

https://dspace.org/

Utilising Local Model Neural Network Jacobian Information in Neurocontrol

Carrelli, David John

2006-11-16T13:01:07Z
http://hdl.handle.net/10539/1815

Downloaded from a DSpace repository. https://dspace.org/

UTILISING LOCAL MODEL NEURAL NETWORK JACOBIAN INFORMATION IN NEUROCONTROL

Author:

David John Carrelli

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirement for the degree of Master of Science in Engineering.

Johannesburg, March 2005

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____

(Signature of candidate.)

On this the _____ day of _____ in the year _____

To Bertha and June

ACKNOWLEDGEMENTS

I am indebted to those people who have provided inspiration, support and the many comments and suggestions that have made this dissertation possible. My supervisor, Professor Brian Wigdorowitz, deserves special mention for his continual interest, unfailing patience, and belief in my abilities during those times when progress was lacking or hindered. I would also like to express my appreciation for the financial assistance provided to me by the Foundation for Research and Development (FRD), and the University of the Witwatersrand Senior Bursary schemes, during a portion of the time in which this research was conducted.

ABSTRACT

In this dissertation an efficient algorithm to calculate the differential of the network output with respect to its inputs is derived for axis orthogonal Local Model (LMN) and Radial Basis Function (RBF) Networks. A new recursive Singular Value Decomposition (SVD) adaptation algorithm, which attempts to circumvent many of the problems found in existing recursive adaptation algorithms, is also derived. Code listings and simulations are presented to demonstrate how the algorithms may be used in on-line adaptive neurocontrol systems. Specifically, the control techniques known as series inverse neural control and instantaneous linearization are highlighted. The presented material illustrates how the approach enhances the flexibility of LMN networks making them suitable for use in both direct and indirect adaptive control methods. By incorporating this ability into LMN networks an important characteristic of Multi Layer Perceptron (MLP) networks is obtained whilst retaining the desirable properties of the RBF and LMN approach.

TABLE OF CONTENTS

INTRODUCTION **1**
1.1 BACKGROUND 1
1.2 OBJECTIVE 2
1.3 SCOPE 3
1.4 RESULTS AND CONCLUSIONS 3

INTRODUCTION TO NEURAL NETWORKS **5**
2.1 INTRODUCTION 5
2.2 ARTIFICIAL NEURAL NETWORK SYSTEM COMPONENTS 5
2.3 THE BIOLOGICAL CONNECTION 8
2.4 THE FEED FORWARD NEURAL NETWORK APPROXIMATION PROBLEM 9
2.4.1 Problem definition 9
2.4.2 Existence of a uniform approximation 10
2.4.3 Approximation Construction 11
2.4.4 Interpolation of the approximation 12
2.4.5 Contending with the curse of dimensionality and using *a-priori* information 17
2.5 DISCUSSION AND CONCLUSIONS 23

NEURAL NETWORKS - STRUCTURE AND IMPLEMENTATION **26**
3.1 INTRODUCTION 26
3.2 NETWORK STRUCTURE - THE ACTIVATION RULE 26
3.2.1 Multi-Layer Perceptron Network Structure 26
3.2.2 Radial Basis Function Network Structure 29
3.2.3 Local Model Network Structure 35
3.3 TRAINING THE NETWORK PARAMETERS – THE LEARNING RULE 40
3.3.1 The Optimisation Problem 41
3.3.2 On-line vs Batch Mode Processing 42
3.3.3 Network Parameterisation as a Linear Optimisation Problem 43
3.3.4 The Steepest or Gradient Descent Method 44
3.3.5 The Recursive Least Squares (RLS) Method 45
3.3.6 The Exponential Forgetting Factor with Conditional Updating Method 47
3.3.7 Regularization - The Constant Trace and Kalman Filter Methods. 53
3.3.8 The Recursive Singular Value Decomposition (SVD) Algorithm 54
3.3.9 Implementation of the Learning Rule 61
3.3.10 Structure Optimisation 66
3.4 DETERMINING THE NETWORK JACOBIAN 67
3.4.1 Determining MLP Network Jacobian Information 67
3.4.2 Determining RBF Jacobian Information 67
3.4.3 Determining LMN Jacobian Information 72
3.5 CONCLUSION 74

CONTROL USING NEURAL NETWORKS - NEUROCONTROL **77**
4.1 INTRODUCTION 77
4.2 HISTORICAL BACKGROUND 78
4.3 PLANT DESCRIPTION 79
4.4 CONTROL LAW FORMULATION 81
4.4.1 Series Inverse Control 81
4.4.2 Minimum Degree Pole Placement Design 83
4.5 ADAPTATION MECHANISMS 88
4.5.1 The adjusted parameters 88
4.5.2 Incorporating the design specification 90

4.5.3 Changing the parameters 91
4.5.4 Using the Neural Network Jacobian Information 93
4.6 CONCLUSIONS 94

SIMULATIONS 97
5.1 INTRODUCTION 97
5.2 INVERSE NEURAL CONTROL 98
5.2.1 System Description 98
5.2.2 Linear State Feedback control 99
5.2.3 Series inverse control using direct adaptation 100
5.2.4 Series inverse control using indirect sensitivity adaptation 102
5.3 CONTROL USING INSTANTANEOUS LINEARIZATION 103
5.3.1 Controlling a non-linear Mass-Spring-Damper System 103
5.3.2 Stabilising an inverted pendulum 111
5.4 DISCUSSION AND CONCLUSIONS 115
5.4.1 A Priori Plant information 115
5.4.2 Identification in the presence of disturbances 116
5.4.3 The network verification step 117
5.4.4 Stability 117
5.4.5 Miscellaneous issues 117

CONCLUSIONS AND RECOMMENDATIONS 119
6.1 SYNOPSIS 119
6.2 OBSERVATIONS AND CONCLUSIONS 121
6.3 RECOMMENDATIONS FOR FUTURE WORK 122

REFERENCES 125

TABLE OF FIGURES

Figure 2.2.1. Neural Network Components. 6
Figure 2.3.2. A Generalised Neuron Structure. 8
Figure 3.2.1. Multi-Layer Perceptron Network Structure. 27
Figure 3.2.2. General RBF Network Structure. 29
Figure 3.2.3. Axis Orthogonal RBF Network Example. 31
Figure 3.2.4. General LMN Network Structure. 36
Figure 3.5.1. Neural Network Function Identification Computation Flow Chart. 76
Figure 4.5.1. Block diagram of a Model Reference Adaptive System. 89
Figure 4.5.2. Block diagram of an indirect Self Tuning Regulator. 90
Figure 5.2.1. Non-Linear First Order Plant - Open Loop System Model and its Response. 98
Figure 5.2.2. State Feedback System Model and its Closed Loop Responses. 99
Figure 5.2.3. Inverse Neural Controller System Model and its Closed Loop Responses. 100
Figure 5.2.4. Comparison of Inverse and Linear State Feedback Controller Mappings. 102
Figure 5.2.5. Indirect MLP Inverse Controller and its Closed Loop Responses. 103
Figure 5.3.1. Open Loop Response of a Non-Linear Mass-Spring-Damper. 106
Figure 5.3.2. Identification System Model for Non-Linear Mass-Spring-Damper. 107
Figure 5.3.3. Polynomial Coefficients of the Non-Linear Mass-Spring-Damper. 108
Figure 5.3.4. Pole Zero Plot for Mass-Spring-Damper System. 109
Figure 5.3.5. Instantaneous Linearization Control of Mass-Spring-Damper. 109
Figure 5.3.6. Mass-Spring-Damper System -Closed Loop Responses. 110
Figure 5.3.7. The Inverted Pendulum Problem. 111
Figure 5.3.8. System Models used for Instantaneous Linerization Control of an Inverted Pendulum. 113
Figure 5.3.9. Instantaneous Linearization Control of an Inverted Pendulum - Closed Loop Responses. 115

NOMENCLUTURE

Variable	Description
$\boldsymbol{u}$	Plant inputs.
$\boldsymbol{y}$	Plant outputs and / or network outputs.
$\boldsymbol{e}_j$	Standardized basis vector along the j^{th} dimension of an n-dimensional hyper surface.
N, N_o	Scalar variable representing number of network hidden units.
l	For MLP networks: An integer layer index. For RBF and LMN networks: The integer number of activated basis function units. Also equal to the number of elements contained in ξ_{I_0}.
m_i	The number of units contained in the i^{th} layer of an MLP network.
n	Number of elements in the network information vector. ($n = pn_b + qn_a$)
n_a, n_b, n_k	Maximum number of unit delays for the plant output elements, plant input elements, and plant transport respectively, contained in the network information vector.
p, q	Number of plant inputs and plant outputs represented in the information vector respectively.
$\boldsymbol{x}$	Network information vector or plant state vector.
θ, Θ	Network weight or vector / matrix of weights.
$\xi_{i,j}, \xi_I$	Elliptical integration function centres. Each centre may be regarded as an element of the set of lattice points ξ_L.
σ, Σ	Elliptical integration function variance as a scalar variables and matrix respectively.
Δ_i	Lattice sample spacing along the i^{th} dimension.

