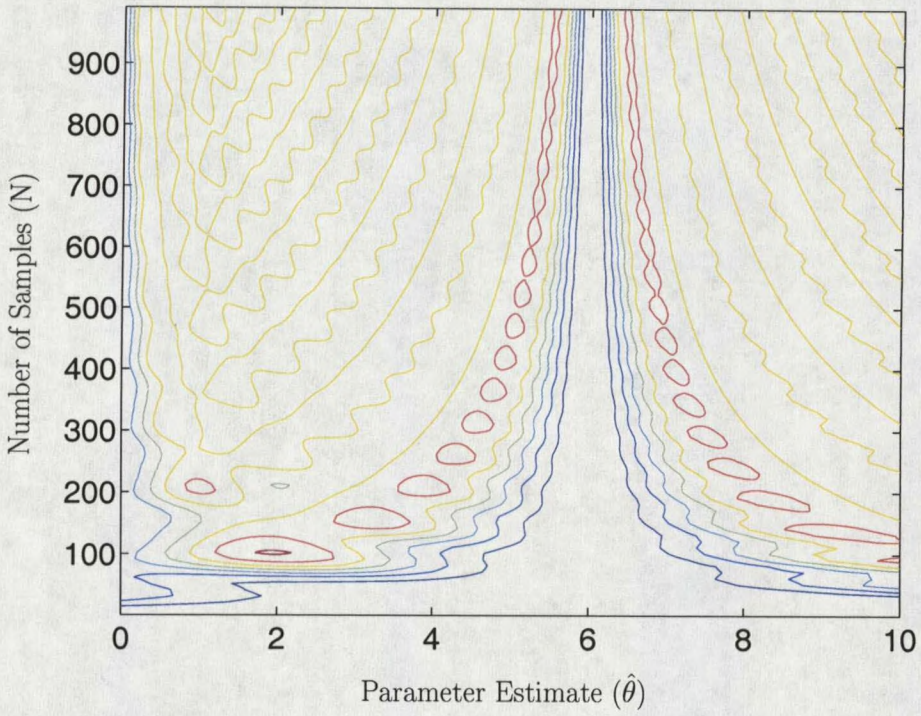

Fig. 3.1. Contour plot of the loss function of (2.17). The colors on the contour map indicate the magnitude of the residual norm of the loss function. In order of increasing magnitude the colors are; blue, green, yellow, and red.

now it is sufficient to notice that the iterative approach taken by the continuation methods can reduce the nonlinearity of the function for small N_k. This reduction in nonlinearity decreases the number of local minima in the loss function and increases the likelihood of converging to the global minimum.

The sine wave of (2.17) demonstrates how iteratively updating the data set can reduce the number of local minima and increase the ROC. The measured data for (2.17) is shown in Fig. 2.2. A contour plot of the loss function for (2.17) versus the number of data points N is shown in Fig. 3.1. As the number of data points decreases the number of local minima decrease as well. Reducing the number of local minima in the loss function for small N provides a large ROC for the function evaluated at small N. Once the continuation methods converge to the global minimum for small N, they

will successfully converge to the global minimum of the entire data set if they stay within a region of quadratic convergence and the loss function is smoothly deforming. The constraint of a smoothly deforming loss function implies the continuation methods should be applied to systems for which the loss function continuously evolves as the number of data samples increases. The continuation methods can be applied to these systems because the loss function for an increased number of samples can be reasonably determined from the loss function for the previous number of samples. Systems that can be modeled by differential equations fall into this category because they are solved by integrating continuous functions. For these systems, a change in the parameters on the order of the error in the parameters generally results in monotonic behavior of the loss function. The continuation methods use this behavior of the loss function to increase the ROC.

The Simple Continuation Method

Each iteration through the simple continuation method (SCM) involves two main tasks; 1) update the parameter estimate $\hat{\theta}$ and 2) update the data set N_k. The SCM updates the parameter estimate using a weighted Gauss-Newton method and updates the data set by following the schedule defined in a user supplied vector $\vec{N}$. The data set was updated using an arbitrary schedule contained in $\vec{N}$ to facilitate a study of how the step in the data space affected the convergence of the SCM. A weighted Gauss-Newton iteration was used to update the parameter estimate because, if applied correctly, iteratively updating the data set keeps the routine in a region of quadratic convergence on all but the first iteration. When the convergence is quadratic, the Gauss-Newton approach is as good as any other NLS estimation technique. The

following sections detail the computations used by the SCM to update the parameter estimate and the data space. The Octave code for the SCM is given in Appendix A.

Updating the Parameter Estimate

The SCM updates the parameter estimate using a weighted Gauss-Newton approach. The SCM follows the traditional Gauss-Newton step with the addition of a weighting matrix P. The weighting matrix P contains information about the variance in the fitted parameters and is computed

$$P = P_0 + J^T J, \tag{3.1}$$

where J is the Jacobian of the current data set evaluated at the previous parameter estimate and P_0 is the initial weighting matrix. The SCM updates the parameter estimate $\hat{\theta}$ with a step

$$\delta_{\hat{\theta}} = -P^{-1} J^T r, \tag{3.2}$$

where J is the Jacobian and r is the residual. The step in parameter space is the Gauss-Newton step δ_{GN} multiplied by the variances of the fitted parameters θ. The initial weighting matrix P_0 in (3.1) provides the expected variance of the initial guess and is supplied by the user. If the user expects the parameters to have a variance of approximately $p\%$, then P_0 should be defined

$$P_0 = \begin{pmatrix} \frac{1}{p*\hat{\theta}_1} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{p*\hat{\theta}_2} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{p*\hat{\theta}_3} & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{p*\hat{\theta}_n} \end{pmatrix}, \tag{3.3}$$

where $\hat{\theta}_i$ is the i'th parameter in the initial guess $\hat{\theta}$. Equations (3.2) and (3.3) assure that the step in parameter space is proportional to the expected variance in the estimate.

Updating the Data Set

A user supplied vector $\vec{N}$ defines the steps taken by the SCM through the data space. The vector $\vec{N}$ must completely define the data set and supply the number of samples to evaluate on each iteration. For example, if the data set consists of a total of N_{max} data samples, $\vec{N}$ may be defined

$$\vec{N} = (20 \quad 40 \quad 60 \quad ... \quad N_{max})$$

although the size of the step in the data space does not need to be constant. However, for this thesis, the SCM updated the data set with a constant step size. The constant step size Δ_N defined the number of data points evaluated on the first iteration and how many data points were added on subsequent iterations. For an arbitrary Δ_N the schedule taken through the data space was

$$\vec{N} = (\Delta_N \quad 2\Delta_N \quad 3\Delta_N \quad \cdots \quad N_{max})$$

Specifically, if the measured data consists of 1000 samples and $\Delta_N = 50$ then the continuation method would step iteratively through the data space according to the schedule

$$\vec{N} = (50 \quad 100 \quad 150 \quad \cdots \quad 950 \quad 1000).$$

This simple approach allows the user to adjust the step in the data space Δ_N based on the complexity of the problem and allowed a study of the affect of Δ_N on the convergence of the SCM.

Illustrations of how the SCM updates the parameter estimate as it steps iteratively through the data space are given in Figs. 3.2 and 3.3. These contour plots demonstrate

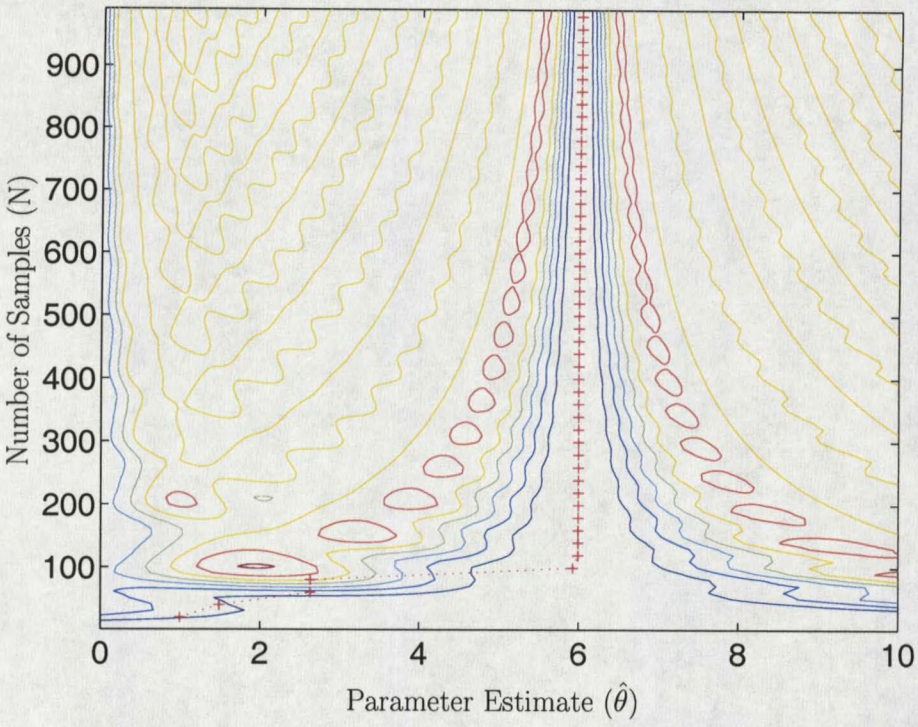

Fig. 3.2. Contour plot demonstrating how the SCM updates the parameter estimate as it steps through the data space with $\Delta_N = 20$.

how the SCM estimated the parameters of the single parameter sine wave function (2.17) for varying Δ_N. Figures 3.2 and 3.3 illustrate how the SCM updates the parameter estimate as it steps through the data space with $\Delta_N = 20$ and $\Delta_N = 50$, respectively. When $\Delta_N = 50$, the SCM almost falls into a local minimum (Fig. 3.3). This implies that the ROC is maximized by keeping Δ_N small.

The Adaptive Continuation Method

The adaptive continuation method (ACM) updates the parameter estimate with a Levenberg-Marquardt step and updates the data set using a *trust region* approach. The hyper-ellipsoidal *trust region* limits the step in the data space so the ratio between

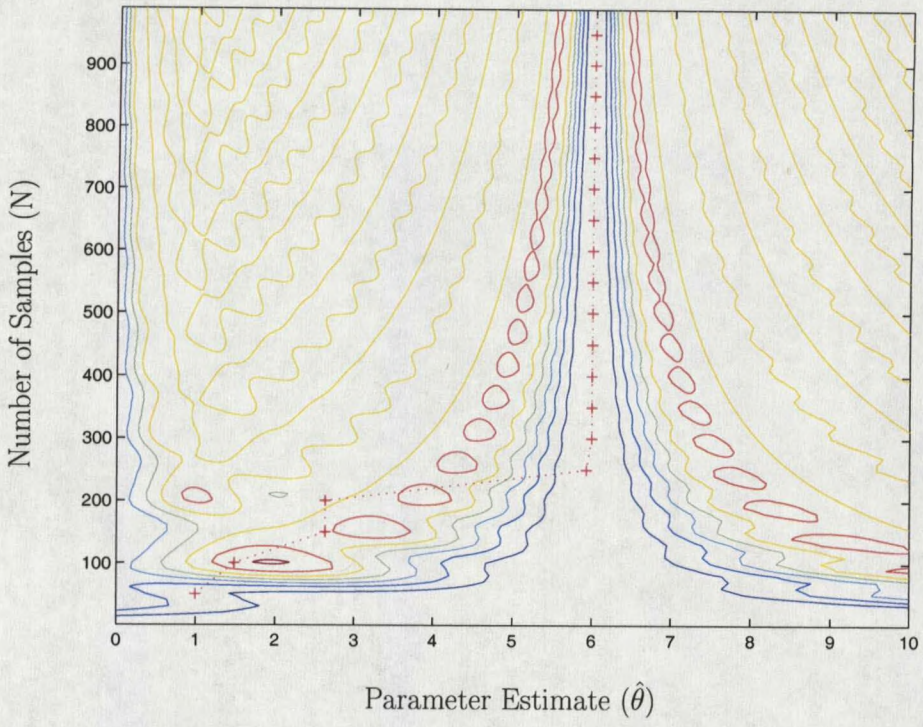

Fig. 3.3. Contour plot demonstrating how the SCM updates the parameter estimate as it steps through the data space with $\Delta_N = 50$.

the step in parameter space and the standard error of the parameters is less than the variance of the measurement noise. The *trust region* was generated using the standard error of the parameters and the measurement noise because they provide an indication of the precision of the estimated parameters and measured data, respectively. The parameter estimate was computed using a Levenberg-Marquardt iteration because it is the most common NLS estimation approach used today [20].

Updating the Parameter Estimate

The ACM updates the parameter estimate via a Levenberg-Marquardt step that is based on the method proposed by Jorge Moré [27] and its corresponding implementation for the MINPACK project. The approach utilized by the ACM differs from

the Levenberg-Marquardt method derived for the MINPACK project by taking the SVD, instead of the QR decomposition, of the Jacobian. The SVD was used because it provided access to the singular values of the Jacobian which improves performance on poorly conditioned problems [22].

The ACM computes a step in the parameter space $\delta_{\hat{\theta}}$ using a traditional LM step. The step $\delta_{\hat{\theta}}$ is computed such that

$$\sqrt{\alpha} \quad < \quad \text{eps}, \tag{3.4}$$

$$\| D\delta_{\hat{\theta}} \| \quad \leq \quad \eta, \tag{3.5}$$

or

$$\sqrt{\alpha} \quad > \quad \text{eps}, \tag{3.6}$$

$$| \| D\delta_{\hat{\theta}} \| - \delta | \quad \leq \quad \sigma\eta, \tag{3.7}$$

where η is the step bound, σ is the desired relative error in the step bound, D is a weighting matrix, and *eps* is the machine precision. When $N < N_{max}$ the data set is updated whenever a parameter estimate $\hat{\theta}$ leads the ACM into a region of quadratic convergence. When $N = N_{max}$ the parameter estimate $\hat{\theta}$ is updated until one of the convergence criteria is satisfied. For a detailed description of the computations used by the ACM to update the parameter estimate see Appendix B and Appendix E.

Updating the Data Set

Once a parameter estimate that results in quadratic convergence has been computed for a given N_k, the data set is updated. The initial number of data points was computed

$$N_{k=0} \quad = \quad \max(5 \quad .05 * N_{max}). \tag{3.8}$$

Equation (3.8) was used because it significantly reduced the complexity of the loss function and produced excellent results when applied to the functions used for vali-

dation. This $N_{k=0}$ is not optimal for all problems. Determining an optimal $N_{k=0}$ for all possible functions is outside the scope of this thesis. However, relatively simple solutions to this problem exist. One solution is to allow the user to supply $N_{k=0}$ based on their knowledge of the function. Another solution can be derived from the observation that the elements in the rows of the Jacobian matrix of a linear system are equal. This solution can be implemented by computing the Jacobian for a relatively large portion of the data set, say 10-20%, and continually reducing the size of the data set until the elements in the rows of the Jacobian are equal within some tolerance. This assures that the ACM sufficiently reduces the nonlinearity of the data set.