Functions	Description
$g(.)$	Activation function or model validity function.
$\iota(.)$	Integration function or net function.
$\delta(.)$	Dirac impulse function.
$\text{col}(.)$	Function that sequentially stacks all the columns of the matrix argument.
$\text{diag}(.)$	Function that constructs a null matrix of appropriate size and places its vector argument along the main diagonal.

Set	Description
$\Re^{a\times b}$	The set of Real numbers. Superscript designated number of dimensions that member variables may take on.
$\aleph^{a\times b}$	The set of Natural numbers. Superscript designated number of dimensions that member variables may take on.

Set	Description
K	Closed and bounded subset representing the domain of the information vector having n dimensions.
L	$L = \{i_1, \ldots, i_n\}$ is an n-tuple of integers representing the locations of the lattice points contained in ξ_L.
ξ_L	Set representing the spatial lattice point vertices on an n-dimensional hyper surface.
ξ_{I_0}	Set representing the spatial lattice point vertices on an n-dimensional hyper surface that contributes to a network output. In general $\xi_I \in \xi_{I_0} \subseteq \xi_L$.

Symbology	Description
Italics	Designates a variable when used in an equation.
Bold Italics	Generally designates a vector variable.
UPPERCASE ITALICS	Generally designates a matrix variable.
UPPERCASE	Generally designates a matrix function.
`Courier`	Designates computer code.
[.]	Indicates discrete time indexes or matrix elements.
$[.]^T$	Matrix or vector transpose.
$\otimes$	Kronecker tensor product. If A and B are matrices having dimension $m \times n$ and $p \times q$ respectively then $A \otimes B$ is the $mp \times nq$ matrix formed by taking all possible products between the elements of A and B. That is: $A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Over the last decade or more, there has been substantial interest in the use of both supervised and unsupervised neural networks to solve non-linear control problems. This has led to a new class of techniques collectively referred to as neurocontrol. Many different methods have been proposed (Psaltis *et al*, 1988; Hunt and Sbarbaro, 1991; Hunt *et al*, 1992; Sorensen, 1994; Suyken *et al*, 1996; van Breeman and Veelenturf, 1996; Soloway and Haley 1997; Motter, 1998) however, all systematic engineering approaches have considered three crucial and interrelated steps.

First, a fundamental strategy or control paradigm must be selected. This can be broadly subdivided into model based or model free approaches, both of which may or may not be adaptive, deterministic or optimal. This decision is influenced by factors such as the control objective, the amount of information available about the process being controlled and the environmental constraints (noise, computing resources, cost, etc.) in which the system is to operate.

Second, the type of neural network(s) to be used in implementing the control paradigm must be selected. To do this, a thorough understanding of the different network types and their fundamental characteristics is essential. These characteristics may include (in no particular order) but are not necessarily limited to:

1. Approximation abilities:
   - Can the network approximate the necessary functions?
   - Generalisation - How well does the network approximate data not in the training set?
   - Over fitting - Does the network tend to over fit the training set?
   - Does the network provide the information required by the selected control paradigm?
2. Identification issues:
   - Number, rate and stability of parameter convergence.
   - Suitability for on-line adaptation. (If required)
   - Discrete time vs. continuous time systems.
3. Physical correlation and suitability to theoretical analysis:
   - Can the network parameters be related to the physical systems involved?
   - Is it possible to include *a priori* information in the network?
   - Can the trained network be "reverse engineered" and analysed with current mathematical tools?
4. Computational requirements:
   - Calculation complexity.
   - Scalability and memory requirements.

As is typical of most engineering problems, no single network paradigm has favourable characteristics in all the above categories, and any network design is thus the result of a trade off between these characteristics.

Third, the conditions under which the closed loop system is stable must be determined. This step is obviously highly dependent on the preceding two points and is a fundamental challenge in the application of neural networks to control. Obviously, successful implementation of this step is closely related to the mathematical foundations of the networks involved. It is therefore most desirable to implement designs that use networks with sound theoretical underpinnings and strong correlations to the physical attributes of the system dynamics. This observation was an important motivating factor behind the work presented in this dissertation.

There are essentially two basic classes of neural networks commonly used in supervised neurocontrol, the Multi-Layer Perceptron (MLP) and Radial Basis Function (RBF) networks. The former typically scores well in approximation abilities, identification issues, and computational requirements. In particular, the differential of the network output with respect to its inputs (termed system gradient or network Jacobian in this dissertation), is readily obtained in an on-line fashion, with little computational overhead. This last feature is most desirable in all forms of model based control. However, the MLP is poorly suited to theoretical analysis using current mathematical tools and there is no easily recognisable correlation between its parameters and the physical system it models. Basic RBF networks are more theoretically tractable but have been criticised for inferior memory use, bad generalisation, a tendency to over fit training data, and their inefficiency in calculating system gradient information on-line.

Local Model Networks (LMN) (Johansen and Foss, 1992, 1993; Bosman, 1996; Zbikowski *et al*, 1994; Murray-Smith and Johansen, 1997) are closely related to basic RBF networks but provide a good compromise between the two extremes mentioned above. Structurally, they attempt to address the problems of inferior memory use, bad generalisation and over fitting while still maintaining the tractability of basic RBF networks. However, since they are based on RBF network principles, they do not appear to be widely used in control methods where on-line system gradient information is required.

1.2 OBJECTIVE

The properties of LMN networks make them most amenable in the analysis of neurocontrol systems. If the system gradient of such networks could be efficiently computed then, from a systems point of view, they could be used as "black-box" replacements for MLP networks. Perhaps the most important attribute though, is that the LMN networks remain transparent to the designer when performing closed loop system analysis or when attempting to include *a priori* plant information. Therefore, the main objectives of this work are:

1) To derive an algorithm to efficiently calculate the system gradient of a LMN.

2) Implement this algorithm in an on-line manner that can be used in non-linear adaptive neurocontrol.

3) Show how this algorithm enhances the flexibility of the LMN permitting its use in both model free and model based neurocontrol applications.

1.3 SCOPE

To achieve the stated objectives the subject matter of this dissertation has been arranged into three main areas, namely, the relevant neural network theory and implementation, the basic approaches to neurocontrol and finally simulation results.

We begin in chapter two by defining what a neural network is (in the context of this work) and presenting selected results from approximation theory with explanations of how these results are relevant to the neural network problem.

In chapter three the specific structural and algorithmic details of the MLP, RBF and LMN networks are presented. This chapter constitutes the main body of the work and discusses in detail the calculation of the network output, the adaptation or training of the network parameters and evaluation of the network Jacobian or system gradient for each the three network types. It is here that the proposed system gradient algorithm is developed, first for RBF networks, and then by extending this result to LMN networks. Although not formally part of the objectives, a new on-line adaptation algorithm utilising a new and interesting objective function formulation and Singular Value Decomposition (SVD) approach is also developed and presented.

Following this, in chapter four, the reader is introduced to the topic of neurocontrol. This is a large topic, in which the majority of the details are beyond the scope of this dissertation. The focus has thus been placed on defining a systematic approach to the topic by covering selected areas of system description and identification, adaptation, and controller structure. Two methods in particular have been highlighted: inverse neural adaptive control and instantaneous linearization. The penultimate section of this chapter highlights the connection between the network Jacobian algorithm of chapter three and the neurocontrol approaches described in chapter four.

In chapter five attention is turned to demonstrating the use of the described theory by presenting a number of simulations. These simulations show the LMN networks used in an on-line environment performing both inverse neural adaptive control and instantaneous linearization control. The chapter concludes with a discussion on the selected control techniques, highlighting possible difficulties and pitfalls.

1.4 RESULTS AND CONCLUSIONS

In the sixth and final chapter discussion, conclusions and recommendations for future research are presented, however, for completeness a very brief summary of the results and conclusions are presented here.

All the research objectives were successfully met and demonstrated for LMN networks. Additionally, algorithms for extracting Jacobian information were derived for MLP and RBF networks and contrasted with the algorithm obtained for LMN networks. Furthermore, a new adaptation algorithm, which attempts to circumvent some of the problems with existing online adaptation schemes, was proposed.

Unfortunately a number of complications were encountered when using network Jacobian information in a neurocontrol setting. These include the following basic problems. The network identification must be highly accurate to achieve good control as the system root locations are highly sensitive to the network parameters.

Additionally, knowledge of the plant delays and number of zeroes is highly desirable to obtain good plant descriptions. These characteristics are embedded in the polynomial description used for the plant, suggesting that other approaches, such as sub-space methods, for the plant description may be better. Finally, the online adaptation algorithms used to perform network training are prone to numerical difficulties and their setup can, in practice, be problematic.

While successful results can be achieved, these characteristics mean that users must be cognizant of the potential pitfalls, requiring that they have a thorough understanding of the underlying algorithms. This is particularly true of the online adaptation computations.

CHAPTER 2

INTRODUCTION TO NEURAL NETWORKS

2.1 INTRODUCTION

Artificial Neural Networks[1](ANN) systems may be viewed as general information processing systems. As such, their analysis may be broken down into three distinct levels of description. The first, computational ability, defines the goal that the system is required to achieve. The second level, the algorithmic level, specifies the mathematical equations or formulas that can be instantiated in some physical way to achieve the computational goals. Finally, there is the level concerned with the physical implementation of the algorithms themselves. This chapter is concerned mainly with the computational abilities of neural networks and pays little attention to algorithmic or implementation issues.

Before the computational abilities of a system can be clarified, the basic components of the system under consideration must be defined. Therefore, in the next section, a very broad description of an artificial neural network and its role in an ANN system will be given. This is followed by a rudimentary description of the biological elements that inspired the development of such networks providing the reader a context in which to place the terminology frequently used in connection with artificial neural networks. Next, the feed forward neural modelling problem is formulated and a number of relevant results from mathematical approximation theory are presented. This supports the validity of the selected approaches by demonstrating that the computational abilities of the networks under consideration are sufficient to achieve a valid solution. Finally, surrounding issues and chapter conclusions are presented.

2.2 ARTIFICIAL NEURAL NETWORK SYSTEM COMPONENTS

ANN systems are comprised of five basic elements. The first element is the components of the network itself (Figure 2.2.1), which may be an abstraction (such as in a computer program) or physical hardware. This consists

[1] The term "Artificial" is used to differentiate the networks described in this work from their true biological counterparts. All the networks studied here are approached from a signal processing and approximation theory point of view. As will be seen in later sections, their structure bears a striking resemblance to their biological cousins. This is undoubtedly due to the original computation goal of attempting to devise networks which mimic certain brain behaviour. As biological networks are not the topic of concern, the term "neural networks" is synonymous with "artificial neural networks" in this work.

of a collection of units or neurons, each of which, at any instant in time, has a state or activation level represented by a real valued scalar. All the unit activations in a network are collectively referred to as the activation state of the network. The units are interconnected by a number of connections, which essentially define the structure of a network. In the ANN such structure is commonly in the form of unit layers. The connection "strengths" define the degree to which the activation of any particular unit influences another. This "connection strength" is determined by the network parameters or weights.

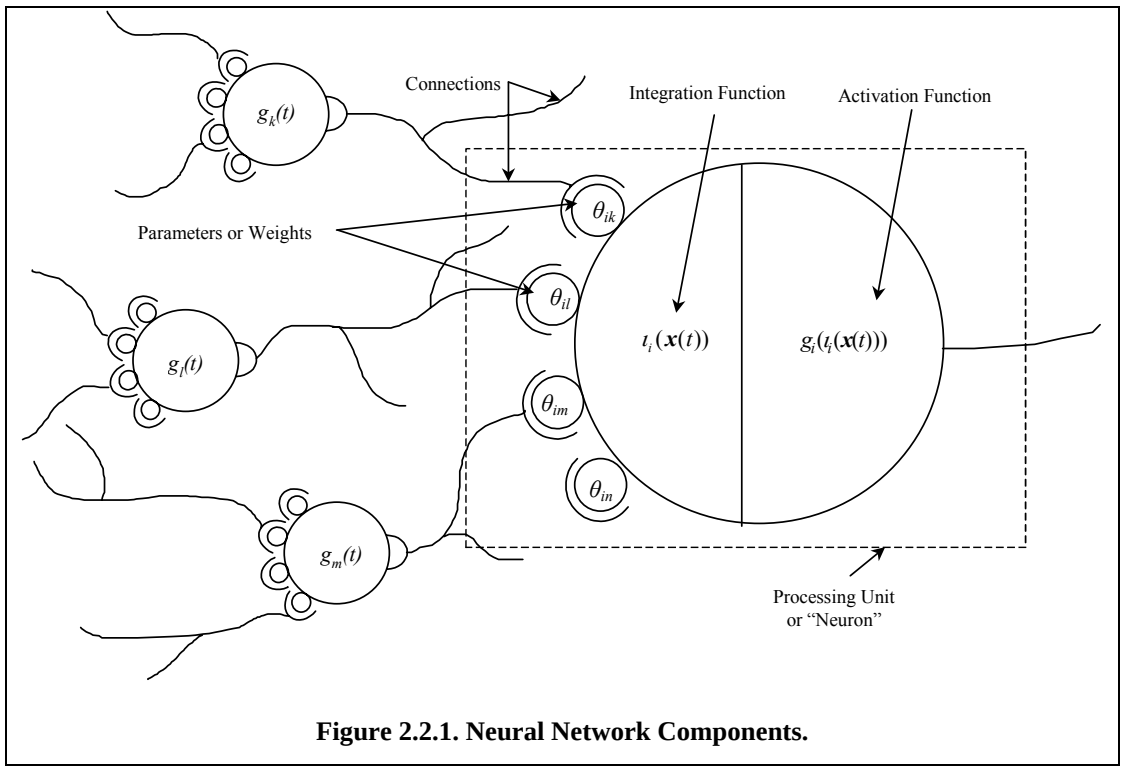


Figure 2.2.1. Neural Network Components.

The second major element is the environment in which the ANN operates. It specifies what type of inputs and outputs are observed by the network during its operation. In this work, these observations are the on-line sampled data obtained from the process we wish to control and / or the signal to be tracked.

The third element is an activation-updating rule that describes how the network activation state is updated at each moment in time. It is usually described by a large system of non-linear differential or, as in this work, difference equations. This rule not only constrains the structure of the connections between the units, it also is clearly related to the process performed by the units themselves. Mathematically, the unit activations are related to the unit input connections in many varying ways. Generally, it may be divided into a two-stage process described by $g(\iota(\boldsymbol{x}(t)))$ where the result is a unit's output to the environment or its activation level, and $\boldsymbol{x}$ is the unit inputs from the environment, and / or the activation level from other connected units. The function $g(.)$ is referred to as the activation function, and $\iota(.)$ is called the integration or net function. Some examples of commonly used integration functions include linear:

$$\iota(\boldsymbol{x}) = \sum_{i=1}^{n} \theta_i x_i \; - b = \Theta^T \boldsymbol{x} - b \,, \tag{2.2.1}$$

quadratic:

$$\iota(\boldsymbol{x}) = \sum_{i=1}^{n} \theta_i x_i^2 - b = \boldsymbol{x}^T \Theta\ \boldsymbol{x} - b, \quad \Theta = \begin{bmatrix} \theta_1 & & 0 \\ & \theta_2 & \\ 0 & & \theta_n \end{bmatrix}, \tag{2.2.2}$$

and elliptical (or spherical):

$$\iota(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{ij} (x_i - \xi_i)(x_j - \xi_j) - b = (\boldsymbol{x} - \xi)^T \Sigma (\boldsymbol{x} - \xi) - b\ . \tag{2.2.3}$$

The purpose of the integration function, which is typically fixed for an entire network, is to combine the unit inputs with the network parameters in some meaningful way. In the above equations Θ, Σ, ξ and b all represent the network parameters. The parameter b plays a particular role referred to as the bias value. This value basically defines the "baseline" activation for the unit.

As with the integration function, many types of activation functions are also used, including signum and its variations such as the threshold functions:

$$g(\boldsymbol{x}) = \begin{cases} 1 & \iota(\boldsymbol{x}) > 0 \\ 0 & \iota(\boldsymbol{x}) = 0 \\ -1 & \iota(\boldsymbol{x}) < 0 \end{cases}, \tag{2.2.4}$$

piecewise linear or entirely linear:

$$g(\boldsymbol{x}) = \begin{cases} 1 & \iota(\boldsymbol{x}) > 1 \\ \iota(x) & 0 \le \iota(\boldsymbol{x}) \le 1 \\ 0 & \iota(\boldsymbol{x}) < 0 \end{cases}, \tag{2.2.5}$$
$$g(\boldsymbol{x}) = a\,\iota(\boldsymbol{x})$$

sigmoid, both unipolar and bipolar (equivalent to the hyperbolic tangent function):

$$g(\boldsymbol{x}) = a\left(\frac{1}{1 + e^{-\iota(\boldsymbol{x})/T}} - 0.5\right) + c\ , \tag{2.2.6}$$

and exponential which is used with the elliptical integration function to give a Gaussian function:

$$g(\boldsymbol{x}) = e^{-\iota(\boldsymbol{x})}\ . \tag{2.2.7}$$

The activation function's purpose is to generate the activation level of the unit given the integration function value. It may differ from unit to unit, within the same network, to provide the desired behaviour for any particular region of the network. A common combination is for a network's input and output units; those units whose activation is directly obtained from and sent to the environment respectively, to be linear, while all other so called hidden units are non-linear. These structures will be discussed in detail in the next chapter.