On subsequent iterations the number of samples added is determined using recursive least squares (RLS) [19], the standard errors of the parameters, and the measurement noise. The standard errors of the parameters σ_θ are given by the diagonal elements of the matrix

$$C = (J^T J)^{-1} \tag{3.9}$$

$$\sigma_\theta(i) = \sqrt{C_{ii}} \tag{3.10}$$

where J is the Jacobian and $\sigma_\theta(i)$ is the standard error of the i'th parameter [1]. The ACM chooses a step in the data space that keeps the step in the parameter space $\delta\hat{\theta}$ within a hyper-ellipsoidal *trust region* that is generated from the standard error of the parameters and the measurement noise. The *trust region* is defined by

$$\| C_\sigma^{-1}(\hat{\theta}_{N_{k+1}} - \hat{\theta}_{N_k}) \| < \sigma_N^2, \tag{3.11}$$

$$\| \frac{C_\sigma^{-1}}{\sigma_N}(\hat{\theta}_{N_{k+1}} - \hat{\theta}_{N_k}) \| < 1, \tag{3.12}$$

where $\hat{\theta}_{N_{k+1}}$ is the RLS parameter estimate for the updated number of samples, $\hat{\theta}_{N_k}$ is the parameter estimate for the number of samples evaluated on the k'th (last) iteration, σ_N^2 is the variance of the measurement noise, and C_σ is a diagonal matrix

that contains the standard errors of the parameters defined in (3.10). The ACM updates the data space one sample at a time until the ratio of the difference between the recursive least squares (RLS) solution of the parameter estimate at the updated number of samples $\hat{\theta}_{N_{k+1}}$ and the parameter estimate at the original number of samples $\hat{\theta}_{N_k}$ to the standard errors of the parameters exceeds the measurement noise (3.11).

An illustration of how the ACM updates the parameter estimate as it adaptively updates the data set is given in Fig. 3.4. The contour plot demonstrates how the

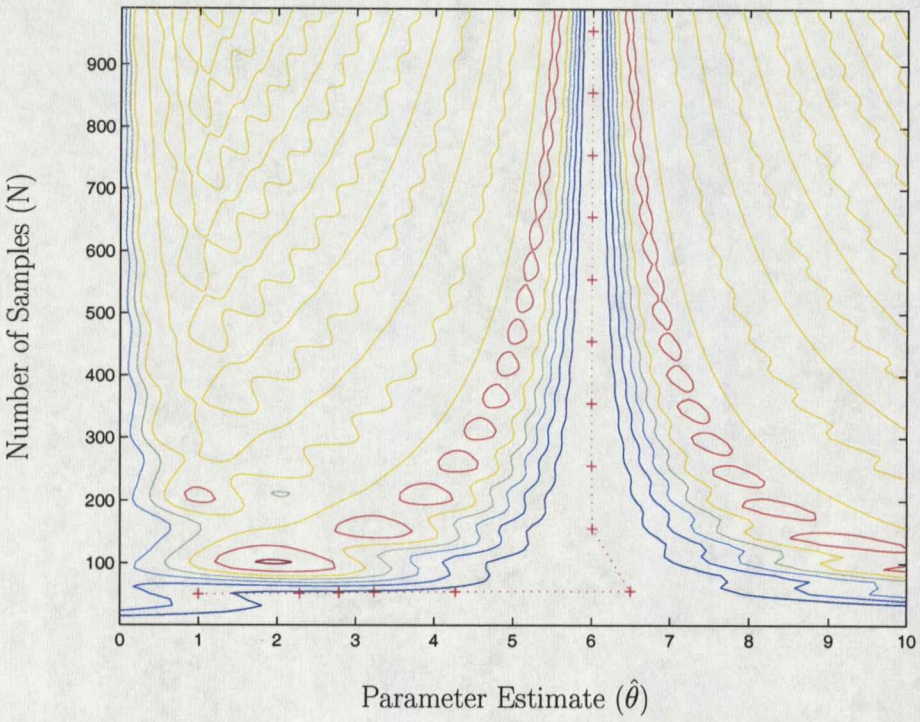

Fig. 3.4. Contour plot demonstrating how an the ACM updates the parameter estimate while adaptively updating the data set.

ACM estimates the parameters of the single parameter sine wave function (2.17). Figure 3.4 shows that the ACM takes small steps in the data space until it is close to the correct parameter estimate. Once the ACM computes an accurate parameter estimate it then adds large amounts of data on subsequent iterations. This adaptive

approach should provide a large ROC and reduce the number of iterations required for convergence.

Relation to the Kalman Filter

The iterative approach taken by the continuation methods may remind the reader of the extended Kalman filter and its application to recursive least squares. However, there is a subtle yet significant difference between the extended Kalman filter and the continuation methods. The extended Kalman filter is closely related to the Kalman filter. The difference between these filters is the extended Kalman filter is applied to nonlinear problems and the Kalman filter is applied to linear problems. The following is a simplistic explanation of the Kalman and extended Kalman filters. The explanation provided is intended to give a basic understanding of the extended Kalman filter and highlight the difference between it and the continuation methods. More detailed information on the Kalman and extended Kalman filters is given in [28, 21, 20]

The Kalman Filter

The Kalman filter is generally used to estimate the n dimensional state vector x of a linear controlled process [28, 24, 19]. Formally, the linear system estimated by the Kalman filter is given by

$$x_{i+1} = \Phi x_i + \Gamma u_i + w_i, \tag{3.13}$$

with an m dimensional linear output equation

$$z_i = H x_i + v_i, \tag{3.14}$$

where x_{i+1} is the current state, x_i is the previous state, u_i is an optional control input, z_i is the measurement vector, w_i is the process noise, and v_i is the measurement

noise [28]. For the simplified explanation provided in thesis v_i, u_i, and w_i are assumed zero. The linear system then becomes

$$x_{i+1} = \Phi x_i, \tag{3.15}$$

with an m dimensional linear output equation

$$z_i = H x_i. \tag{3.16}$$

A recursive solution of the state x of the system described in (3.15) and (3.16) is

$$\hat{x}_{i+1} = \Phi_i \hat{x}_i + K_i(z_i - H_i \hat{x}_i), \tag{3.17}$$

where K_i is the *Kalman gain* and the difference $z_i - H_i\hat{x}_i$ is known as the *innovation* or *residual* [28]. The error between the state estimates is

$$\epsilon_i = x_i - \hat{x}_i, \tag{3.18}$$

where x_i is the actual and unknown value of the state at time i. The *Kalman gain* matrix K in (3.17) is chosen to minimize the error covariance of the state estimate error (3.18) given by

$$P_i = E\left[\epsilon_i \epsilon_i^T\right], \tag{3.19}$$

where E is the expected value of the error covariance.

The Extended Kalman Filter

The application of the Kalman filter to nonlinear systems is known as the extended Kalman filter. The extended Kalman filter is a Kalman filter that linearizes around the current mean and covariance [28]. The linearizations performed by the extended Kalman filter are generally accomplished through use of the Jacobian matrix. Consider the simple nonlinear system with an n dimensional state vector x

$$x_{i+1} = f(x_i), \tag{3.20}$$

and an m dimensional linear output equation

$$z_i = h(x_i), \tag{3.21}$$

where both f and h are nonlinear functions. The equations that estimate the state x and measurement z in the extended Kalman filter are the same as those used in standard Kalman filter. The only difference is that the matrices Φ and H in the extended Kalman filter are Jacobian matrices that contain the partial derivatives of f and h with respect to the state x, respectively.

The extended Kalman filter is reminiscent of the continuation methods because they both update estimates using increasing amounts of data. The extended Kalman filter estimates the next state using information obtained from the previous state which accumulates data as the filter progresses. The continuation methods begin by evaluating a small portion of the data set and then successively increasing the size of the data set until they obtain a final parameter estimate for the entire data set. However, the iterative solution used by the extended Kalman filter differs from the continuation methods. Iterative solution of the extended Kalman filter does not evaluate the linearizations obtained through use of the Jacobian at the correct operating points determined by the most recent estimates [24]. The extended Kalman filter is equivalent to the continuation methods only if all of the previous states $i = 0, 1, 2, \cdots, k-1, k$ are used to estimate the next state $i = k+1$.

VALIDATIONS WITH SIMULATED DATA SETS

Experimental Procedures

Adequately testing a parameter estimation routine is a difficult task. The testing procedures used to validate the continuation methods were developed following the testing guidelines proposed by Moré [29]. The guidelines in [29] advocate that testing optimization software should focus on reliability and robustness, should include a large variety of test functions, and the methods compared should have similar convergence criteria. The validation tests used for this thesis were designed to meet these goals. The performance of the continuation methods were compared to an implementation of Levenberg-Marquardt. To ensure that the convergence results were comparable, the convergence criteria used by each of these methods were similar. The robustness of the methods was tested using thousands of different parameters for each test function. The test functions used for validation encompassed a wide range of complexity and functional behavior. The following sections detail how the testing procedures addressed the guidelines in [29].

Test Functions

The test functions used to validate the continuation methods were generated using sines and exponentials because the responses of these functions are similar to that of many differential equations. Furthermore, identification of sine waves is important in many test and measurement systems. The number and variety of the test functions used for validation were not as wide as in [29]. This is reasonable considering that the continuation methods were developed to identify a specific problem class and not

the general NLS problem. The test functions used to generate simulated data sets for validation were:

1) Two Parameter Sine Wave Function

$$\begin{aligned} y(t) &= x_1 \sin(x_2 t) \\ x_1 &= \text{amplitude} \\ x_2 &= \text{frequency} \end{aligned} \tag{4.1}$$

2) Four Parameter Sine Wave Function

$$\begin{aligned} y(t) &= x_1 \sin(x_2 t + x_3) + x_4 \\ x_1 &= \text{amplitude} \\ x_2 &= \text{frequency} \\ x_3 &= \text{phase} \\ x_4 &= \text{offset} \end{aligned} \tag{4.2}$$

3) Sum of Two Sine Waves Function

$$\begin{aligned} y(t) &= x_1 \sin(x_2 t) + x_3 \sin(x_4 t) \\ x_1, x_3 &= \text{amplitude} \\ x_2, x_4 &= \text{frequency} \end{aligned} \tag{4.3}$$

4) Sum and Difference of Sinusoids

$$\begin{aligned} y(t) &= x_1 \sin(x_2 t) + x_3 \sin(x_4 t) - x_5 \cos(x_6 t) \\ x_1, x_3, x_5 &= \text{amplitude} \\ x_2, x_4, x_6 &= \text{frequency} \end{aligned} \tag{4.4}$$

5) The Box Three Dimensional Function

$$f_i(x) = e^{-t_i x_1} - e^{-t_i x_2} - x_3(e^{-t_i} - e^{-10t_i}) \tag{4.5}$$

$$\begin{aligned} t_i &= .1i \\ 0 \leq \quad i &\leq 20 \end{aligned}$$

The Box Three Dimensional function (4.5) was one of the standard test functions included in [29] and was originally published in [30]. With the exception of the Box Three Dimensional Function, t is a time vector sampled at 100 samples/sec on the interval $[0, 10]$ seconds. Thus, each of the test functions in 1-4 have $N_{max} = 1000$.

Testing the Performance of the Continuation Methods

The continuation methods were validated by comparing their performance to a version of the Levenberg-Marquardt method. The Levenberg-Marquardt method was chosen as a benchmark because it is the most commonly used approach to NLS estimation [20]. The implementation of Levenberg-Marquardt used for comparison was written by the author and S.R. Shaw. This routine, referred to as LM, updates the parameter estimate in exactly the same manner as the ACM with $N = N_{max}$. The ACM and LM differ only in how the data set is manipulated. The octave code for LM is given in Appendix C.

Each of the routines used the same convergence tests and tolerances. Most convergence tolerances were set to the default values defined in the subroutine *goptions.m* (see Appendix D). However, the tolerances on the step in parameter space and the actual and predicted reductions were set to $\sqrt{eps}$.

Testing the Robustness of each Method

The robustness of each method was tested by identifying each test function for thousands of different parameters. Validation consisted of either fixing the initial guess and randomly generating the actual system parameters or fixing the actual parameters and randomly generating the initial guess. Randomly generating both

the initial guess and the system parameters ensured the test results were not skewed by an anomaly caused by a certain initial guess or data set. For either case, the random values were generated within a hypercube in the parameter space that limited the error between the initial guess and the actual parameter to 100%. The error between the initial guess and actual parameter value was computed as the relative error percentage

$$100 * \left(\frac{\| \hat{\theta} - \theta \|}{\| \theta \|} \right), \tag{4.6}$$

where $\hat{\theta}$ is the initial guess and θ is the actual parameter value.

Testing the SCM

Two important questions experimental procedures sought to answer was how Δ_N affected the convergence of the SCM and how Δ_N affected the number of function evaluations required by the SCM. These questions were addressed by estimating the parameters of each test problem using 11 different Δ_N's. The SCM updated the data set using

$$\vec{\Delta}_N = (2 \quad 5 \quad 10 \quad 20 \quad 25 \quad 40 \quad 50 \quad 100 \quad 200 \quad 500 \quad 1000) \tag{4.7}$$

for each parameter estimation problem. The convergence of the SCM and the average number of function evaluations required by the SCM at a given Δ_N were determined by applying the SCM to all the test problems with a fixed Δ_N. For example, to compute the convergence and average number of function evaluations of the SCM with $\Delta_N = 20$, the SCM with $\Delta_N = 20$ was applied to all of the test problems. The percentage of attempts for which the SCM with $\Delta_N = 20$ converged to the correct parameter value was the convergence of the SCM with $\Delta_N = 20$. The number of function evaluations necessary to obtain each correct parameter estimate averaged over all successful tests was the average number of function evaluations required by

the SCM with $\Delta_N = 20$. This process was repeated for each Δ_N defined in (4.7) and the results were plotted.

Testing the ACM

The number of data points evaluted by the ACM on the initial iteration was determined using the default value computed by (3.8). The ACM was tested by comparing it to the SCM with $\Delta_N = 50$ and LM. The SCM with $\Delta_N = 50$ was used for comparison because it provided an indication of how effectively the ACM updated the data set. The SCM with $\Delta_N = 50$ and the ACM solve identical problems on the initial iteration. On subsequent iterations the SCM updates the data set with a fixed $\Delta_N = 50$, whereas the ACM adaptively computes the optimal number of samples using the *trust region* approach outlined in Chapter 3. If $\Delta_N = 50$ was the optimal update in the data space, the ACM should choose to update the data set by 50 samples on each iteration. If $\Delta_N = 50$ is not optimal, the ACM should update the data space in a manner that is optimal. Thus, the ACM should converge at least as often as the SCM with $\Delta_N = 50$ if the *trust region* approach is effective.

Validation Results for the Single Sine Wave Functions

The single sine wave functions defined in (4.1) and (4.2) were used for validation because much attention has focused on developing methods to estimate the parameters of sine waves [31, 32, 33, 34]. Identification of sine waves is important because it is a fundamental task in many test and measurement systems. Characterization of A/D converters, digital oscilloscopes, and linear circuits are some examples. The two parameter sine wave function (4.1) allowed for a graphical comparison between the continuation methods and LM. This comparison provided valuable information about the ROC of these routines. However, most test and measurement systems are

characterized by the four parameter sine wave function (4.2). Validations with both (4.1) and (4.2) may seem redundant, but the insight provided by the graphical analysis of (4.1) and the practicality of the four parameter sine wave function (4.2) suggest that both be used as test functions.

Experimental tests using (4.1) consisted of 4000 test problems and tests conducted with (4.2) consisted of 2000 test problems. For both of these test functions, half of the test problems were defined with a fixed initial guess and randomly generated system parameters and the other half of the test problems were defined with fixed system parameters and randomly generated initial guesses. For the two parameter sine wave function, the random parameters were generated within the range

$$\begin{aligned} 0 \leq \ & x_1 \ \leq 4 \\ 0 \leq \ & x_2 \ \leq 8 \end{aligned}$$

and the fixed parameters guess were given by

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}.$$

For the four parameter sine wave function, the random parameters were generated within the hypercube

$$\begin{aligned} 0 \leq \ & x_1 \ \leq 4 \\ 0 \leq \ & x_2 \ \leq 8 \\ 0 \leq \ & x_3 \ \leq \frac{\pi}{2} \\ 0 \leq \ & x_4 \ \leq 5 \end{aligned} \tag{4.8}$$

and the fixed parameters were given by

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ \frac{\pi}{2} \\ 2.5 \end{pmatrix}.$$

Two Parameter Sine Wave with a Fixed Initial Guess

Results for the validations performed with (4.1), a fixed initial guess, and randomly generated system parameters are shown in Figs. 4.1- 4.4 and Table 4.1. Computing the convergence of the SCM for each Δ_N revealed that the convergence of the SCM is inversely proportional to Δ_N. Figure 4.1 illustrates the importance of choosing a reasonable step in the data space. When $\Delta_N = 2$ the SCM converged 100% of the time. However, as Δ_N increased the convergence of the SCM decreased and

Table 4.1
Validation results for test function (4.1) with a fixed initial guess.