The fourth major element is the interpretation and objective functions. The interpretation function translates a given networks activation state (distributed representation) or particular unit activation levels (local representation) to some semantic interpretation. (i.e. It maps system states to physical interpretations.) The type of function used varies, dependent on the ANN system's intended purpose. In this work, the interpretation function is a scale factor (usually unity), which maps the network output unit activations to values which

represent some physical signal within the process we are attempting to control. Coupled with the interpretation function is the objective function. This function maps system variables or states into a real number, whose magnitude reflects how well the system is achieving its computation goal.

The fifth and final element is the learning or adaptation rule. This rule derives from a non-linear optimisation problem, and forms a system of differential or difference equations that define a dynamic process that determines how the network parameters are updated as a function of the system's current and past experiences. The optimisation problem is defined by the objective function together with the network constraints and environment. Essentially all ANNs attempt to combine these five elements in such a way as to solve some meaningful problem.

2.3 THE BIOLOGICAL CONNECTION

There are numerous types of neural cells and structures found in the central nervous system. Cells found in the sensory organs, and various parts of the brain, are frequently specialised. However, some neural cells exhibit regular physical characteristics. These anatomical regularities were, on occasion, the inspiration for some of the fundamental units found in artificial neural network models (Albus, 1971, 1981; Stevens, 1985). Clearly, the neuron shown in Figure 2.3.2 is not anatomically correct, but, hypothetically it is sufficiently accurate to describe the principal features of most neural cells. The reader is referred to Rumelhart and McClelland (1986), Eberhart and Dobins (1990), Anderson and Rosenfeld (1988), and Anderson *et al* (1990) for more detailed information.

The neuron can be broken down into three main sections; the cell body or soma, the dendritic tree, and the axon. The soma contains those constituents common to most cells such as the nucleus, and substances necessary for metabolism and protein synthesis. The intracellular liquid, cytoplasm and various other particles fill the entire cell. The exact nature of the functions taking place in this area of the cell is unclear. Conceptually it will be assumed that any information processing the neuron performs takes place here.

In order for the neuron to make contact with many other neurons, (as many as 10^5 in certain cells) the outer cell membrane is shaped into many branches called dendrites, making up what is termed the dendritic tree. The

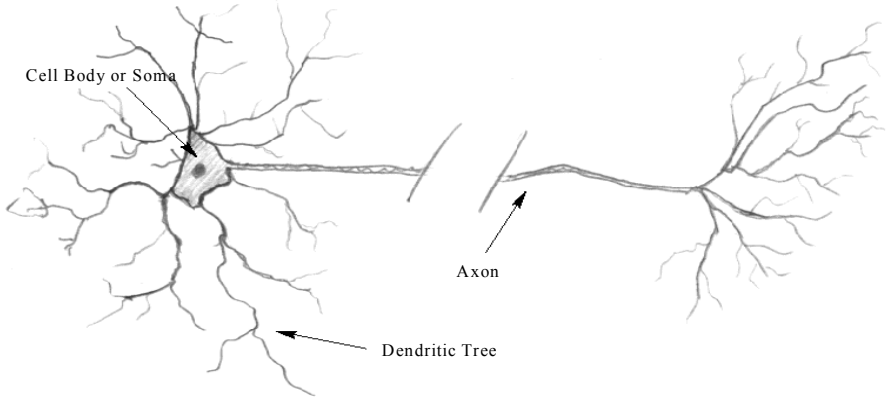


Figure 2.3.2. A Generalised Neuron Structure.

shape size and structure of the dendritic tree varies dramatically depending on the cell type and presumed function.

The neuron sends its output signal to other neurons via the axon. These output signals take the form of a series of electrical impulses. A resting potential of about 70 mV is present across the cellular membrane. By a process of selective ion diffusion impulses of about 100 mV in amplitude and approximately 0.5 ms to 2 ms in duration can be generated. The output information of the neuron is usually encoded in the frequency of these pulses, which can travel at speeds of up to 100 m/s down the axon depending on the diameter and tissue covering it.

The connection between the output axons and dendrites of neural cells occurs in special formations called synapses. The mechanisms involved in these synapses are complicated biophysical electrochemical processes. Although there are still many questions surrounding these processes, a few basic facts can be stated. There are two fundamental types of synapses, namely, excitatory and inhibitory, and to a reasonable first approximation, it can be stated that the effects of the synapses on the membrane potential can be summed up linearly.

Unfortunately, the elementary structural features of a neuron give very little indication as to the workings of the neuron itself. Functionally the neuron can best be described as a dynamic multi-input, single output device with memory nonlinearities. This description encompasses a variety of extremely complex behaviour. As emphasised by Lewis (1983), the neuron is an exceptionally complex machine, and warrants extensive study in its own right. (Deutsch, 1983 and Niznik, 1983.)

From a systems viewpoint the human brain is estimated to contain more than 10^{12} neurons each having as many as 10^3 synaptic junctions. This suggests there are approximately 10^{15} potentially modifiable connections; a formidable system to analyse.

2.4 THE FEED FORWARD NEURAL NETWORK APPROXIMATION PROBLEM

Numerous authors have studied the problem discussed in the next four sections over a considerable length of time. The details presented here are a small sampling showing only the most important results. The explanations and theorems presented here closely follow the work in Zbikowski *et al.* (1994), (Sanner and Slotine, 1992), and (Johansen and Foss, 1993). The interested reader is referred to these references for a more substantial list of citations and in depth discussion.

2.4.1 PROBLEM DEFINITION

Before we can confidently use a neural network in control system design, it is necessary to establish their approximation abilities given a particular control or modelling environment. We therefore begin by defining the environment in which we expect the ANN to perform.

The use of feed forward neural networks (FNNs) in control is primarily based on the fact that many non-linear systems can be described by the discrete time I/O representation shown in equation (2.4.1) below.

$$y[k] = f(y[k-1], \ldots, y[k-n_a], u[k-n_k], \ldots, u[k-n_k-n_b]) + e[k] \qquad (2.4.1)$$

Here, $y[k]$ is the system output, $u[k]$ is the system input, and $e[k]$ is a zero mean disturbance term. The system inputs and outputs are sampled with unit sample time and each sample is identified by the time index *k*. The values n_a, n_k, and n_b represent a fixed number of unit delays. The function *f*(.) is minimally assumed to be continuous. This type of system model is called a NARX (Non-linear Auto-regressive model with eXogenous inputs) model and has been widely studied in non-linear systems identification. For notational convenience the information vector $\boldsymbol{x}$ is defined as:

$$\boldsymbol{x}[k] = [y[k-1], \ldots, y[k-n_a], u[k-n_k], \ldots, u[k-n_k-n_b]]^T \quad \boldsymbol{x} \in \Re^n, n = (pn_b + qn_a). \tag{2.4.2}$$

This permits the system to be written as:

$$y[k] = f(\boldsymbol{x}[k]) + e[k]. \tag{2.4.3}$$

The problem addressed by the FNN is therefore to find a parameterised structure which emulates the non-linear function *f*(.) given a finite number of I/O samples. The input to the network would thus be the information vector while the output is an estimation of the plant output. We represent this by:

$$\hat{y}[k] = \hat{f}(\boldsymbol{x}[k]). \tag{2.4.4}$$

Mathematically we may define the FNN problem more concisely as follows; given a continuous mapping[2] $f : K \to \Re^q$, find a representation of *f* by means of known functions, and a finite number of real parameters, such that the representation yields a uniform approximation of *f* over *K*. The set *K* is an uncountable compact (closed and bounded) subset of $\Re^{pn_b+qn_a}$ where *p* is the number of inputs, *q* the number of outputs, and n_b and n_a are, respectively, the number of input and output samples, for each input and output in $\boldsymbol{x}$. The mapping *f* is given by a finite number of sample pairs $(U_k, Y_k) \in K \times \Re^q, k = 1, \ldots, s$ where *s* is the number of observed input output pairs. This problem definition consists of two major elements. Firstly, the network must perform a uniform approximation of *f* on *K*, and secondly, this approximation must be obtained by an interpolation of *f*(*K*) from the samples (U_k, Y_k). To solve these problems we refer to a number of results from approximation theory.

2.4.2 EXISTENCE OF A UNIFORM APPROXIMATION

The first result is the Stone - Weierstrass theorem. This theorem provides a relatively simple set of criteria which given functions must satisfy in order to uniformly approximate an arbitrary set of continuous functions on a compact set *K*. Before stating the theorem the following definitions are presented:

Definition 1 *- A set A of functions from* $K \subset \Re^{pn_b+qn_a}$ *to* $\Re$ *is called an* **algebra of functions** *iff* $\forall f, g \in A$ *and* $\forall \alpha \in \Re$ *:*

(i) $f + g \in A$;

(ii) $fg \in A$;

---

[2] For simplicity, only the single output case (i.e. *q*=1) for the function *f* shall be considered, however, all the results presented also hold for *q*>1.

(iii) $\alpha f \in A$. ◊

Definition 2 *- A set A of functions from* $K \subset \Re^{pn_b+qn_a}$ *to* $\Re$ *is said to* **separate points on** *K iff* $\forall x_1, x_2 \in K \quad x_1 \neq x_2 \Rightarrow \exists f \in A, f(x_1) \neq f(x_2)$. ◊

Definition 3 *- Let A be a set of functions from* $K \subset \Re^{pn_b+qn_a}$ *to* $\Re$. *We say that A* **vanishes at no point** on *K iff* $\forall x \in K \; \exists f \in A$, *such that* $f(x) \neq 0$. ◊

Definition 4 *- Let B be the set of all functions which are limits of uniformly convergent sequences with terms in A, a set of functions from* $K \subset \Re^{pn_b+qn_a}$ *to* $\Re$. *Then B is called the* **uniform closure** of *A*. ◊

Theorem 1 (Stone - Weierstrass) *- Let A be an algebra of some continuous functions from a compact set* $K \subset \Re^{pn_b+qn_a}$ *to* $\Re$, *such that A separates points on K and vanishes at no point of K. Then the uniform closure B of A consists of all continuous functions from K to* $\Re$. ◊ (See Zbikowski *et al.* (1994) for proof citations.)

Thus the uniform approximation of an arbitrary continuous mapping $f : K \to \Re$, may be constructed from some other function, if the set of all finite linear combinations of that function, is a non-vanishing algebra, separating all points on the compact set $K \subset \Re^{pn_b+qn_a}$, as specified in definitions 1,2 and 3. Although this result is useful in determining whether a particular function can be used to approximate a mapping, it provides no indication of what forms such a function may take.

2.4.3 APPROXIMATION CONSTRUCTION

A result which provides an indication of how an approximation may be constructed is Kolmogorov's well known representation theorem:

Theorem 2 (Kolmogorov) *- Any function continuous on the n-dimensional cube* κ *can be represented in the form*

$$f(\boldsymbol{x}) = \sum_{i=1}^{2n+1} \chi_i \left(\sum_{j=1}^{n} \phi_{ij}(x_j) \right) \tag{2.4.5}$$

where $\boldsymbol{x} = [x_1, \cdots, x_n]^T$, *and* χ_i *and* ϕ_{ij} *are real continuous functions of one variable.* ◊ (See Zbikowski *et al.* (1994) for proof citations)

Notice that the representation is *exact* and is constructed of a finite number of continuous functions. A number of authors have reformulated the main representation theorem in ways that make its direct application to neural networks more plausible, however, the approach suffers from practical limitations. The functions ϕ_{ij} may be independent of $f(\boldsymbol{x})$ but may be highly non-smooth. Furthermore the functions χ_i are specific to the given function *f* and may not be representable in a parameterised form.

In neural networks however, an *approximation* of the function *f* is sufficient. Kurková has shown that when staircase-like sigmoidal functions are used, Kolomogorov's theorem may be reformulated to show that any continuous function on any closed interval can be approximated to within an arbitrary accuracy. Thus given:

Definition 5 *- A C^k sigmoid function $g : \Re \to \Re$ is a non-constant, bounded and monotone increasing function of class C^k (continuously differentiable up to order k).* ◊

The following theorem may be stated:

Theorem 3 (Kurková) *- Let $n \in \aleph$ with $n \geq 2$, $g : \Re \to E$, $E = [0,1]$ be a sigmoidal function, $f \in C^0(E^n)$, and ε be a positive real number. Then there exists $k \in \aleph$ and staircase-like functions $\chi_i, \phi_{ij} \in S(g)$ such that for every $\boldsymbol{x} \in E^n$*

$$\left| f(\boldsymbol{x}) - \sum_{i=1}^{k} \chi_i \left(\sum_{j=1}^{n} \phi_{ij}(x_j) \right) \right| < \varepsilon \qquad (2.4.6)$$

where $S(g)$ is the set of all staircase-like functions of the form $\sum_{i=1}^{k} a_i g(b_i x + c_i)$. ◊ (See Zbikowski *et al.* (1994) for proof citations.)

This result implies that a four-layered sigmoidal network may be used to *approximate*, with arbitrary error, any continuous function. Others have established, not necessarily by Kolmorgorov's argument, that only three layers are sufficient for approximation of general continuous functions. Although three layers are sufficient, from a practical perspective this construction may not be optimal due to the potentially large number of units required by the hidden layer. In the next chapter it is shown how this result serves as the theoretical foundation for the construction of the Multi Layer Perceptron (MLP) network, a sigmoidal feedforward network with an arbitrary number of hidden layers.

2.4.4 INTERPOLATION OF THE APPROXIMATION

In the previous section the question of uniform approximation was addressed. However, the second question of how to interpolate the approximation given a finite set of samples has not been considered. As stated in section 2.4.1, for practical purposes, the mapping from the compact set $K \subset \Re^{pn_b+qn_a}$ by $f : K \to \Re^q$ is given by a finite number of samples $(U_k, Y_k) \in K \times \Re^q, k = 1, \ldots, s$ where *s* is the number of observed input output pairs, even though the domain *K* and the corresponding hyper surface *f(K)* are, in general, a continuum. We therefore wish to interpolate the approximation in order to reconstruct *f* from the given number of samples. Neural network terminology refers to this as a networks ability to generalise.

It is important to note that in the single dimension case $f : \Re \to \Re$ that this is the same problem faced by the signal processing community in trying to reconstruct a signal from its samples. Shannon's well known Sampling Theorem states:

Theorem 4 (Shannon Sampling Theorem) *- Let* $f : \Re \to \Re$ *be such that both its direct (F) and inverse Fourier transforms are well defined. If the spectrum* $F(\omega)$ *vanishes for* $|\omega| > 2\pi\beta$*, then f can be exactly reconstructed from its samples* $\{f(t_k)\}_{k\in Z}$, $t_k = k/2\beta$. $\Diamond$ (See Zbikowski *et al.* (1994) for proof citations.)

Although the above theorem deals explicitly with one-dimensional functions of time, it may be applied with equal effectiveness to multidimensional functions of space. The question of existence of an *exact* solution to the interpolation problem, for functions that are both band limited and whose direct and inverse Fourier transform are well defined, is thus positively answered. To gain insight into how one might use this result to construct a network that satisfies our goal we examine the reconstruction process in more detail.

Consider a function $f(\boldsymbol{x})$, $x \in \Re^n$ that is absolutely integrable (i.e. $\int |f(\boldsymbol{x})| d\boldsymbol{x} < \infty$) and whose spatial Fourier transform has compact support. The sampling of *f* on an *n*-dimensional uniformly spaced square lattice can be interpreted as the modulation of the hypersurface *f* with a field of Dirac distributions, thus:

$$f_s(\boldsymbol{x}) = f(\boldsymbol{x}) \sum_{L\in Z^n} \delta(\boldsymbol{x} - \xi_L) \tag{2.4.7}$$

where the lattice $\xi_L = \{\Delta(i_1\boldsymbol{e}_1 + i_2\boldsymbol{e}_2 + \cdots + i_n\boldsymbol{e}_n)\}$ has sample spacing[3] Δ along each of the standard basis vectors $\boldsymbol{e}_j$ for $\Re^n$, and $L = \{i_1,\ldots,i_n\}$ is an *n*-tuple of integers. Fourier transforming (2.4.7) and performing the resulting convolution gives:

$$F_s(\nu) = \frac{1}{\Delta} \sum_{I\in Z^n} F(\nu - \zeta_L) . \tag{2.4.8}$$

Thus, the spectrum of the sampled function consists of copies of the original spectrum centred at the frequency lattice points $\zeta_L = \{\Delta^{-1}(i_1\boldsymbol{e}_1 + i_2\boldsymbol{e}_2 + \cdots + i_n\boldsymbol{e}_n)\}$. Now let $\kappa(\beta)$ be the smallest *n*-cube $[-\beta,\beta]^n$, $\beta > 0$, centred at the origin, which completely encloses the support for $F(\nu)$. Let $G_c(\nu)$ be the spectrum of the canonical reconstructor function that is equal to Δ on $\kappa(\beta)$ and zero elsewhere. If $\Delta \le 1/(2\beta)$ then the copies of $F(\nu)$ in the sampled spectrum $F_s(\nu)$ will be non-overlapping and $G_c(\nu)$ can be used to extract the original spectrum:

$$F(\nu) = F_s(\nu) G_c(\nu) . \tag{2.4.9}$$

The structure of equation (2.4.9) suggests that other interpolating functions can be used in the reconstruction process. To see this let $g_1(\boldsymbol{x})$ be an interpolating function whose spectrum is bounded, real valued and strictly positive on $\kappa(\beta)$. Additionally its spectrum $G_1(\nu)$, vanishes outside of some *n*-cube κ_Δ, which completely contains $\kappa(\beta)$ where the size of κ_Δ is determined by setting $\Delta < 1/(2\beta)$. (This is the same as oversampling the hypersurface.) Now define a new function $c(\boldsymbol{x})$, whose spectrum obeys the relationship:

$$C(\nu) = \Delta F(\nu) G_1^{-1}(\nu) . \tag{2.4.10}$$

---

[3] The sample spacing may be different for each input dimension. To reduce complexity the sample spacing is assumed constant in all dimensions. This does not reduce the generality of the analysis as each input may be scaled to achieve the same effect as varying the sample spacing. This phenomenon will be exploited later in this work.