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	9.6	2	1000
SCM	97.9	87	50
ACM	98.4	5	varied

approached that of LM. When $\Delta_N = N_{max}$ the SCM converged 9.1% of the time versus 9.6% for LM. The similarity between the convergence of the SCM with $\Delta_N = N_{max}$ and LM was expected. This behavior was expected because when $\Delta_N = N_{max}$ the SCM becomes a traditional NLS method that estimates the parameters using a weighted Gauss-Newton iteration.

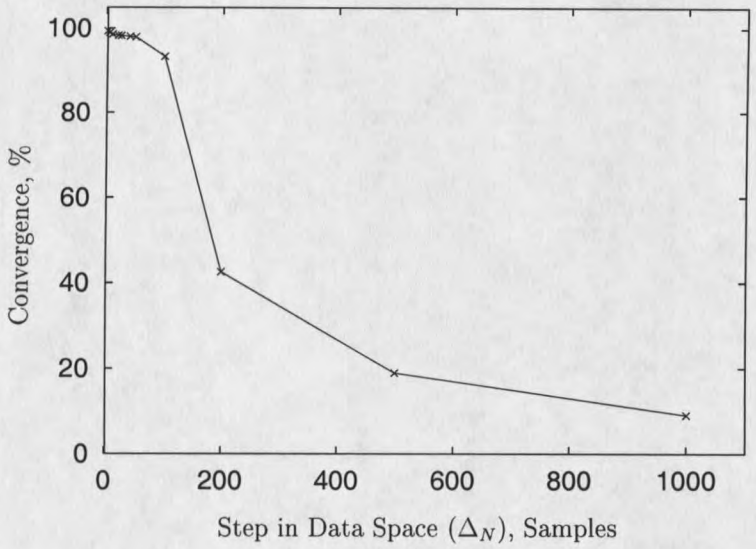

Fig. 4.1. The convergence of the SCM as a function of Δ_N when fitting equation (4.1), the initial guess was fixed, and the actual parameters were generated randomly.

Figure 4.2 shows the average number of function evaluations required by the SCM as a function of Δ_N. Notice that Δ_N is also inversely proportional to the number of function evaluations. When Δ_N is small there is a significant computational penalty. However, by analyzing Figs. 4.1 and 4.2 it is possible to find a Δ_N that provides both good convergence and minimal function evaluations. For example, the SCM converged 97.9% of the time and required an average of 87 function evaluations for convergence when $\Delta_N = 50$. This convergence is only marginally less than the 100% convergence obtained when $\Delta_N = 2$, but the number of function evaluations is over an order of magnitude less than the 1649 function evaluations required when $\Delta_N = 2$.

The reader should note that the lines connecting the convergence of the SCM at the discrete Δ_N's in Fig. 4.1 and the number of function evaluations of the SCM at discrete Δ_N's in Fig. 4.2 do not imply these quantities are linearly related to Δ_N between the discrete step sizes. The lines were included only to show the relationship between these quantities and Δ_N.

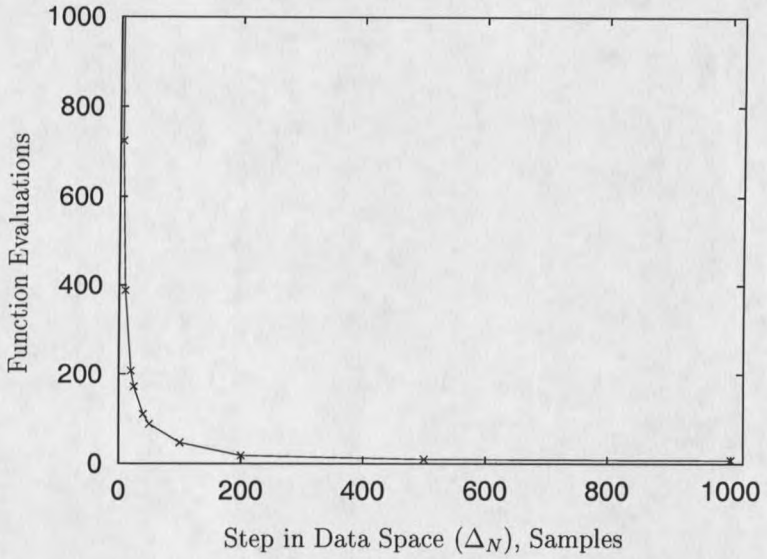

Fig. 4.2. Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.1), the initial guess was fixed, and the actual parameters were generated randomly.

A comparison of the validation results for the SCM with $\Delta_N = 50$, the ACM, and LM are given in Table 4.1. The validation results in Table 4.1 illustrate the increased ROC afforded by iteratively updating the data set. The convergence of the SCM and ACM are an order of magnitude better than LM. Another interesting result is that the convergence of the ACM was better than the SCM with $\Delta_N = 50$, but the ACM required significantly fewer function evaluations than the SCM. This demonstrates the value of adaptively updating the data set. Adaptively updating the data set can significantly reduce the trade-off between convergence and function evaluations.

Equation (4.1) had only two parameters making it possible to produce scatter plots showing the parameters for which the routines converged. The scatter plots were produced to demonstrate how the ROC of a traditional LM approach compared to the ROC of a routine that iteratively updated the data set. This was accomplished by choosing a single representative iterative approach (the SCM with $\Delta_N = 20$) and

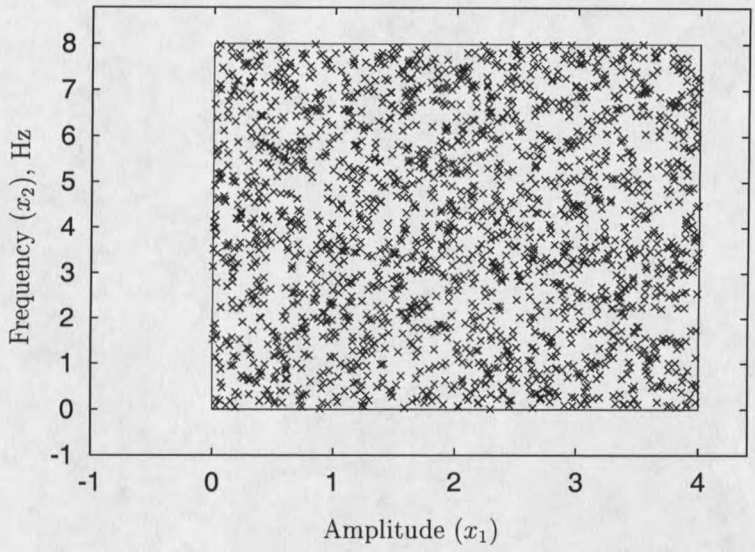

Fig. 4.3. Scatter plot of the 2000 actual system parameters used for the experiments with equation (4.1) and a fixed initial guess.

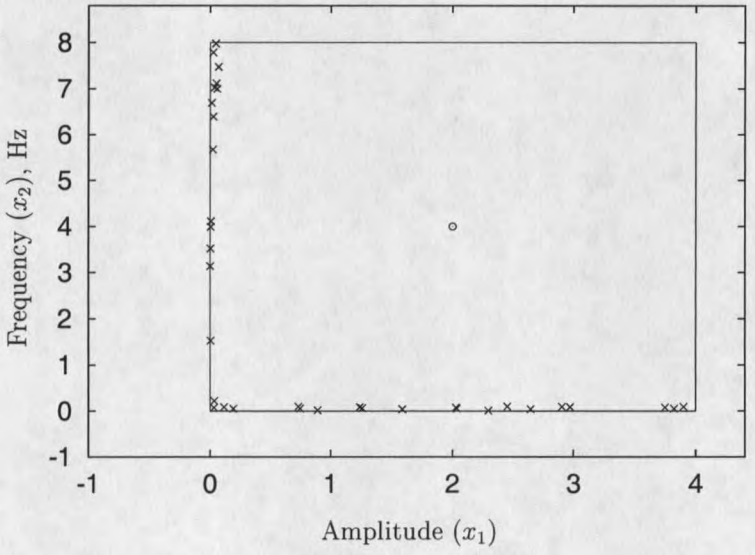

Fig. 4.4. Scatter plot of the actual system parameters for which the SCM did not converge when fitting (4.1), the initial guess was fixed, and the actual parameters were generated randomly.

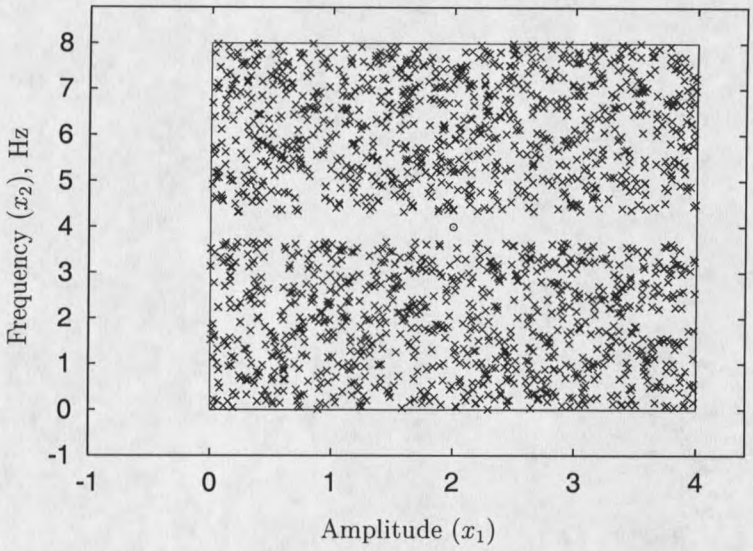

Fig. 4.5. Scatter plot of the actual parameters for which LM did not converge when fitting (4.1), the initial guess was fixed, and the actual parameters were generated randomly.

comparing it to a single representative traditional approach (LM). Figure 4.3 shows a scatter plot of the 2000 actual parameter values generated for this test function. These parameter values evenly cover the data space allowing up to 100% error between the initial guess and actual parameters. Figure 4.4 shows the actual parameters for which the SCM did not converge when $\Delta_N = 20$. The SCM failed to converge for only 34 of the 2000 test problems. Furthermore, the SCM only failed when the error in either x_1 or x_2 was nearly 100%. Figure 4.5 shows the actual parameters for which LM failed to converge. LM failed to converge for 1809 of the 2000 test problems. LM converged reliably only when the relative error between the actual frequency of the sine wave (x_2) and the initial guess was less than 15%. These scatter plots illustrate

Table 4.2
Validation results for test function (4.1) with fixed system parameters.

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	9.9	3	1000
SCM	99.5	40	50
ACM	99.5	5	varied

that the SCM has a much larger ROC than LM when estimating the parameters of a sine wave.

Two Parameter Sinewave with Fixed System Parameters

Results for the experiments conducted on (4.1), with fixed system parameters, and randomly generated initial guesses are shown in Figs. 4.6 - 4.9 and Table 4.2. The convergence of both the ACM and the SCM with $\Delta_N = 50$ was 99.5%. This was significantly better than LM which converged only 9.9% of the time. Figure 4.7 plots the average number of function evaluations for the SCM as a function of Δ_N. Figure 4.7 and Table 4.2 once again demonstrate that the ACM provides convergence that is comparable to the SCM while requiring significantly fewer function evaluations when estimating the parameters of (4.1). The scatter plots shown in Figs 4.8- 4.10 demonstrate that the SCM with $\Delta_N = 20$ has a much larger ROC than LM.

Four Parameter Sine Wave with a Fixed Initial Guess

Estimation results for (4.2) with a fixed initial guess and 1000 randomly generated system parameters are shown in Figs. 4.11, 4.12 and Table 4.3. The convergence of the SCM and ACM was quite high although they were lower than for the tests conducted with the two parameter sine wave (4.1). The convergence of the SCM was 91.9%

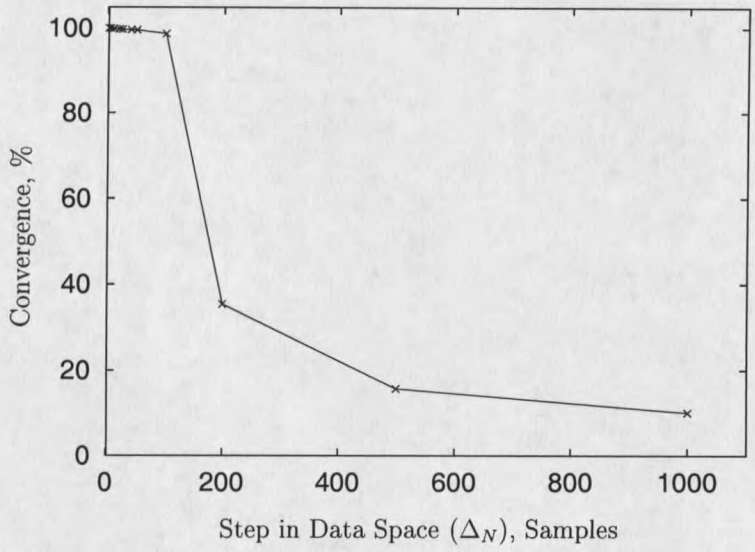

Fig. 4.6. Convergence of the SCM as a function of Δ_N when fitting equation (4.1), the actual parameters were fixed, and the initial guesses were generated randomly.

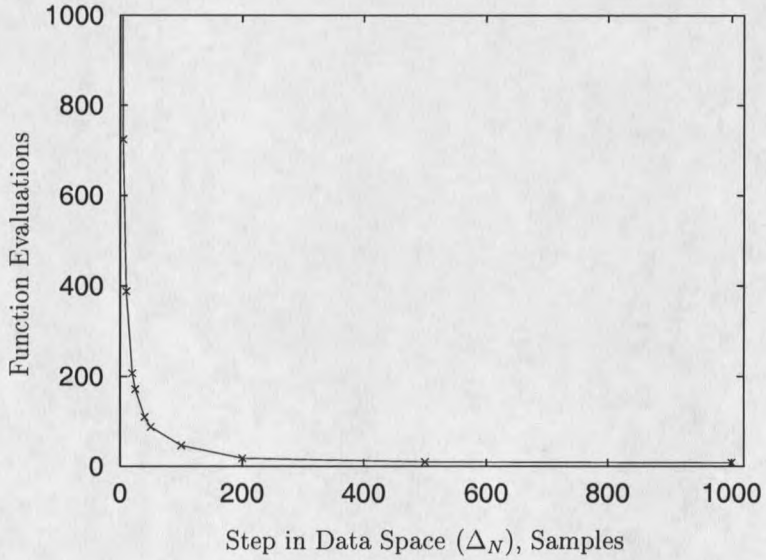

Fig. 4.7. Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.1), the actual parameters were fixed, and the initial guesses were generated randomly.