Under these conditions it is easy to verify that:

$$F(\nu) = C_s(\nu)G_1(\nu) \tag{2.4.11}$$

where $C_s(\nu)$ is the sampled spectrum of the bounded continuous function $c(\boldsymbol{x})$. Inverse transforming this equation results in an *exact* expansion of $f(\boldsymbol{x})$:

$$\begin{aligned} f(\boldsymbol{x}) &= c_s(\boldsymbol{x}) * g_1(\boldsymbol{x}) \\ &= \sum_{L \in Z^n} c(\xi_L) g_1(\boldsymbol{x} - \xi_L). \end{aligned} \tag{2.4.12}$$

Forfeiting the exactness of equation (2.4.12) for an *approximation* of $f(\boldsymbol{x})$ over some chosen compact set *K,* designated by $\hat{f}(\boldsymbol{x})$, it is possible to apply this technique to a much larger class of functions while using a wider range of interpolating functions. The accuracy of such an approximation may be represented by an equation summing five contributing error terms:

$$\begin{aligned} \left| f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) \right| &= \alpha_1 + \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \alpha_2 \\ &= \alpha + \varepsilon_f. \end{aligned} \tag{2.4.13}$$

The first term α_1, represents the error introduced by forcing (if necessary) the condition that $f(x)$ is globally absolutely integrable. This is achieved by multiplying $f(x)$ by an infinitely smooth function $m_1(\boldsymbol{x})$ which is unity on *K* and decays to zero more rapidly than $|f|$ outside of *K*. Therefore:

$$\alpha_1(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in K \\ \left| f(\boldsymbol{x}) - m_1(\boldsymbol{x}) f(\boldsymbol{x}) \right|, & \boldsymbol{x} \in K^c \end{cases} \tag{2.4.14}$$

where K^c is the complement of the set *K* in $\Re^n$. The second term ε_1, is the error resulting from the truncation of the spectrum of $m_1(\boldsymbol{x})f(\boldsymbol{x})$ at the value determined by the bounds of $\kappa(\beta)$ in the frequency domain. Provided the aforementioned spectrum is absolutely integrable on $\Re^n$ (a requirement satisfied by the fact that the approximated function is continuous over the set *K*) the value of ε_1 may be made arbitrarily small by choosing a sufficiently large spectral truncation radius β.

The term ε_2 is the error attributed to reconstruction using an interpolating function that is not an ideal low-pass filter. As the degree of overlap experienced by repeating spectra may be controlled by the chosen sample spacing of the mesh, the value of ε_2 may be made arbitrarily small by judicious selection of $\Delta \le 1/(2\beta)$. The relationship:

$$\mu = \frac{1}{2\Delta\beta} \tag{2.4.15}$$

may be conveniently used to represent the degree to which the surface is oversampled.

The final two terms ε_3 and α_2 arise from the need to truncate the reconstruction series using a finite number of terms. By construction, let the term $\alpha_2 = 0, \forall \boldsymbol{x} \in K$ and, when $\boldsymbol{x} \in K^c$, let α_2 equal the upper bound of the approximation error resulting from the truncated terms. Then, the error when $\boldsymbol{x} \in K$, ε_3, may be controlled, at

each $\boldsymbol{x}$, by omitting the terms corresponding to samples ξ_L which lie outside an *n*-ball of radius ρ surrounding $\boldsymbol{x}$. For convenience ρ may be expressed as a multiple of the mesh size:

$$\rho = l\Delta . \tag{2.4.16}$$

Clearly this truncation radius will be highly sensitive to the specific interpolating function used.

If we combine all the contributing factors mentioned above, and let $\alpha = \alpha_1 + \alpha_2$ and $\varepsilon_f = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$ we see that for $\boldsymbol{x} \in K$ the entire error associated with the approximation:

$$\hat{f}(\boldsymbol{x}) = \sum_{I \in I_o} c(\xi_I) g_1(\boldsymbol{x} - \xi_I) \quad I_o = \left\{ I \;\middle|\; |\boldsymbol{x} - \xi_I| \le \rho, \boldsymbol{x} \in K, \xi_I \in \xi_L \right\} \tag{2.4.17}$$

is contained in the term ε_f. Clearly, as $\boldsymbol{x}$ moves outside the set *K*, the error, represented by α, may increase rapidly. This rapid degradation of the approximation must be considered when the approximation is to be used in the implementation of a control law that must exhibit global stability.

Although the above discussion has not solved the problem of which function $g_1(\boldsymbol{x} - \xi_I)$ would be the best, it has shown that the function must satisfy the conditions of Stone - Weierstrass theorem (section 2.4.2), be bounded, strictly positive and absolutely integrable on $\Re^n$. A class of functions that meets these requirements is the Radial Basis Functions (RBF):

Definition 6 *- A C^k radial basis function $g_{\xi,m} : \Re^n \to \Re$, with $\xi \in \Re^n$ and $m \in \Re_+$, is a C^k function constant on spheres $\left\{ \boldsymbol{x} \in \Re^n \;\middle|\; \|\boldsymbol{x} - \xi\|_2 / m = r \right\}$, centre ξ, radius $mr \in \Re_+$, where $\|.\|_2$ is the Euclidean norm on $\Re^n$.* ◊

Research using regularisation theory (Poggio and Girosi, 1990) has demonstrated that, under certain constraints a linear superposition of Gaussian radial basis functions is the optimal solution to a class of function approximation problems, given a finite set of data points in $\Re^n$.

Radial Gaussian functions may be expressed as:

$$\begin{aligned} g(\boldsymbol{x} - \xi) &= e^{-\left(\frac{\pi \|\boldsymbol{x} - \xi\|^2}{\sigma_x^2} \right)} \\ &= e^{-\left(\frac{\pi (\boldsymbol{x} - \xi)^T (\boldsymbol{x} - \xi)}{\sigma_x^2} \right)}. \end{aligned} \tag{2.4.18}$$

This function posses a number of desirable properties that make them particularly amenable to network construction:

(i) The functional form does not change when undergoing a forward or inverse Fourier transform. Thus, as shown by the transform pair, $e^{\left(-\pi \sigma_\nu^{-2} \nu^T \nu\right)} \xleftrightarrow{F} \sigma_x^{-n} e^{\left(-\pi \sigma_x^{-2} \boldsymbol{x}^T \boldsymbol{x}\right)}$ where $\sigma_\nu = \sigma_x^{-1}$, variance[4] parameters in the spatial domain have a direct counterpart in the frequency domain. The low-pass spectral characteristics can thus be easily controlled by the variance in the spatial domain.

[4] The variance subscript has been used to emphasis in which domain the variance, as it is typically defined, applies.

(ii) They may be easily stretched or shrunk along any particular direction in $\Re^n$ permitting an adjustment of the spectral support along a chosen direction. Such adjustments may also rotate the spectral support by the use of cross coupling terms, resulting in the more general representation $G(\nu) = e^{-(\pi \nu^T \Sigma \nu)}$ where Σ is a positive definite matrix.

(iii) The structure is uniform and independent of the number of dimensions *n*.

(iv) It is a separable nonlinearity, i.e. $g(\boldsymbol{x} - \xi) = g(x_1 - \xi_1) g(x_2 - \xi_2) \cdots g(x_n - \xi_n)$, permitting individual subspaces of $\Re^n$ to be transformed separately, then multiplied together to form the final approximation.

If $g_1(\boldsymbol{x} - \xi_I) = g(\boldsymbol{x} - \xi_I)$ and the required spectral support β is large then, as alluded to in (ii) above, σ_ν^2 must also be large so that the profile of $G(\nu)$ is broad enough to cover all the frequencies contained in $\kappa(\beta)$. Conversely, if β is small the variance may also be reduced. Noting the reciprocal nature of the variances between the spatial and frequency domains results in the following intuitively pleasing observation. Approximations of highly smooth functions (limited high frequency content) may be achieved using a sparse array of Gaussians with wide profiles. Conversely, less smooth functions require more densely packed Gaussians with narrower profiles.