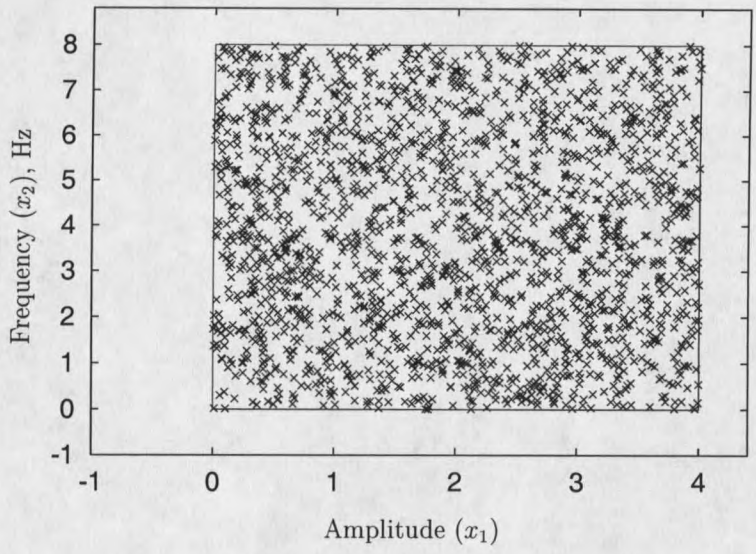

Fig. 4.8. Scatter plot of the 2000 initial guesses generated for the experiments with equation (4.1) and fixed system parameters.

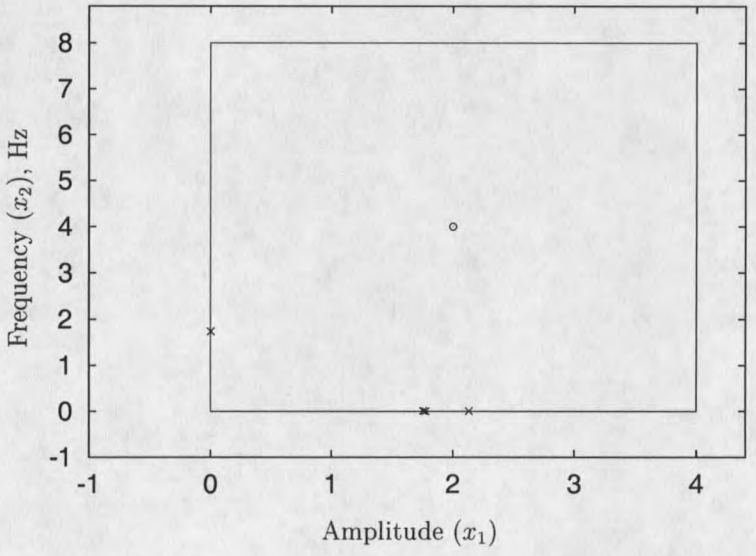

Fig. 4.9. Scatter plot of the initial guesses for which the SCM did not converge when fitting (4.1), the actual parameters were fixed, and the initial guesses were generated randomly.

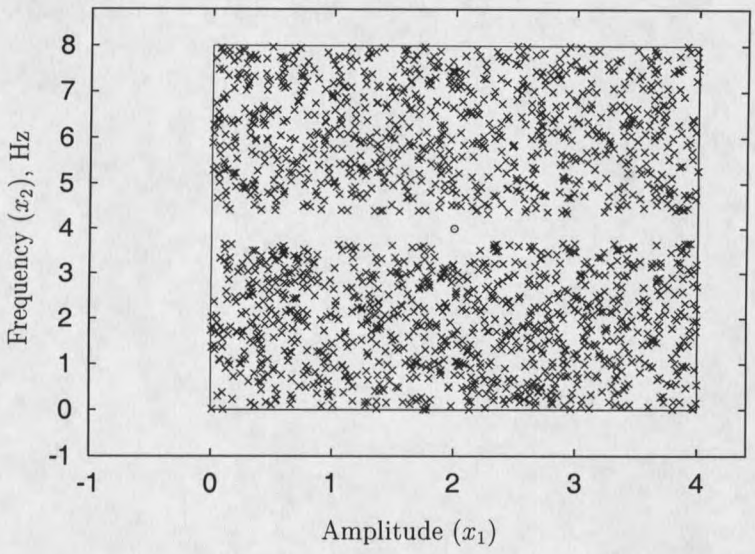

Fig. 4.10. Scatter plot of the initial guesses for which LM did not converge when fitting (4.1), the actual parameters were fixed, and the initial guesses were generated randomly.

with $\Delta_N = 2$ and the convergence of the ACM was 76.4%. Both of these routines performed significantly better than LM which converged only 9.3% of the time. The ACM required fewer function evaluations than the SCM with $\Delta_N = 50$. However, the ACM did not converge as often as the SCM with $\Delta_N = 50$. The convergence of the SCM when $\Delta_N = 50$ was 89.7% compared to 76.4% for the ACM. Even though both routines evaluated the same number of data samples on the initial iteration, the SCM converged 13.3% more often than the ACM. This indicates that the ACM may not be updating the data set optimally.

Four Parameter Sine Wave with Fixed System Parameters

Estimation results for (4.2) with fixed system parameters and 1000 randomly generated initial guesses are shown in Figs. 4.13, 4.14 and Table 4.4. Once again, the SCM and ACM performed much better than LM. The convergence of the SCM

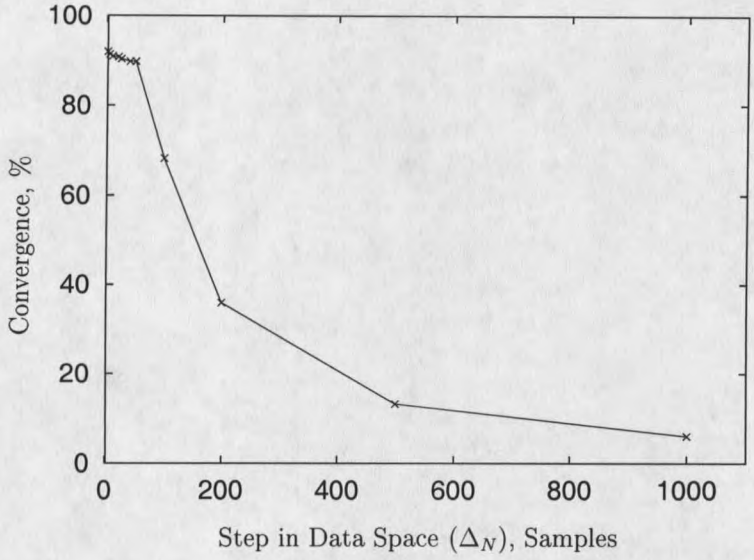

Fig. 4.11. Convergence of the SCM as a function of Δ_N when fitting equation (4.2), the initial guess was fixed, and the actual parameters were generated randomly.

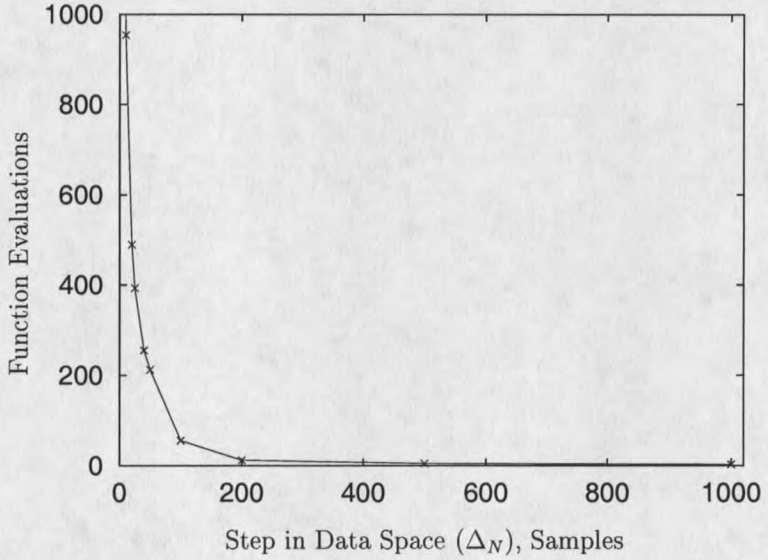

Fig. 4.12. Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.2), the initial guess was fixed, and the system parameters were generated randomly.

Table 4.3
Validation results for test function (4.2) with a fixed initial guess.

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	9.3	4	1000
SCM	89.7	255	50
ACM	76.4	14	varied

Table 4.4
Validation results for test function (4.2) with fixed system parameters.

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	9.1	4	1000
SCM	85.5	90	50
ACM	81.3	12	varied

was 96% when $\Delta_N = 2$. The convergence of the SCM was greater than 85% when $\Delta_N \leq 50$ and the convergence of the ACM was 81.3%. LM converged only 9.1% of the time. The SCM with $\Delta_n = 50$ once again had better convergence than the ACM. This is another indication that the ACM is not optimally updating the data set.

Validation Results for Sum of two Sine Waves Function

Experimental tests for (4.3) were conducted with the system parameters fixed at

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 3 \\ 5 \end{pmatrix}$$

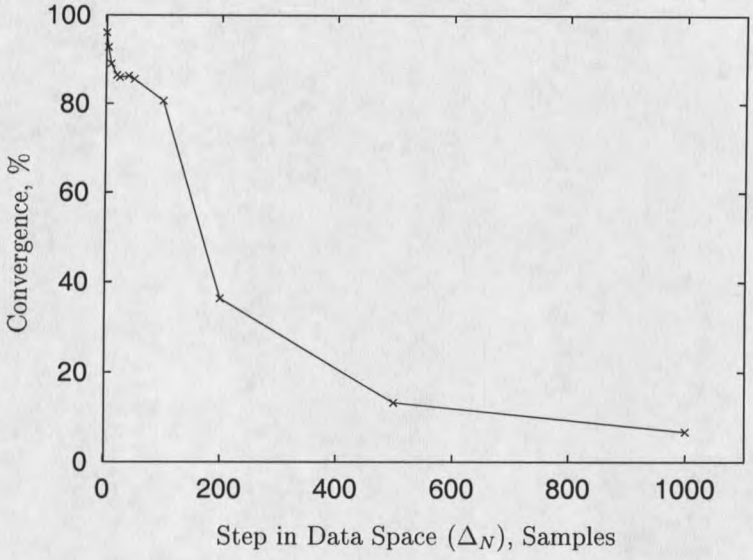

Fig. 4.13. Convergence of the SCM as a function of Δ_N when fitting equation (4.2), the system parameters were fixed, and the initial guesses were generated randomly.

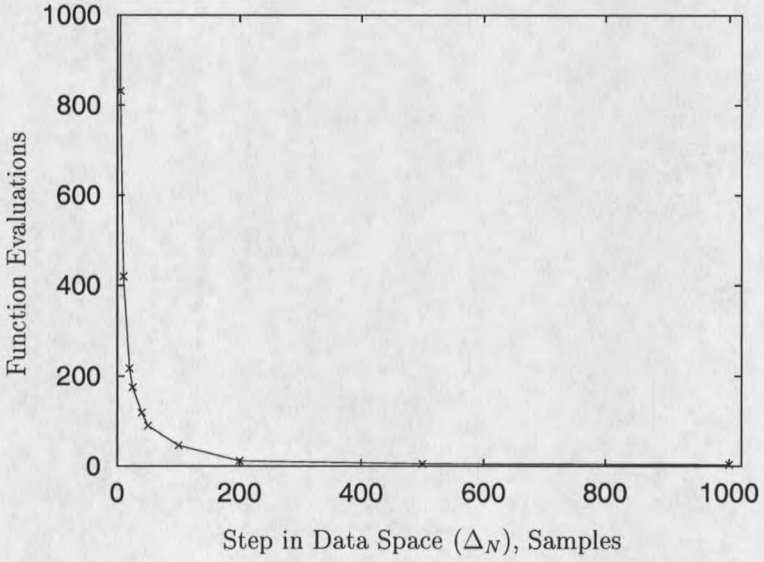

Fig. 4.14. Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.2), the actual parameters were fixed, and the initial guesses were generated randomly.

Table 4.5
Validation results for test function (4.3) with fixed system parameters.

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	2	48	1000
SCM	91.9	348	50
ACM	76.7	154	varied

and the initial guesses generated randomly within the range

$$\begin{aligned} 0 \leq\ & x_1 \leq 4 \\ 0 \leq\ & x_2 \leq 8 \\ 0 \leq\ & x_3 \leq 6 \\ 0 \leq\ & x_4 \leq 10. \end{aligned}$$

Validations involving (4.3) were performed using only fixed system parameters because tests conducted with (4.1) and (4.2) indicated that the results were not skewed by fixing either the initial guess or the system parameters. Results for (4.3) with fixed system parameters and 1000 randomly generated initial guesses are shown in Figs. 4.15, 4.16 and Table 4.5.

The results obtained from (4.3) illustrated how the continuation methods and LM behave when functional complexity increases. Figures 4.15 and 4.16 demonstrate that the SCM becomes more dependent on Δ_N and requires more function evaluations as the complexity of the function increases. This implies that Δ_N should be decreased as the complexity of the problem increases to safeguard against falling into a local minimum. The convergence results were still quite remarkable. The convergence of the SCM was 96.4% when $\Delta_N = 2$ and 91.9% when $\Delta_N = 50$. The ACM converged 76.7% of the time. The convergence of the SCM and ACM was significantly better than LM which had a convergence of only 2%.

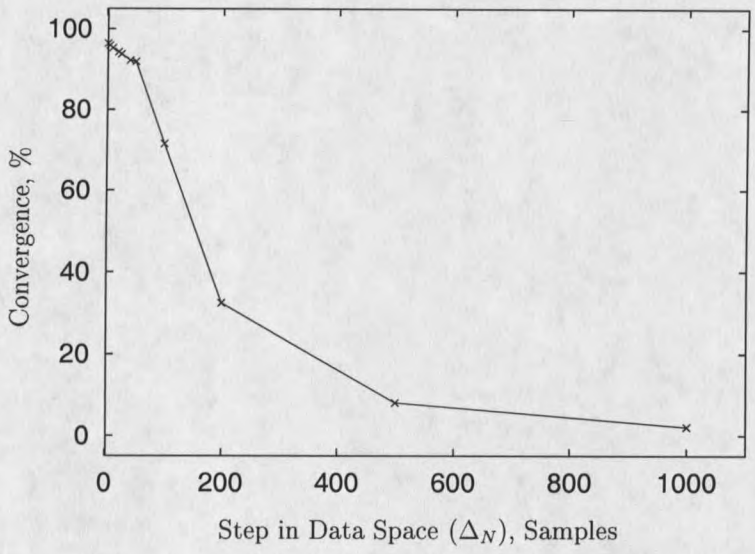

Fig. 4.15. Convergence of the SCM as a function of Δ_N when fitting equation (4.3), the system parameters were fixed, and the initial guesses were generated randomly.

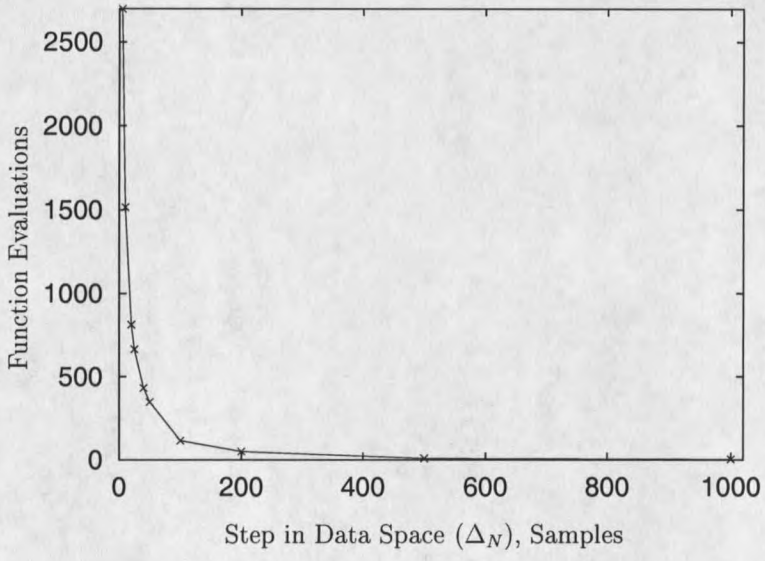

Fig. 4.16. Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.3), the system parameters were fixed, and the initial guesses were generated randomly.