To formalise the relationship between β and σ_ν^2 consider the effect of choosing a small variance when β is large. Under these conditions $G(\nu)$ approaches zero as ν increases towards β. Using equation (2.4.10) it becomes clear that this results in large magnitude and substantial high frequency components in the weighting function $c(\boldsymbol{x})$ in equation (2.4.12). By replacing β for ν in the expression for $G(\nu)$ we observe that the order of $G^{-1}(\nu)$ may be maintained close to unity by setting $\sigma_\nu^2 = n\pi\beta^2$. The dependence on the dimension *n* can be removed if we assume that $\max|C(\nu)|$ is inside the ball of radius β in $\Re^n$ resulting in:

$$\sigma_\nu^2 = \sigma_x^{-2} = \pi\beta^2 \tag{2.4.19}$$

Sanner and Slotine (1992) have shown, in the case of a square sampling lattice ξ_L, with parameters chosen according to the conditions specified in equations (2.4.15), (2.4.16), (2.4.18) and (2.4.19), that the terms contributing to $\varepsilon_f \ \forall \boldsymbol{x} \in K$ in equation (2.4.13) may be expressed as:

$$\begin{aligned}
\varepsilon_1 &\le \int_{\nu \notin \kappa(\beta)} \left| F\left(m_1(\boldsymbol{x}) f(\boldsymbol{x})\right) \right| d\nu \\
\varepsilon_2 &\le \left[eF_{\max} \left(\sqrt{\pi}\beta\right)^n \right] \left[1 - \operatorname{erf}^n(2\mu - 1) \right] \\
\varepsilon_3 &\le \left[eF_{\max} \left(\sqrt{\pi}\beta\right)^n \right] \mu^{-n} \sum_{j \in J} 2^{j+1} \binom{n}{j} \sum_{m=l+1}^{\infty} m^j e^{-\left(\frac{\pi m}{2\mu}\right)^2}
\end{aligned} \tag{2.4.20}$$

where $J = \left\{ 0 \le j \le n-1 \,\middle|\, n - j \text{ is odd} \right\}$, $\operatorname{erf}(\mu)$ is the error function and $F_{\max}$ is the upper bound of $\left| F\left(m_1(\boldsymbol{x}) f(\boldsymbol{x})\right) \right|, \quad \forall \boldsymbol{x} \in K$.

From the arguments presented in this section we have seen that, given a finite (undetermined) number of samples the function $\hat{f}(\boldsymbol{x})$, constructed according to equation (2.4.17), results in an approximation that is both uniform and interpolated. Furthermore, the approach provides us with a method that uses minimal information about the approximated function, to construct a square lattice of gaussian radial basis functions such that the resulting approximation is accurate within some target tolerance ε_f. Such a design procedure may be summarised by the following steps:

1. Determine the function's input dimension *n* and lattice extremities from *K*.
2. Estimate or determine the functions smoothness information $F_{\max}$ and β from given data.
3. Ensure or assume that the spectral support β is sufficiently large such that $\varepsilon_1 \leq \varepsilon_f / 3$.
4. Select the lattice spacing Δ, then calculate μ and ε_2 using equations (2.4.15) and (2.4.20) respectively, ensuring that ε_2 is sufficiently small.
5. Select a truncation radius ρ and use equation (2.4.16) to obtain *l*, thus permitting the calculation ε_3. Ensure that $\varepsilon_1 + \varepsilon_2 + \varepsilon_3 \leq \varepsilon_f$ adjusting β, Δ or ρ in steps 3, 4 or 5 as necessary.
6. Finally calculate the gaussian radial basis function variances using equation (2.4.19) and the centres using the lattice spacing Δ.

This approximation method leads to a class of neural networks generally referred to as radial basis function (RBF) networks and their algorithmic instantiation is discussed further in the next chapter.

2.4.5 CONTENDING WITH THE CURSE OF DIMENSIONALITY AND USING *A-PRIORI* INFORMATION

The discussion in the previous section outlines a powerful and usable solution to the feed forward neural network approximation problem however it suffers from two practical drawbacks.

The first problem, revealed by careful examination of equation (2.4.17), is that the number of coefficients and basis functions increases exponentially as the system dimension *n* is increased. This is the well-known *Curse of Dimensionality* problem evident in many approximation schemes. As each element of the vector $\boldsymbol{x}$ in equation (2.4.2) increases the dimension of the domain of the approximating function by one, this problem may rapidly become a severe limitation, particularly in MIMO systems. The problem can be reduced if the approximated function can be separated into lower dimension component functions, each of which is then approximated individually and then summed to form the final result. Similarly, if the radial basis function used (such as the Gaussian) is separable, then the number of RBF computations can be made to scale linearly with the number of dimensions however, there is still a multiplication term which scales exponentially. This problem is further exacerbated by the fact that, in order to guarantee the approximation accuracy over the entire domain, the spectral support must be sufficiently wide to account for the most rapidly changing regions of the function. This results in a small spatial mesh size and thus many more node terms to the summation in equation (2.4.17).

Sanner and Slotine (1992) address this problem by a variety of techniques and extensions which attempt to contract the domain on which the approximation is required and reduce the required node density. Such techniques include dynamic network construction and dynamic modification of the basis function parameters such as the Gaussian centres and variances. The extensions discussed introduce the Gabor and Wavelet models, which make use of the simultaneous spatial and frequency properties of the Gaussian function to automatically tune the local bandwidth requirements thus circumventing the need to globally contract the node mesh.

An important feature to any model developer is the ability to include *a-priori* information into the modelling technique. This information frequently takes the form of a set of simplified state equations based on physical principles or a number of transfer functions representing a linear approximation of the systems behaviour around selected nominal operating points. Clearly another problem with both approximation techniques described thus far is the inability to gracefully 'build in' or absorb such information into the approximation construction. This results in an approximation mechanism that must undergo extensive 'training' before providing accurate model information. Conversely the ability to examine the internals of the final approximation and relate its parameters to the physical system under consideration is also highly desirable. This problem is frequently resolved by allowing the approximation to compute a residual in parallel with a more conventional (linear) model and summing the results. However, the disadvantage to this technique is the added complexity, and thus computational overhead, of running two models in parallel to each other.

Both the Gabor and Wavelet approaches mentioned earlier result in an approximation that may be interpreted as a summation of basis functions where each basis function is modulated in a manner that is dependent on the local properties of the function being approximated. Interestingly, another approximation approach analysed and developed by Johansen and Foss (1993) called Local Model Networks (LMN), may be interpreted in the same manner, however, unlike the more theoretical Gabor and Wavelet approaches, its design is highly amenable to physical interpretation and the inclusion of *a-priori* system information. The remainder of this section thus examines the approximation capabilities of this approach. A broad background discussion and analysis to the LMN approach can be found in Murray-Smith and Johansen (1997).

Let us choose a set of localised functions $\{g(\boldsymbol{x}-\xi_I) : K \to [0,1], \forall \xi_I \in \xi_L\}$ that are significantly larger than zero for $\boldsymbol{x} \in K_I, K_I \subset K$ and close to zero otherwise. As before the lattice $\xi_L = \{\Delta(i_1\boldsymbol{e}_1 + i_2\boldsymbol{e}_2 + \cdots + i_n\boldsymbol{e}_n)\}$ has sample spacing Δ along each of the standard basis vectors $\boldsymbol{e}_j$ for $\Re^n$, and $L = \{i_1,\ldots,i_n\}$ is an *n*-tuple of integers[5]. If not all $g(\boldsymbol{x}-\xi_I)$ vanish for $\boldsymbol{x} \in K$ and there exists a good local model $\hat{f}(\boldsymbol{x}-\xi_I)$ for $\boldsymbol{x} \in K_I$ then the following approximation may be written:

$$\hat{f}(\boldsymbol{x}) = \frac{\sum_{\xi_I \in \xi_L} \hat{f}(\boldsymbol{x}-\xi_I) g(\boldsymbol{x}-\xi_I)}{\sum_{\xi_I \in \xi_L} g(\boldsymbol{x}-\xi_I)}. \qquad (2.4.21)$$

[5] The analysis does not constrain the points ξ_I to exist on a uniformly sampled lattice, however this definition is sufficient to describe the cases of interest while simultaneously permitting the use of a consistent notation throughout.