Table 4.6
Validation results for test function (4.4) with a fixed initial guess.

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	0.5	95	1000
SCM	71.6	1184	50
ACM	43.2	286	varied

Validation Results for the Sum and Difference of Sinusoids

Experiments with (4.4) consisted of 1000 separate test problems with the initial guess fixed at

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 3 \\ 5 \\ 1 \\ 4 \end{pmatrix}$$

and system parameters randomly generated in the range

$$\begin{aligned} 0 \leq\ & x_1 \leq 4 \\ 0 \leq\ & x_2 \leq 8 \\ 0 \leq\ & x_3 \leq 6 \\ 0 \leq\ & x_4 \leq 10 \\ 0 \leq\ & x_5 \leq 2 \\ 0 \leq\ & x_6 \leq 8. \end{aligned}$$

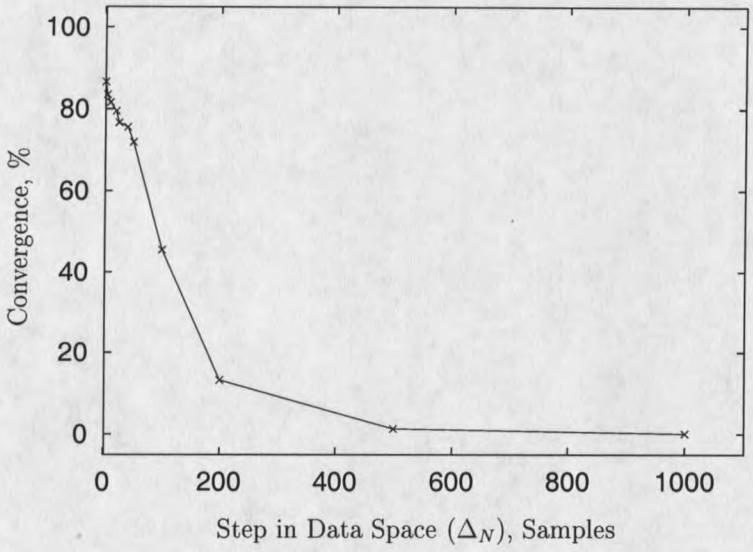

Fig. 4.17. Convergence of the SCM as a function of Δ_N when fitting equation (4.4), the initial guess was fixed, and the system parameters generated randomly.

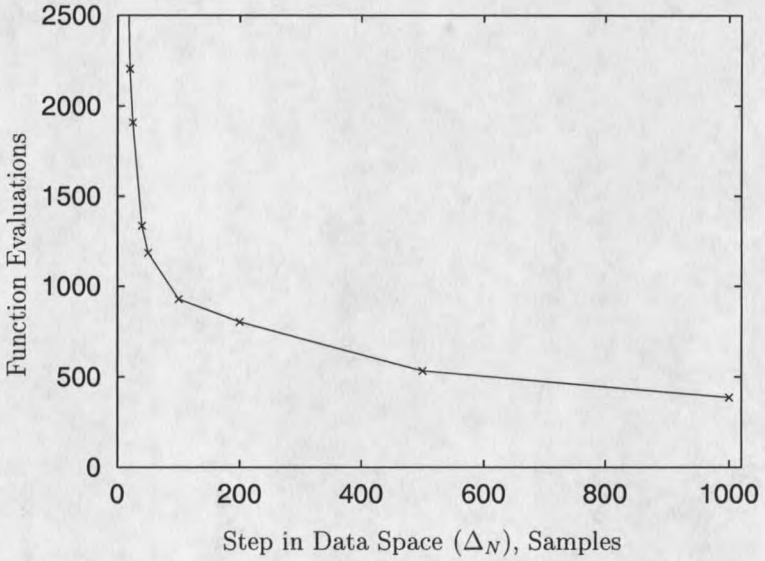

Fig. 4.18. Average number of function evaluations required for the SCM to converge as a function of Δ_N when fitting equation (4.4), the initial guess was fixed, and the system parameters were generated randomly.

Validation results for (4.4) are shown in Figs. 4.17, 4.18 and Table 4.6. Another increase in functional complexity resulted in another decrease in the convergence and another increase in the average number of function evaluations required by each routine. The convergence of the SCM was 86.6% when $\Delta_N = 2$. The SCM converged 71% of the time when $\Delta_N = 50$ and the ACM converged 43.2% of the time. Both the SCM and the ACM performed significantly better than LM which had a convergence of 0.5%.

Validation Results for the Box Three Dimensional Function

Experiments conducted with (4.5) tested the ability of the continuation methods and LM to estimate the true parameters of this function using 21 different initial guesses defined by

$$\hat{\theta}_0 \;=\; m * (0 \quad 10 \quad 20), \tag{4.9}$$

where

$$m \;=\; (0 \quad .5 \quad 1 \quad 1.5 \quad \ldots \quad 9.5 \quad 10). \tag{4.10}$$

The schedule taken by the SCM through the data space was defined by

$$\vec{N} \;=\; (2 \quad 4 \quad \cdots \quad 18 \quad 20). \tag{4.11}$$

One schedule through the data space $\vec{N}$ was used because the function only consisted of 20 data points. This limited the number of possible schedules, so a reasonable schedule was chosen and used for experimental purposes. Validation results are shown in Table 4.7. The convergence of the SCM was 3 times greater than either LM or the ACM. The SCM was expected to converge more than LM, but it was unusual that the SCM performed so much better than the ACM. However, a direct comparison between the SCM and the ACM is impossible because the SCM evaluated 2 data

Table 4.7
Validation results for the Box 3-Dimensional function (4.5)

Routine	Convergence (%)	Average Function Evaluations	Δ_N
LM	28	45	20
SCM	95	2353	2
ACM	28	153	varied

points on the first iteration whereas the ACM evaluated 5 data points. The ACM may have failed on the first iteration and not while updating the data set.

The experimental results presented in this chapter demonstrate that the *trust region* approach used by the ACM to update the data space is not optimal. The SCM with $\Delta_N = 50$ often had better convergence than the ACM. Moreover, as the complexity of the test functions increased the difference between the SCM with $\Delta_N = 50$ and the ACM increased. The convergence of the ACM when fitting the single sine wave function (4.1) was better than the SCM with $\Delta_N = 50$. For the more complex functions in (4.2), (4.3), and (4.4) the SCM with $\Delta_N = 50$ converged more often than the ACM. The convergence of the ACM was only 4.2% less that the SCM with $\Delta_N = 50$ for (4.2) with fixed system parameters. The difference then began to steadily increase. The SCM with $\Delta_N = 50$ converged 13.3%, 15.2%, 28.4%, and 67% more often than the ACM for (4.2) with the initial guess fixed, (4.3), (4.4), and (4.5), respectively. This indicates that the *trust region* approach used by the ACM is valid, but not optimal. The approach is valid because it produced a larger ROC than a traditional NLS approach. It is not optimal because it often had a smaller ROC than the SCM with $\Delta_N = 50$. The ACM worked great for simple functions, but only marginally for more complex functions. This seems to indicate that the *trust region* approach often chose steps that were too large in the data space.

More tests need to be conducted to determine if the ACM failed because it chose too large of step in the data space. The experimental tests were not specifically designed to test whether the ACM failed while updating the data set or failed while estimating the parameters. When estimating the parameters of (4.5) the ACM only converged when LM converged. This may have implied that the ACM failed while updating the parameter estimate and not the data space. However, it is unlikely that a Levenberg-Marquardt approach would fail to converge when Gauss-Newton converges.

A PRACTICAL IMPLEMENTATION ON A FUEL CELL

Experimental results from chapter 4 tested the validity of the continuation methods using a wide variety of simulated data sets. This chapter validates the continuation methods using data obtained from a real world power system. This chapter compares the ability of the continuation methods and LM to estimate the dynamic response of an Avista Labs SR-12 500W PEM fuel cell. A fuel cell was chosen because of the recent interest in fuel cells as power generation systems [35, 36, 37, 38] and because it provided a test of how well the methods performed in the presence of significant noise.

The following sections present the dynamic model used to predict the response of the fuel cell, the experimental setup used to obtain the dynamic data from the fuel cell, and the validation results obtained from the continuation methods and LM. This level of detail is included because the model used to predict the response of the fuel cell is unpublished and was developed by the author and S.R. Shaw. The validation model is an integral part of any validation test and understanding the validation model is an important part of understanding the validation results.

Dynamic Fuel Cell Model

The dynamic electrical terminal model used for validation was developed by coupling a linear steady-state electrical terminal model with a transient thermal model. The steady-state model describes the equilibrium conditions of the fuel cell and the thermal model describes the transient behavior between steady-states. The linear model, expanded to facilitate thermal modeling, was expressed

$$V_{stack} = V_1 + V_2 T - i r_1 - i T r_2. \tag{5.1}$$

The transient thermal response was modeled as

$$\frac{dT_{meas}}{dt} = ai + bi^2 - c(T_{meas} - T_{amb}), \tag{5.2}$$

where a, b, and c are parametric coefficients. In equation (5.2) the current was assumed to be proportional to the reactants consumed. Thus each parameter was implicitly scaled by a factor of

$$\frac{1}{M_{stack} HC_{stack}} \tag{5.3}$$

where M_{stack} is the mass of fuel cell and HC_{stack} is the heat capacity of the fuel cell. Several processes account for the net heat generated by the chemical reactions of a fuel cell. These include the sensible heat absorbed, the theoretical energy produced by the reaction, the electrical energy output, heat loss from the stack [14]) and heat generated by the internal resistance of the fuel cell. It can be verified that (5.2) accounts for all of the heat terms included in [14] and the heat generated by the internal resistance. The first current term (i) accounts for the theoretical energy produced by the fuel cell, a portion of the electrical energy output, and a portion of the sensible heat produced by the reactants. The current squared term (i^2) models the increase in temperature that results from the membrane heating up as the current increases. The temperature term ($T_{meas} - T_{amb}$) accounts for heat loss from the stack and a portion of sensible heat produced by the reactants.

The transient thermal response of (5.2) was modeled using the electric circuit analogy shown in Fig. 5.1. The resistor r_a is the thermal resistance between T_{amb} and T_{meas}, the resistor r_b is the thermal resistance between T_{meas} and T_{mem}, and the capacitor C represents the heat capacity of the membrane. The combined response of r_a, r_b, and C models the thermal time constant of the fuel cell.

The temperature measurement T_{meas} used for experimental purposes was located near the gas diffusion layer in the outlet air channel of a central cartridge as shown in Fig. 5.2. The membrane temperature T_{mem} would ideally be measured, but it was

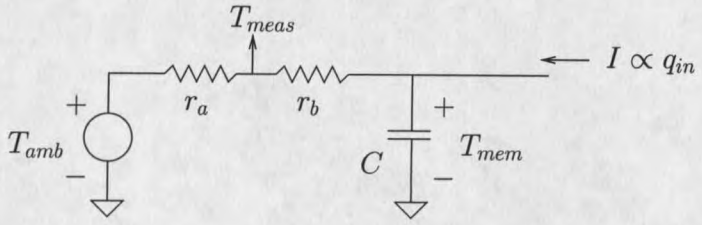

Fig. 5.1. Circuit analogy for the temperature response of a fuel cell.

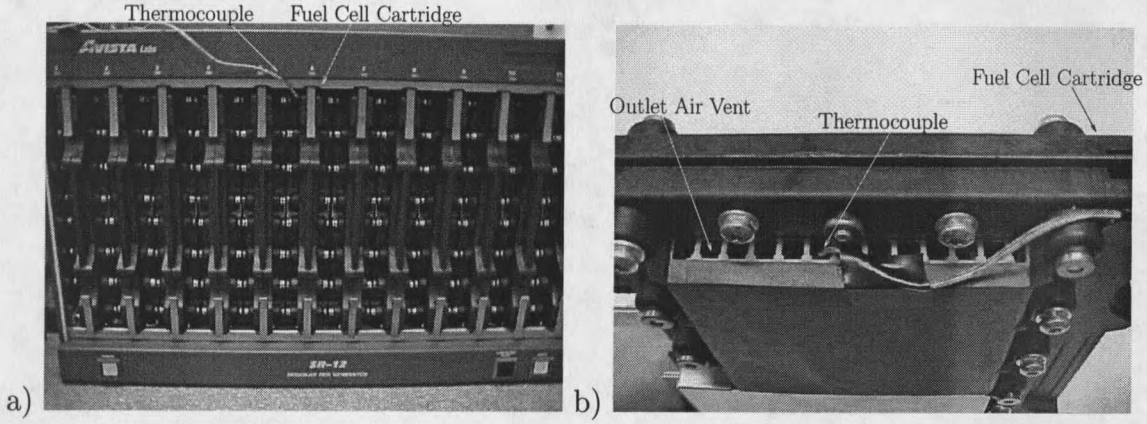

Fig. 5.2. Location of K-type thermocouple in SR-12 stack a) and location of thermocouple in the fuel cell cartridge b).

not possible to install a thermocouple inside the fuel cell cartridge without damaging the cartridge. However, T_{meas} is proportional to T_{mem} and the use of T_{meas} did not inhibit the ability of (5.2) to model the temperature response. It simply scaled the model parameters by a constant factor.

The measured temperature (T_{meas}) was broken apart using superposition

$$T_{meas} = \alpha_1 T_1 + \alpha_2 T_2 + T_3, \tag{5.4}$$

to simplify estimation. Here T_1 is the temperature response due to the current in the fuel cell, T_2 is the temperature response due to the internal resistance of the fuel cell, T_3 is the temperature response due to the difference between T_{amb} and T_{meas}, and the

$\alpha_i's$ are constants that scale T_1 and T_2 to obtain the correct units. The differential equations used to predict the transient behavior of T_1, T_2, and T_3 were

$$\frac{dT_1}{dt} = \beta(i - T_1) \tag{5.5}$$

$$\frac{dT_2}{dt} = \beta(i^2 - T_2) \tag{5.6}$$

$$\frac{dT_3}{dt} = \beta(T_{amb} - T_3) \tag{5.7}$$

$$\beta = \frac{1}{(r_a + r_b)C}. \tag{5.8}$$

The $T_i's$ in (5.5) - (5.7) are the homogeneous solutions to the differential equations. The terms i and T_{amb} represent the particular solution to the differential equation due to the given input. The final dynamic model was obtained by substituting equation (5.4) into equation (5.1) which yields

$$V_{stack} = V_1 + V_2 T_{meas} - i r_1 - r_2 i T_{meas} \tag{5.9}$$

where V_1, V_2, r_1, r_2, and the α_i's are parametric coefficients.

A simplified version of the dynamic model (5.9) was used to validate the continuation methods. The simplified model only included the temperature responses T_1 and T_3. The validation model was obtained by substituting

$$T_{meas} = \alpha_1 T_1 + T_3 \tag{5.10}$$

into (5.9). Validation was performed with this model because parameter estimations conducted with LM determined that the inclusion of T_2 reduced the prediction error by less than .001%. Removing T_2 simplified the model, reduced the computation time, and had a negligible effect on the predicition results.

Experimental Setup

The validity of (5.9) was tested using dynamic data taken from an Avista Labs SR-12 500W PEM fuel cell. Model validation used the SR-12 stack voltage (V_{stack})

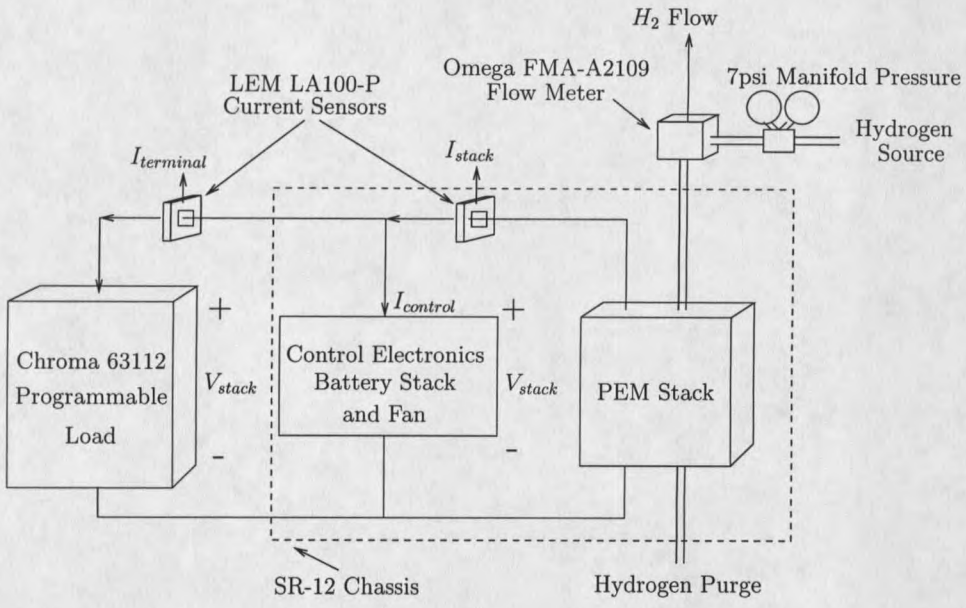

Fig. 5.3. Schematic of Experimental Setup.

and current (I_{stack}) shown in Fig. 5.3 and the temperature (T_{meas}) shown in Fig. 5.2.

An LEM LA100-P current sensor measured the stack current and an LEM LV25-P voltage sensor measured the stack voltage. A K-type thermocouple near the gas diffusion layer on the outlet air side of a central membrane of the SR-12 measured the temperature. The output of the thermocouple was processed by an Omega SMCJ thermocouple to analog converter. An Advantech PCI-1710 12-bit data acquisition card linked each of the measurement devices to a computer.

The SR-12 operated for 30 minutes under a 2A load after completing its factory programmed warm-up procedure before any experiments were conducted. This warm-up cycle resulted in a temperature of 305 K near the gas diffusion layer. Several dynamic experiments were then performed. Between each experiment the fuel cell operated under a 2A load and was allowed to cool until the outlet air temperature

reached 305 K. The stack was allowed to cool to 305 K to ensure that each experiment had similar starting conditions.

Experimental Results

The dynamic data obtained from the SR-12 fuel cell was measured at 50 samples/sec over a 180 second time interval and consisted of 9000 data samples. Validating this entire data set was very time consuming. The dynamic model (5.9) required the simulation of two differential equations to model the temperature response. Each of these simulations took about 8-10 minutes to complete. Thus, a single iteration through any of the routines took 20-30 minutes when the entire measured data set was used. The data set was downsampled to reduce computation time for validation. The original data set was downsampled by a factor of 3 to generate a new validation set consisting of 3000 data samples. The system parameters were estimated using this decimated data set. Once a parameter estimate $\hat{\theta}$ was obtained from the decimated data set, $\hat{\theta}$ was used to estimate the response for the entire measured data set. The errors between the actual and predicted responses and the plots showing the estimation results were obtained using the entire measured (non-decimated) data set. The error between the predicted and actual response was computed as the relative error percentage

$$r = 100 * \left(\frac{\| f_{obs} - f_{est} \|}{\| f_{obs} \|} \right) \tag{5.11}$$

where f_{obs} is the observed data set and f_{est} is the estimated data set obtained from the model (5.9) and the estimated parameters.

The convergence of each routine was tested from the common initial guess

$$\hat{\theta}_0 = (\, .0337 \quad 37.868 \quad -1.2959 \quad -1 \times 10^{-4} \quad 50 \quad 1 \times 10^{-3} \,)$$

A single initial guess was utilized because the purpose of identifying the dynamic response of the fuel cell was to demonstrate that the continuation methods can identify the parameters of practical systems. The SCM stepped through the data space following the schedule

$$\vec{N} = (1500 \quad 3000).$$

This schedule was chosen because it provided excellent results while reducing the number of function evaluations.

Validation results are shown in Table 5.1. All three methods estimated parameters that accurately predicted the dynamic response of the fuel cell. The relative error between the actual and predicted responses were 1.807%, 1.807%, and 1.808% for the SCM, the ACM, and LM, respectively. The measured output current obtained

Table 5.1
Validation results for the dynamic fuel cell model (5.9).

Parameter	$\hat{\theta}_{LM}$	$\hat{\theta}_{SCM}$	$\hat{\theta}_{ACM}$
β	.03315	.0329	.0322
V_1	37.29	37.23	37.09
r_1	-1.08	-1.06	-1.00
α_1	85.81	112.4	632.9
V_2	-7.72×10^{-4}	-5.85×10^{-4}	-1.02×10^{-4}
r_2	3.68×10^{-4}	2.84×10^{-4}	5.20×10^{-5}

from the SR-12 is shown in Fig. 5.4 and the actual voltage response and the response predicted by the SCM are plotted in Fig. 5.5. Only the response predicted by the SCM is plotted in Fig. 5.5 because the responses predicted by the ACM and LM were very similar. When plotted together, it was impossible to distinguish between the

three predictions. The prediction results demonstrate that, even in the presence of significant noise, both the SCM and ACM can accurately identify a practical system.

Some of the estimated parameters varied significantly even though the prediction errors for each routine were similar. All of the routines produced similar values for β, V_1, and r_1 which represent the reciprocal of the thermal time constant, the part of the open-circuit voltage that is independent of temperature, and the part of the total cell resistance that is independent of temperature, respectively. However, the routines predicted significantly different values for V_2, α_1, and r_2 which represent the part of the open-circuit voltage that depends on temperature, an arbitrary constant that scales T_1 to obtain the correct units, and the part of the resistance that is dependent on temperature, respectively. Each of the methods estimated V_2 and r_2 to have magnitudes on the order of 10^{-4}. Thus, their effect on the predicted response was negligible. This allowed similar prediction results despite relatively large differences in these parameter values. This result implies that the main response of the fuel cell is governed by the the thermal time constant β^{-1} and the portion of the open-circuit voltage and total cell resistance that are independent of temperature (V_1 and r_1, respectively).

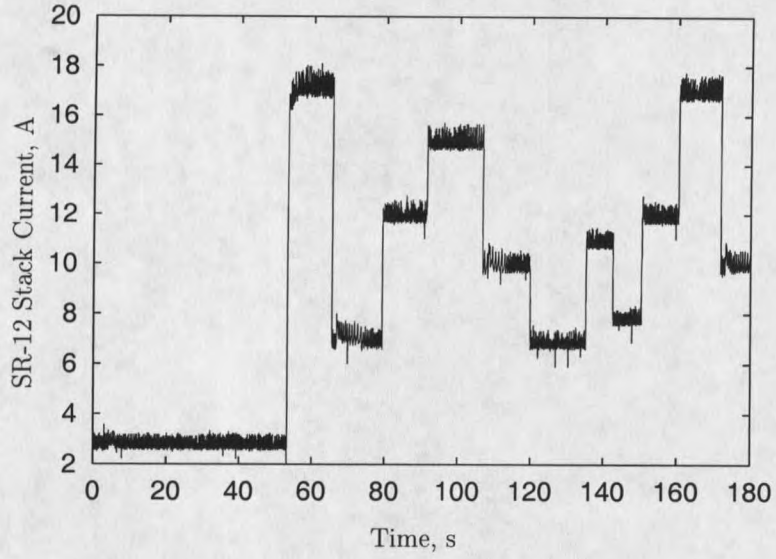

Fig. 5.4. Measured output current of the SR-12 PEM fuel cell used to model the output voltage.

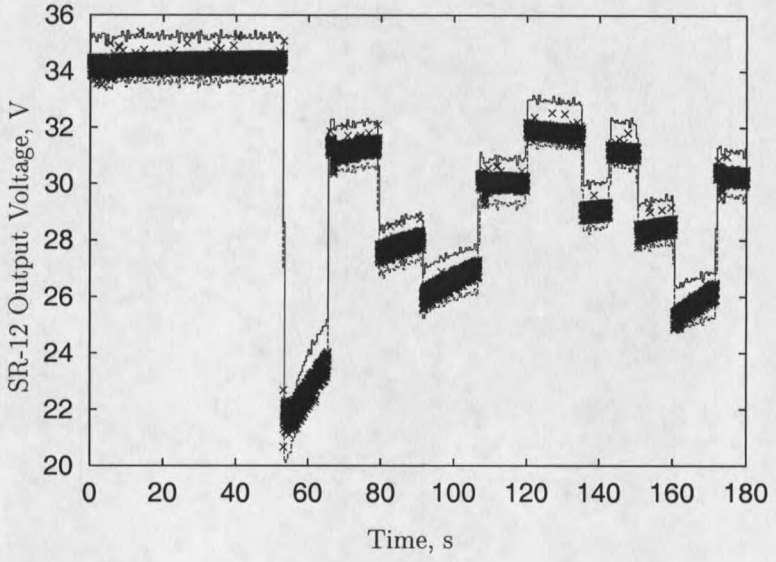

Fig. 5.5. Predicted and actual voltage response of the SR-12. The dark line is the response predicted by the SCM.

CONCLUSIONS

This thesis demonstrated that NLS estimation in conjunction with an iteratively updated data set provides a significantly larger ROC than a traditional Levenberg-Marquardt approach when applied to functions that can be represented by differential equations. The increased ROC afforded by these methods makes them a valuable tool for identifying physical systems. The following sections summarize important conclusions drawn from experiments, any contributions made by each of the continuation methods, and suggest directions for future research.

The Simple Continuation Method (SCM)

Experiments conducted with the SCM provided valuable insight into the behavior of routines that iteratively update the data set. The SCM demonstrated that there exists an inverse relationship between convergence and the number of function evaluations. The best convergence was obtained by the SCM when the number of data points added on each iteration was as small as possible ($\Delta_N = 2$). However, if $\Delta_N = 2$ the SCM required a large number of function evaluations for convergence. Experimental results demonstrated that a compromise between convergence and function evaluations could be obtained. A step in data space of

$$.02 * N_{max} \leq \quad \Delta_N \quad \leq .05 * N_{max}$$

generally provided the best combination of convergence and computational overhead for the test functions included in this thesis. Adding 2-5% of the data set on each iteration may not be optimal for every situation, but it is a good starting point.

The Adaptive Continuation Method (ACM)

The ACM required fewer function evaluations than the SCM, but this came at the price of reduced convergence. Given the experimental results from the SCM demonstrating the inverse relationship between convergence and the number of function evaluations, this was not entirely unexpected. However, a comparison between the convergence of the SCM with $\Delta_N = 50$ and the ACM indicated that the ACM did not update the data set optimally. The ACM could be improved by adaptively determining the initial interval $N_{k=0}$ and reducing the size of the *trust region* used to update the data set. The *trust region* used for the ACM limited the step in data space by keeping ratio between the step in the parameter space and the standard deviation of the parameters within the standard deviation of the measured noise. The trust region could be reduced by multiplying the standard deviation of the measured noise by some multiple less than one. The ACM could also be improved by adaptively determining the initial interval $N_{k=0}$. One way in which the ACM could adaptively determine $N_{k=0}$ would be to choose an initial data set $N_{k=0}$ over which the function is minimized on the initial iteration and if the minimization is unsuccessful reduce $N_{k=0}$. Repeat this process until the minimization is successful. This approach is similar to that proposed in [2] except it is only applied to determine the initial interval $N_{k=0}$ and it never increases the size of the interval.

Future Research

Future research should focus on generating an adaptive routine that provides a better trade-off between convergence and function evaluations. This research should focus on refining the specific methods and the general approaches presented in this thesis to develop routines that are suitable for on-line parameter estimation. This

may first require extensive testing of the ability of these methods to identify physical systems off-line before progressing to on-line applications.

REFERENCES

LIST OF REFERENCES

[1] W. H. Press, S. A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[2] S. R. Shaw and S. B. Leeb. Identification of induction motor parameters from transient stator current measurements. *IEEE Transactions on Industrial Electronics*, 46(1):139–149, February 1999.

[3] S. B. Leeb. *A Conjoint Pattern Recognition Approach to Nonintrusive Load Monitoring.* Phd, MIT, Department of Electrical Engineering and Computer Science, February 1993.

[4] S. B. Leeb and J. L. Kirtley. *A Transient Event Detector for Nonintrusive Load Monitoring.* U. S. Patent Number 5,483,153, Issued January 1996.

[5] S. B. Leeb and J. L. Kirtley. A multiscale transient event detector for nonintrusive load monitoring. In *Proceedings of the 1993 IECON International Conference on Industrial Electronics, Control, and Instrumentation*, 93CH3234-2, pp. 354–359.

[6] S. B. Leeb, S. R. Shaw, and J. L. Kirtley. Transient event detection in spectral envelope estimates for nonintrusive load monitoring. *IEEE Transactions on Power Delivery*, 7(3):1200–1210, July 1995.

[7] L. K. Norford and S. B. Leeb. Nonintrusive electrical load monitoring in commercial buildings base on steady-state and transient load detection algorithms. *Energy and Buildings*, 24(1):51–64, May 1996.

[8] S. R. Shaw, C. B. Abler, R. F. Lepard, D. Luo, S. B. Leeb, and L. K. Norford. Instrumentation for high performance nonintrusive electrical load monitoring. *ASME Journal of Solar Energy Engineering*, 120(3):224–229, August 1998.

[9] EGG Services Parsons Inc. Fuel cell handbook (5th edition). *DEO of Fossil Energy, National Energy Technology Lab*, 2000.

[10] J. Kim, S.M. Lee, S. Srinivasan, and C.E. Chamberlin. Modeling of proton exchange membrane fuel cell performance with an empirical equation. *Journal of the Electrochemical Society*, 142(8), August 1995.

[11] G. Maggio, V. Recupero, and L. Pino. Modeling polymer electrolyte fuel cells: An innovative approach. *Journal of Power Sources*, 101:275–286, 2001.

[12] J.C. Amphlett, R.M. Baumert, R.F. Mann, B.A. Peppley, P.R. Roberge, and T.J. Harris. Performance modeling of the ballard mark iv solid polymer electrolyte fuel cell, i. mechanistic model development. *Journal of the Electrochemical Society*, 142(1):1–8, January 1995.

[13] J.C. Amphlett, R.M. Baumert, R.F. Mann, B.A. Peppleyand P.R. Roberge, and T.J. Harris. Performance modeling of the ballard mark iv solid polymer electrolyte fuel cell, ii. empirical model development. *Journal of the Electrochemical Society*, 142(1):9–15, January 1995.

[14] J.C. Amphlett, R.F. Mann, B.A. Peppley, P.R. Roberge, and A. Rodrigues. A model predicting transient responses of proton exchange membrane fuel cells. *Journal of Power Sources*, 61:183–188, 1996.

[15] D.J. Hall and R.G. Colclaser. Transient modeling and simulation of a tubular solid oxide fuel cell. *IEEE Transactions on Energy Conversion*, 14(3):749–753, September 1999.

[16] R. Lasseter. Dynamic models for micro-turbines and fuel cells. In *IEEE Power Engineering Society Summer Meeting*, volume 2, pages 761–766, July 2001.

[17] H.K. Geyer, R.K. Ahluwalia, and R. Kumar. Dynamic response of steam-reformed, methanol-fueled polymer electrolyte fuel cell systems. In *31st Intersociety Energy Conversion Engineering Conference*, volume 2, pages 1101–1106, Feburary 1996.

[18] J. Padulles, G.W. Ault, and J.R. McDonald. An approach to the dynamic modelling of fuel cell characteristics for distributed generation operation. In *IEEE Power Engineering Society Winter Meeting*, volume 1, pages 134–138, January 2000.

[19] Rolf Johansson. *System Modeling and Identification.* Prentice Hall Information and System Sciences Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[20] Neil Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press, 1999.

[21] Lennart Ljung. *System Identification: Theory for the User.* Prentice-Hall, second edition, January 1999.

[22] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996.

[23] G. A. F. Seber and C. J. Wild. *Nonlinear Regression.* Probability and Mathematical Statistics. Wiley, 1989.

[24] S. R. Shaw. *System Identification Techniques and Modeling for Non-intrusive Load Diagnostics.* Phd, MIT, Department of Electrical Engineering and Computer Science, February 2000.

[25] S. J. Wright and J. N. Holt. Algorithms for nonlinear least squares with linear inequality constraints. *SIAM Journal on Scientific and Statistical Computing*, 6(4):1033–1048, October 1985.

[26] Paul T. Boggs, Richard H. Byrd, and Robert B. Schnabel. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal on Scientific and Statistical Computing*, 8(6):1052–1078, November 1987.

[27] Jorge J. Moré. The levenberg-marquardt algorithm: Implementation and theory. Technical report, Argonne National Laboratory, 1977.

[28] G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report 95-041, Department of Computer Science, University of North Carolina at Chapel Hill, March 2002.

[29] Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, March 1981.

[30] M.J. Box. A comparison of several current optimaization methods, and use of transformations in constrained problems. *Computer Journal*, 9(1):67–77, July 1966.

[31] Y. Jenq and P.B. Crosby. Sinewave parameter estimation algorithm with application to waveform digitizer effective bits measurement. *IEEE Transactions on Instrumentation and Measurement*, 37(4):529–532, December 1988.

[32] J.Q. Zhang, A. Xinmin, S. Jinwei, and H. Xiao. Sinwave fit algorithm based on total least-squares method. *IEEE Instrumentation and Measurement Technology Conference*, 46(4):1020–1024, August 1996.

[33] G. Cavone, N. Giaquinto, M. Savino, and A. Trotta. Improved sine-wave parameter estimation for a/d converter testing. *Electrotechnical Conference, 8th Mediterranean, MELECON 1996*, 1:501–504, May 1996.

[34] N. Giaquinto and A. Trotta. Fast and accurate adc testing via an enhanced sine wave fitting algorithm. *IEEE Transactions on Instrumentation and Measurement*, 46(4):1020–1024, August 1997.

[35] R. Kyoungsoo and S. Rahman. Two-loop controller for maximizing performance of a grid-connected photovoltaic-fuel cell hybrid power plant. *IEEE Transactions on Energy Conversion*, 13(3), September 1998.

[36] K. Sedghisigarchi and A. Feliachi. Control of grid-connected fuel cell power plant for transientstability enhancement. In *IEEE Power Engineering Society Winter Meeting*, volume 1, pages 383–388, January 2002.

[37] M.D. Lukas, K.Y. Lee, and H. Ghezel-Ayagh. Development of a stack simulation model for control study on direct reforming molten carbonate fuel cell power plant. *IEEE Transactions on Energy Conversion*, 14(4):1651–1657, December 1999.

[38] J. Padulles, G.W. Ault, C.A. Smith, and J.R. McDonald. Fuel cell plant modelling for power systems simulation. In *34th Universities Power Engineering Conference*, volume 1, pages 21–25, September 1999.

APPENDICES

APPENDIX A

OCTAVE CODE FOR THE SCM

Main Program: SCM

```
function [mu,info,nfev] = SCM(f, g, mu, Pinv0, schedule, ...)
  info = "";
  nfev = 0;
  % main loop
  for i = 1:length(schedule)
    N = schedule(i);
    [mu,Pinv,J,dx,grad,nevs,info] = iterate(f,Pinv0,mu,N,all_va_args);
    nfev = nfev + nevs / schedule(length(schedule));
    if(g != '')
      feval(g,mu,Pinv,N,all_va_args);
    end;
  end;
end;
```

Subroutine: iterate

```
function [x,pinv,J,dx,grad,nevals,info] = iterate(f, P0, x, N, ...)
  nevals = 0;
  newnorm = 0;
  rval = 0;
  while(rval == 0)
    % compute residual and Jacobian
    [J,err] = fdjac(f,x,[],N, all_va_args);
    nevals = nevals + length(x)+1;
    oldnorm = newnorm;
    newnorm = norm(err);
    % Compute inverse covariance
```

```
    % matrix, gradient, and step
    pinv = P0 + J'*J;
    grad = J'*err;
    dx = -pinv\grad;
    % take step
    x = x + dx;
    % check convergence
    [rval, info] = isconverged(grad, nevals, dx, x, newnorm,
       oldnorm, norm(err+J*dx));
  end;
  nevals = nevals * N;   % scale by number of points we're evaluating
end;
```

Subroutine: isconverged

```
function [rval, info] = isconverged(grad,nfev,dx,x,
                                    newnorm,oldnorm,prednorm)
  opts = goptions();
  rval = 0;
  info = "";
  % check the gradient
  if(norm(grad) < opts(4))
    rval = 4;
    info = sprintf("gradient norm = %.3e < goptions(4) = %.3e",
                   norm(grad), opts(4));
  end;
  % number of function evaluations
  if(nfev > opts(14))
    info = sprintf("number of function evaluations exceeds %d",
```

```
                    opts(14)*length(x));
    rval = 14;
  end;
  % check relative step size
  if(norm(x) && norm(dx) < opts(2)*norm(x))
    info = sprintf("relative step norm = %.3e, less than
                    goptions(2) = %.3e'', norm(dx)/norm(x), opts(2));
    rval = 2;
  end;
  % check actual and predicted reductions.
  % do not perform check if oldnorm is zero.
  if(oldnorm != 0)
    if(abs((newnorm - oldnorm)/oldnorm) < opts(3) &&
       abs((newnorm - prednorm)/oldnorm)  < opts(3))
      rval = 3;
      info = sprintf("actual and predicted reduction in residual
                      norm less than %.3e", opts(3));
    end;
  end;
end;
```

APPENDIX B

OCTAVE CODE FOR THE ACM

Main Program: ACM

```
function [x,info,nfev,N] = ACM(f,x,D,N_max,sigma2,...)
  %# Make sure x is a column vector
  if(size(x,1) < size(x,2))
    x = x';
  end;
  %# Initialize variables
  opt = goptions();
  if opt(14) == 0
    opt(14) = 100 * length(x);
  end;
  sigma2 = max([1e-4 sigma2]);
  alpha = 0;
  info = "";
  N = max([5 round(.05*N_max)]);
  fvec = feval(f,x,N,all_va_args);
  nfev = 1;
  mode = (D == []);
  iter = 1;
  fnorm = norm(fvec);
  %# begin outer loop
  while (info == "")
    %# evaluate the Jacobian and compute its SVD
    [fjac,fvec] = fdjac(f,x,fvec,N,all_va_args);
    nfev = nfev + columns(fjac);
    [u,s,v] = svd(fjac,0);
    jcnorms = sqrt(diag(v*s*s*v'));          %# column norms
```

```
utf = u'*fvec;
%# on first iteration compute r, D, and initialize step
if iter ==  1;
  if mode == 1
    D = diag(jcnorms + (jcnorms == 0));
  end;
  xnorm = norm(D*x);
  step = opt(18)*xnorm;
  if step == 0
    step = opt(18);
  end;
end;
%# compute norm of the scaled gradient
gnorm = 0;
if fnorm != 0
  temp = (diag(s) == 0);
  temp = v*diag((1-temp)./(diag(s)+temp))*utf/fnorm;
  %# if column norm = zero, corresponding x is zero
  %# otherwise, scale by column norm
  temp = abs(temp.*(jcnorms != 0)./(jcnorms+(jcnorms==0)));
  gnorm = max([gnorm; temp]);
end;
%# check if gradient norm is less than opt(4)    (gtol)
if (gnorm < opt(4))
  info = sprintf("gradient norm = %.3e, less than gtol = %.3e",
                 gnorm, opt(4));
else
```

```
%# Rescale D
if mode == 1
  D = diag(max([diag(D)'; jcnorms']));
end;
%# begin inner loop
ratio = 0;
while (ratio < .0001)
 %# determine LevenburgMarquardt parameter
 [u1,s1,v1] = svd(s*v'*diag(1./diag(D)),0);
 [delta,alpha] = lsvd(step, .1, alpha, v1, s1, u1'*utf);
 pnorm = norm(delta);
 delta = -(diag(1./diag(D)))*delta;
 %# determine new x and store in x1
 x1 = x + delta;
 %# on the first iteration adjust the initial step bound
 if iter == 1;
   step = min([step pnorm]);
 end;
 fx1 = feval(f,x1,N,all_va_args);
 fnorm1 = norm(fx1);
 nfev = nfev + 1;
 if fnorm == 0
   error("fnorm == 0\n");
 end;
 %# compute the scaled actual reduction
 actred = -1;
 if (.1*fnorm1 < fnorm);
```

```
  actred = 1 - (fnorm1/fnorm)^2;
end;
%# compute the scaled predicted reduction and
%#directional derivative
temp1 = norm(s*v'*delta)/fnorm;
temp2 = (sqrt(alpha)*pnorm)/fnorm;
predred = temp1*temp1 + 2*temp2*temp2;
dirder = -(temp1*temp1 + temp2*temp2);
%# calculate ratio of actual to predicted reduction
ratio = 0;
if predred != 0
  ratio = actred/predred;
end;
%# Update the step bound
if (ratio > .25)
 if (ratio >= .75 || alpha == 0)
   step = pnorm*2;
   alpha = .5*alpha;
 end;
else
 if (actred >= 0)
   temp = .5;
 else
   temp = .5*dirder/(dirder + .5*actred);
 end;
 if (.1*fnorm1 >= fnorm || temp < .1)
   temp = .1;
```

```
 end;
 step = temp*min([step pnorm*10]);
 alpha = alpha/temp;
end;
%# when iteration successful, save x, fvec, and
%#calculate norm x
if ratio >= .0001
  x = x1;
  xnorm = norm(D*x);
  fvec = fx1;
  fnorm = fnorm1;
  iter = iter + 1;
 end;
 %%%% Tests of convergence %%%%
 %# is actual and predicted reduction less than opt(3)?
 if(abs(actred)<=opt(3) && predred<=opt(3) && .5*ratio<=1)
   if (step <= opt(2)*xnorm)
     info = sprintf("actual and predicted reduction less than
                    %.3e and relative step less than %.3e",
                    opt(3),opt(2));
   else
     info = sprintf("actual and predicted reduction less than
                    %.3e", opt(3));
   end;
   break;
 end;
 %# is step in parameter space less than opt(2)*norm(x). (xtol)
```

```
      if (step <= opt(2)*xnorm)
info = sprintf("maximum relative step less than %.3e", opt(2));
        break;
      end;
      %# too many iterations?
      if nfev >= opt(14)
        info = sprintf("number of function evaluations exceeds
                        %d",opt(14));
        return;
      end;
    end;
  end;
  if ((info != "" || (ratio >= .75 && alpha == 0))
                      && N < N_max)
    fnorm1 = 0;
    [fjac,fx1] = fdjac(f,x,[],N,all_va_args);
    nfev = nfev + columns(fjac);
    [u,s,v] = svd(fjac,0);
    temp = sqrt(abs(diag(pinv(v*s*s*v'))*sigma2));
    seinv = diag(1./(max(temp, eps)));
    while(fnorm1 < eps && N < N_max)
      Nlim = N + min([round(N_max/10) N_max-N]);
      [fjac,fx1] = fdjac(f,x,[],Nlim,all_va_args);
      nfev = nfev + columns(fjac);
      %# search in N until GN step exceeds the trust
      %# region established by the standard error estimate.
      Pk = pinv(fjac(1:N,:)'*fjac(1:N,:));
```

```
        temp = Pk*fjac(1:N,:)'*fx1(1:N);
        theta0 = temp;          %# remember where we started
        N = N + 1;
        ek = fx1(N,:) - fjac(N,:)*temp;
        Pk = Pk - ((Pk*fjac(N,:)'*fjac(N,:)*Pk)/
                                (1+fjac(N,:)*Pk*fjac(N,:)'));
        temp = temp + Pk*fjac(N,:)'*ek;
        while (N < Nlim && norm(seinv*(temp-theta0)) <= 1)
          N = N + 1;
          ek = fx1(N,:) - fjac(N,:)*temp;
          Pk = Pk-((Pk*fjac(N,:)'*fjac(N,:)*Pk)/
                                 (1+fjac(N,:)*Pk*fjac(N,:)'));
          temp = temp + Pk*fjac(N,:)'*ek;
        end;
        %# update the residual norm
        fnorm1 = norm(fx1(1:N));
      end;
      fvec  = fx1(1:N);
      fnorm = fnorm1;
      info = "";                    %# reset info...
      iter = 1;
    end;
  end;
end;
```

APPENDIX C

OCTAVE CODE FOR LM

Main Program: lmsvd

```
%# Copyright (C) 2001 Montana State University
%#
%# This program is free software; you can redistribute it and/or
%# modify it under the terms of the GNU General Public License
%# as published by the Free Software Foundation; either version 2
%# of the License, or (at your option) any later version.
%#
%# This program is distributed in the hope that it will be useful,
%# but WITHOUT ANY WARRANTY; without even the implied warranty of
%# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%# GNU General Public License for more details.
%#
%# You should have received a copy of the GNU General Public License
%# along with this program; if not, write to the Free Software
%# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
%# 02111-1307, USA.
%#
%# Contact info:
%#
%# sshaw@alum.mit.edu
%# mkeppler@montana.edu
%#
%# S. R. Shaw and M. Keppler
%# ECE Department
%# 610 Cobleigh Hall
%# Bozeman, MT 59715
```

```
%#
function [x,info,nfev] = lmsvd(f,x,D,...)
  %# Initialize variables
  %# Make sure x is a column vector
  if(size(x,1) < size(x,2))
    x = x';
  end;
  opt = goptions();                          %# exit options
  %# opt(14) sets max iterations, Default is 100*length(x)
  if opt(14) == 0
    opt(14) = 100 * length(x);
  end;
  alpha = 0;
  info = "";
  fvec = feval(f,x,all_va_args);
  nfev = 1;
  fnorm = norm(fvec);
  mode = (D == []);
  iter = 1;
  %# begin outer loop
  while (info == "")
    %# evaluate the Jacobian and compute its SVD
    [fjac,fvec] = fdjac(f,x,fvec,all_va_args);
    [u,s,v] = svd(fjac,0);
    jcnorms = sqrt(diag(v*s*s*v'));          %# column norms
    utf = u'*fvec;
    nfev = nfev + columns(fjac);
```

```
%# on first iteration compute r, D, and initialize step
if iter ==  1
  if mode == 1
    D = diag(jcnorms + (jcnorms == 0));
  end;
  xnorm = norm(D*x);
  step = opt(18)*xnorm;
  if step == 0
    step = opt(18);
  end;
end;
%# compute norm of the scaled gradient
gnorm = 0;
if fnorm != 0
  temp = (diag(s) == 0);
  temp = v*diag((1-temp)./(diag(s)+temp))*utf/fnorm;
  %# if column norm = zero, corresponding x is zero
  %# otherwise, scale by column norm
  temp = abs(temp.*(jcnorms != 0)./(jcnorms+(jcnorms==0)));
  gnorm = max([gnorm; temp]);
end;
%# check if gradient norm is less than opt(4)    (gtol)
if (gnorm < opt(4))
  info = sprintf("gradient norm = %.3e, less than gtol = %.3e",
                 gnorm, opt(4));
  break;
end;
```

```
      %# Rescale D
      if mode == 1
        D = diag(max([diag(D)'; jcnorms']));
      end;
      %# begin inner loop
      ratio = 0;
      while (ratio < .0001)
        %# determine LevenburgMarquardt parameter
        [u1,s1,v1] = svd(s*v'*diag(1./diag(D)),0);
        [delta,alpha] = lsvd(step, .1, alpha, v1, s1, u1'*utf);
        pnorm = norm(delta);
        delta = (diag(1./diag(D)))*delta;
        %# determine new x and store in x1
        delta = -1*delta;
        x1 = x + delta;
        %# on the first iteration adjust the initial step bound
        if iter == 1
          step = min([step pnorm]);
        end;
        %# evaluate f at x1 and calculate the residual
        fx1 = feval(f,x1,all_va_args);  %# current value of fvec
        fnorm1 = norm(fx1);
        nfev = nfev + 1;
        %# compute the scaled actual reduction
        actred = -1;
        if (.1*fnorm1 < fnorm)
actred = 1 - (fnorm1/fnorm)^2;
```

```
end;
%# compute scaled predicted reduction & directional derivative
temp1 = norm(s*v'*delta)/fnorm;
temp2 = (sqrt(alpha)*pnorm)/fnorm;
predred = temp1*temp1 + 2*temp2*temp2;
dirder = -(temp1*temp1 + temp2*temp2);
%# calculate ratio of actual to predicted reduction
ratio = 0;
if predred != 0
  ratio = actred/predred;
end;
%# Update the step bound
if (ratio > .25)
  if (alpha == 0 || ratio >= .75)
    step = pnorm*2;
    alpha = .5*alpha;
  end;
else
  if (actred >= 0)
    temp = .5;
  else
    temp = .5*dirder/(dirder + .5*actred);
  end;
  if (.1*fnorm1 >= fnorm || temp < .1)
    temp = .1;
  end;
  step = temp*min([step pnorm*10]);
```

```
  alpha = alpha/temp;
end;
%# if iteration successful save x, fvec, and calculate norm x
if ratio >= .0001
  x = x1;
  fvec = fx1;
  xnorm = norm(D*x);
  fnorm = fnorm1;
  iter = iter + 1;
end;
%%%% tests of convergence %%%%
%# is actual and predicted reduction less than opt(3)?   (ftol)
if(abs(actred) <= opt(3) && predred <= opt(3) && .5*ratio <= 1)
  if (step <= opt(2)*xnorm)
    info = sprintf("actual and predicted reduction less
                    than %.3e and relative step less than
                    %.3e", opt(3),opt(2));
  else
    info = sprintf("actual and predicted reduction less
                    than %.3e", opt(3));
    break;
  end;
end;
%# is step in parameter space less than opt(2)*norm(x)?
if step <= opt(2)*xnorm
  info = sprintf("maximum relative step less than
                  %.3e", opt(2));
```

```
        break;
      end;
      %# too many iterations?
      if nfev >= opt(14)
        info = sprintf("number of function evaluations exceeds
                        %d",opt(14));
        break;
      end;
    end;
  end;
end;
```

APPENDIX D

SHARED SUBROUTINES

Subroutine: fdjac

```
%# Copyright (C) 2001 Montana State University
%#
%# This program is free software; you can redistribute it and/or
%# modify it under the terms of the GNU General Public License
%# as published by the Free Software Foundation; either version 2
%# of the License, or (at your option) any later version.
%#
%# This program is distributed in the hope that it will be useful,
%# but WITHOUT ANY WARRANTY; without even the implied warranty of
%# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%# GNU General Public License for more details.
%#
%# You should have received a copy of the GNU General Public License
%# along with this program; if not, write to the Free Software
%# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
%# 02111-1307, USA.
%#
%# Contact info:
%#
%# sshaw@alum.mit.edu
%# mkeppler@montana.edu
%#
%# S. R. Shaw and M. Keppler
%# ECE Department
%# 610 Cobleigh Hall
%# Bozeman, MT 59715
```

```
%#
function [fjac,fvec] = fdjac(f, theta, fvec, ...)
  %# find the appropriate step size scale factor
  opt = goptions();
  small = opt(16);
  if nargin < 3 || fvec == []
    fvec = feval(f,theta,all_va_args);
  end;
  %# initialize fjac
  fjac = zeros(size(fvec,1),
       size(theta,1));
  %# evaluate Jacobian
  for i = 1:size(fjac,2)                    %# for each column
    smu = theta(i);                         %# save the element
    h = min([max(abs([small small*smu])) opt(17)]);
    theta(i) = theta(i) + h;                %# increment
    fjac(:,i) = (feval(f,theta,all_va_args)-fvec) / h;
    theta(i) = smu;                         %# restore element
  end;
end;
```

Subroutine: lsvd

```
%# Copyright (C) 2001 Montana State University
%#
%# This program is free software; you can redistribute it and/or
%# modify it under the terms of the GNU General Public License
%# as published by the Free Software Foundation; either version 2
%# of the License, or (at your option) any later version.
%#
%# This program is distributed in the hope that it will be useful,
%# but WITHOUT ANY WARRANTY; without even the implied warranty of
%# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%# GNU General Public License for more details.
%#
%# You should have received a copy of the GNU General Public License
%# along with this program; if not, write to the Free Software
%# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
%# 02111-1307, USA.
%#
%# Contact info:
%#
%# sshaw@alum.mit.edu
%# mkeppler@montana.edu
%# S. R. Shaw and M. Keppler
%#
%# ECE Department
%# 610 Cobleigh Hall
%# Bozeman, MT 59715
```

```
%#
function [x,alpha] = lsvd(delta,sigma,alpha,v,s,utf)
  %# Compute the OLS solution.  If s is rank deficient (within
  %# machine precision) then obtain a solution that minimizes the
  %# norm of x.
  sinv = diag(s);
  x = (abs(sinv) == 0);
  rankdeficient = sum(x);
  sinv = diag((1-x) ./ (sinv+x));
  x = v*sinv*utf;
  %# Check to see if we can accept this solution, i.e.
  %# is the ||x|| < delta+sigma*delta?
  nx = norm(x);
  tol = sigma*delta;
  fp = nx-delta;
  if fp < tol
    alpha = 0;
    return;
  end;
  %# The least-square solution didn't work, so an iteration
  %# to find a regularizing alpha is necessary.  Establish
  %# bounds for the regularization parameter.
  %# Initialize lower bound --> lower bound less than epsilon
  %# means that ||z(alpha)|| may *exceed* our minimum norm
  %# solution above.
  if rankdeficient > 0
    L = eps;
```

```
else
  z = v'*x;
  phi_prime = -(z'*sinv*sinv*z) / nx;
  L = ((delta-nx) / phi_prime);
end;
%# Initialize upper bound.
sts = s'*s;
stutf = s'*utf;
gnorm = norm(stutf);
U = (gnorm/delta) - min(diag(sts));
%# check alpha
alpha = min([U max([L alpha])]);
if alpha == 0
  alpha = gnorm / nx;
end;
%# Iterate to find a new alpha
for iter = 1:10
  %# compute new x with current alpha
  temp = diag(1./(diag(sts)+alpha));
  z = temp*stutf;
  fp = norm(z)-delta;
  %# check exit criteria
  if (abs(fp) <= tol || (abs(fp) <= temp || fp < 0
      && alpha == eps))
    x = v*z;
    return;
  end;
```

```
    %# compute update to alpha
    phi_prime = -(z'*temp*z)/(fp+delta);
    parc = -((fp+delta)/delta)*(fp/phi_prime);
    %# update the bounds
    if(fp > 0)
      L = max([L alpha]);
    else
      U = min([U alpha]);
    end;
    %# calculate new alpha, within the bounds.
    alpha = min([max([L alpha+parc]) U]);
  end;
  %# update x
  x = v*z;
end;
```

Subroutine: goptions

```
%# Copyright (C) 2000, 2001 Montana State University
%#
%# This program is free software; you can redistribute it and/or
%# modify it under the terms of the GNU General Public License
%# as published by the Free Software Foundation; either version 2
%# of the License, or (at your option) any later version.
%#
%# This program is distributed in the hope that it will be useful,
%# but WITHOUT ANY WARRANTY; without even the implied warranty of
%# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%# GNU General Public License for more details.
%#
%# You should have received a copy of the GNU General Public License
%# along with this program; if not, write to the Free Software
%# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
%# 02111-1307, USA.
%#
%# Contact info:
%#
%# sshaw@alum.mit.edu
%# mkeppler@montana.edu
%#
%# S. R. Shaw and M. Keppler
%# ECE Department
%# 610 Cobleigh Hall
%# Bozeman, MT 59715
```

```
%#
function opts = goptions(optnumber, value)
  global goptions_defaults_ = [0 1e-4 1e-4 sqrt(eps) 0 0 0 0 0 0 0
                               0 0 0 0 sqrt(eps) .1 10];
  global goptions_ = goptions_defaults_;

  %# reset to defaults
  if nargin == 1
    goptions_ = goptions_defaults_;
  end;

  if nargin == 2
    goptions_(optnumber) = value;
  end;

  opts = goptions_;
end;
```

APPENDIX E

UPDATING THE LM PARAMETER

The ACM and LM update the parameter estimate $\hat{\theta}$ with a Levenberg-Marquardt iteration that utilizes the SVD of the Jacobian [27]. The parameter estimate $\hat{\theta}$ and the Levenberg-Marquardt parameter α are updated in the subroutine *lsvd.m* (see appendix D). The purpose of *lsvd.m* is to compute an α that produces a step $D\delta_{\hat{\theta}}$ such that

$$\sqrt{\alpha} \;<\; \text{eps}, \tag{6.1}$$

$$\| D\delta_{\hat{\theta}} \| \;\leq\; \eta, \tag{6.2}$$

or

$$\sqrt{\alpha} \;>\; \text{eps}, \tag{6.3}$$

$$|\| D\delta_{\hat{\theta}} \| - \delta \;| \;\leq\; \sigma\eta, \tag{6.4}$$

where η is the step bound, σ is the desired relative error in the step bound, D is a weighting matrix, and eps is the machine precision. In (6.2) α is zero within the machine precision and $D\delta_{\hat{\theta}}$ is the ordinary least squares solution. The Levenberg-Marquardt step is generally determined by solving the matrix equation

$$\begin{pmatrix} J \\ D\sqrt{\alpha} \end{pmatrix} \delta_{\hat{\theta}} = \begin{pmatrix} r \\ 0 \end{pmatrix} \tag{6.5}$$

for α in the least square sense so the error in the normal equation

$$\| J^T\delta_{\hat{\theta}} - r \| + \alpha \| D\delta_{\hat{\theta}} \| \tag{6.6}$$

is minimized. The subroutine *lsvd.m*, on the other hand, solves the matrix equation

$$\begin{pmatrix} JD^{-1} \\ I\sqrt{\alpha} \end{pmatrix} D\delta_{\hat{\theta}} = \begin{pmatrix} r \\ 0 \end{pmatrix} \tag{6.7}$$

because it simplifies the computations required to determine α. A change of variables $y = D\delta_{\hat{\theta}}$ yields

$$\begin{pmatrix} JD^{-1} \\ I\sqrt{\alpha} \end{pmatrix} y = \begin{pmatrix} r \\ 0 \end{pmatrix}. \tag{6.8}$$

This matrix equation is solved using the SVD of JD^{-1}. If the SVD of JD^{-1} is given by $usv^T = JD^{-1}$ (6.8) becomes

$$\begin{pmatrix} usv^T \\ I\sqrt{\alpha} \end{pmatrix} y = \begin{pmatrix} r \\ 0 \end{pmatrix}. \tag{6.9}$$

The normal equations of (6.9) are

$$(vs^T u^T usv^T + \alpha I)y = vs^T u^T r. \tag{6.10}$$

Pulling v and v^T outside of the parenthesis and taking advantage of the fact that u is orthogonal, the normal equation becomes

$$v(s^T s + v^T \alpha I v)v^T y = vs^T u^T r. \tag{6.11}$$

Since α is a constant, I is the identity matrix, and v is orthogonal (6.11) can be rewritten

$$v(s^T s + \alpha I)v^T y = vs^T u^T r. \tag{6.12}$$

Multiplying both sides of (6.12) on the left by v^T and solving for y gives

$$y = v(s^T s + \alpha I)^{-1} s^T u^T r. \tag{6.13}$$

The variable y is the solution of the matrix equation (6.9) for any given value of α. If $\alpha = 0$, y is the ordinary least squares solution. If the least squares solution does not meet the criteria

$$\| y \| \leq \eta, \tag{6.14}$$

a new $\alpha \neq 0$ is computed. To simplify the computation of α a new variable

$$z = v^T y = (s^T s + \alpha I)^{-1} s^T u^T r \tag{6.15}$$

is introduced. The variable z can be used to compute α because v is orthogonal which implies $\| y \| = \| z \|$. The Levenberg-Marquardt parameter α is obtained from the derivative

$$\frac{d \| z \|}{d\alpha}. \tag{6.16}$$

The derivative in (6.16) is computed by finding

$$\frac{d \parallel z \parallel^2}{d\alpha} \tag{6.17}$$

and using the property that

$$\frac{d \parallel z \parallel^2}{d\alpha} = 2z\frac{d \parallel z \parallel}{d\alpha}. \tag{6.18}$$

Solving the derivative in (6.17) and applying (6.15) yields

$$\frac{d \parallel z \parallel^2}{d\alpha} = \sum \frac{d}{d\alpha}\left[\left(\frac{s_i}{s_i^2+\alpha}\right)u_i^T r_i\right]^2 \tag{6.19}$$

$$= \sum -2\frac{s_i u_i^T r_i}{s_i^2+\alpha}\frac{s_i u_i^T r_i}{(s_i^2+\alpha)^2} \tag{6.20}$$

$$= -2z^T(s^T s+\alpha I)^{-1}z. \tag{6.21}$$

The derivative in (6.16) is found by substituting (6.21) into (6.18) to obtain

$$\frac{d \parallel z \parallel}{d\alpha} = \frac{-z^T(s^T s+\alpha I)^{-1}z}{\parallel z \parallel} \tag{6.22}$$

which is used to compute α.

The subroutine *lsvd.m* computes the Levenberg-Marquardt step $D\delta_{\hat{\theta}}$ iteratively. The iterative aproach proceeds by first computing the ordinary least squares solution. This is equivalent to solving for $D\delta_{\hat{\theta}}$ with $\alpha = 0$. If the step $D\delta_{\hat{\theta}}$ satisfies the criteria in (6.1) and (6.2) the step is returned to the main program. If the step does not satisfy (6.1) and (6.2), the step z (6.15) is computed using an updated α obtained from (6.22). The Levenberg-Marquardt parameter α is updated ten times or until z satisfies (6.3) and (6.4). Once an appropriate step z is computed, it is multiplied by v to obtain the weighted step in parameter space $D\delta_{\hat{\theta}}$ for the given $\alpha \neq 0$. After returning to the calling routine, the step returned from *lsvd.m* $D\delta_{\hat{\theta}}$ is multiplied by D^{-1} to produce the Levenberg-Marquardt step $\delta_{\hat{\theta}}$.