The interpolation functions are defined as the normalisation of the localised functions $g(\boldsymbol{x}-\xi_I)$ giving:

$$\hat{g}(\boldsymbol{x}-\xi_I)=\frac{g(\boldsymbol{x}-\xi_I)}{\sum_{\xi_J\in\xi_L} g(\boldsymbol{x}-\xi_J)}. \tag{2.4.22}$$

Substituting (2.4.21) into (2.4.22) results in the following LMN approximation:

$$\hat{f}(\boldsymbol{x})=\sum_{\xi_I\in\xi_L}\hat{f}(\boldsymbol{x}-\xi_I)\hat{g}(\boldsymbol{x}-\xi_I). \tag{2.4.23}$$

If $f\in C^{p+1}$, the set $\left\{\hat{f}(\boldsymbol{x}-\xi_I)\right\}$ are local models equal to the first p terms of the Taylor series expansion of f about ξ_I and we assume $\left\|\nabla^{p+1}f(\boldsymbol{x})\right\|<M, \forall \boldsymbol{x}\in K$ where $\|\cdot\|$ denotes the induced operator norm, then appropriate substitution into the Lagrange form of the Taylor theorem remainder provides the following expression:

$$\left\|f(\boldsymbol{x})-\hat{f}(\boldsymbol{x}-\xi_I)\right\|_2<\varepsilon<\sum_{\xi_I\in\xi_L}\frac{M}{(p+1)!}\left\|\boldsymbol{x}-\xi_I\right\|_2^{p+1}\hat{g}(\boldsymbol{x}-\xi_I). \tag{2.4.24}$$

Here $\|\cdot\|_2$ denotes the Euclidean norm that we require being less than some arbitrary $\varepsilon>0$. If we define the set of functions:

$$\left\{z(\boldsymbol{x}-\xi_I):K\to\Re\,\middle|\, z(\boldsymbol{x}-\xi_I)=\left\|\boldsymbol{x}-\xi_I\right\|_2^{p+1}-\varepsilon\frac{(p+1)!}{M}, \forall\xi_I\in\xi_L\right\} \tag{2.4.25}$$

then simple rearrangement of equation (2.4.24) gives the following two equivalent conditions that must hold for this to be true:

$$\sum_{\xi_I\in\xi_L} z(\boldsymbol{x}-\xi_I)g(\boldsymbol{x}-\xi_I)<0 \tag{2.4.26}$$

$$\sum_{\xi_I\in\xi_L} z(\boldsymbol{x}-\xi_I)\hat{g}(\boldsymbol{x}-\xi_I)<0. \tag{2.4.27}$$

Careful observation of (2.4.25) reveals that the functions $z(\boldsymbol{x}-\xi_I)$ are dependent only on the accuracy of the local model. Furthermore, their *shape* is not influenced by the value of $\boldsymbol{x}$, the extent or density of the lattice ξ_L, or the form of the interpolation functions; however, the *value* of $z(\boldsymbol{x}-\xi_I)$ approaches a minimum that is less than zero[6], as $\boldsymbol{x}$ approaches ξ_I. The problem thus reduces to choosing ξ_L and $g(\boldsymbol{x}-\xi_I)$ such that equation (2.4.26) or (2.4.27) be satisfied for some arbitrary $\varepsilon>0$.

Equation (2.4.26) can be satisfied if any negative $z(\boldsymbol{x}-\xi_I)g(\boldsymbol{x}-\xi_I)$ term dominates the summation. We can guarantee that at least one $z(\boldsymbol{x}-\xi_I)$ is negative at any $\boldsymbol{x}$ if the set ξ_L is sufficiently large and dense in *K*. Since *K* is bounded a finite number of points in ξ_L is sufficient to achieve this. To state the condition that must hold

[6] The exception to this is when the local model is a perfect representation of the function being approximated. Under these circumstances the entire function $z(\boldsymbol{x}-\xi_I)$ is zero.

for ξ_L to be "sufficiently dense" the following definition, similar to the Haussdorf metric, of the distance between sets is introduced:

Definition 7 *- Assume A and B are two non-empty subsets of a vector space. Then the distance between the sets is defined as:*

$$D(A,B) = \inf_{a\in A} \sup_{b\in B} \|a-b\|_2 \,. \qquad \Diamond$$

Using this definition and rearranging the $z(\boldsymbol{x}-\xi_I)$ function in equation (2.4.25) results in the condition:

$$D(\xi_L, K) \le \left(\varepsilon \frac{(p+1)!}{M}\right)^{\frac{1}{p+1}} \tag{2.4.28}$$

which must hold if at least one $z(\boldsymbol{x}-\xi_I)$ function is to be negative at any $\boldsymbol{x}\in K$ for some chosen $\varepsilon > 0$.

We must now choose the functions $g(\boldsymbol{x}-\xi_I)$ such that one of the terms, ensured by the argument in the previous paragraph to be negative, is guaranteed to dominate the summation in equation (2.4.26) or (2.4.27). One necessary, but not sufficient, condition is that $z(\boldsymbol{x}-\xi_I)g(\boldsymbol{x}-\xi_I) \to 0$ as $\|\boldsymbol{x}\|_2 \to \infty$. Choosing $g(\boldsymbol{x}-\xi_I)$ to be Gaussian easily satisfies this condition but to ensure sufficiency the variances must be carefully selected. Consider the limit as the variance goes to zero. The interpolation functions of equation (2.4.22), will approach step functions where there exists a *J* such that:

$$\hat{g}(\boldsymbol{x}-\xi_I) = \begin{cases} 1 & \text{if } I = J \\ 0 & \text{if } I \neq J \end{cases}.$$

It was previously shown that choosing ξ_L such that (2.4.28) is satisfied will result in some $z(\boldsymbol{x}-\xi_J) < 0$, but since $\hat{g}(\boldsymbol{x}-\xi_I) = 0$ for $I \neq J$, we know that, in the limit, equation (2.4.27) will hold. However, since the number of points in ξ_L is finite we only need to be sufficiently close to the limit for this to be true.

The arguments presented above have been used by Johannsen and Foss (1993) to prove that the approximation accuracy defined by the infinity norm $\left\|f-\hat{f}\right\|_\infty = \sup_{\boldsymbol{x}\in K}\left\|f(\boldsymbol{x})-\hat{f}(\boldsymbol{x})\right\|_2$ may be described by the following theorem:

Theorem 5 *- Suppose we are given any integer $p \ge 0$. If K is bounded and f(.) has bounded $(p+1)$ th derivative, i.e. $\left\|\nabla^{p+1} f(\boldsymbol{x})\right\| < M, \forall \boldsymbol{x}\in K$, then for any $\hat{f} \in \left\{\hat{f} : K \to \Re^q \,\middle|\, \hat{f}(\boldsymbol{x}) = \sum_{\xi_I\in\xi_L} \hat{f}(\boldsymbol{x}-\xi_I)\hat{g}(\boldsymbol{x}-\xi_I)\right\}$ with finite countable ξ_L, and sufficiently narrow functions $\{g(\boldsymbol{x}-\xi_I) : K \to [0,1], \forall \xi_I \in \xi_L\}$, an upper bound on the approximation error is given by:*

$$\left\|f-\hat{f}\right\|_\infty \le \frac{M}{(p+1)!}\left(D(\xi_L, K)\right)^{p+1}. \tag{2.4.29}$$

◊ (See Johannsen and Foss (1993) for proof.)

By using the triangle inequality in conjunction with the Weierstrass approximation theorem, they also show, as a corollary to theorem 5, that the smoothness assumption on *f*(.) may be relaxed to assuming only continuity.

Notice that when order *p* of the local models are set to zero the resulting approximation is a normalised form of the approximation stated in equation (2.4.17) of the previous section. We can thus argue that although the effects of the curse of dimensionality may be reduced when using higher order local models, there has been no change to the structure that caused the problem. To do this it becomes necessary to define the interpolation functions on a space of smaller dimension than that of the original information space.

Let this smaller dimension space be represented by the bounded set Φ. Let any $\Phi_I \subset \Phi$ represent an operating regime within this space. Furthermore, let us assume that there exists a mapping $H : K \to \Phi$. If we choose a set of model validity functions $\{g(\phi - \eta_I) : \Phi \to [0,1], \forall \eta_I \in \eta_L, \eta_L = H(\xi_L)\}$ which are approximately equal to one for $\phi \in \Phi_I$ and tend toward zero outside Φ_I then, if not all $g(\phi - \eta_I)$ vanish for $\phi \in \Phi$, the interpolation functions may be defined as the normalisation of the model validity functions:

$$\hat{g}(\phi - \eta_I) = \frac{g(\phi - \eta_I)}{\sum_{\eta_J \in \eta_L} g(\phi - \eta_J)}.$$

This results in the following LMN approximation:

$$\hat{f}(\boldsymbol{x}) = \sum_{I \in I_0} \hat{f}(\boldsymbol{x} - \xi_I)\hat{g}(\phi - \eta_I) \quad I_0 = \{I \mid \eta_I = H(\xi_I), \xi_I \in \xi_L\}. \tag{2.4.30}$$

Furthermore let us assume that, for local model expansions of (polynomial) order *p*, we can split $\boldsymbol{x}$ into two parts $\boldsymbol{x}^T = [\boldsymbol{x}_L^T, \boldsymbol{x}_H^T]$, $\boldsymbol{x}_L \in K_L$, $\boldsymbol{x}_H \in K_H$, $K_L, K_H \subset K$ such that the approximated function may, as is frequently possible, be expressed as:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_L, \boldsymbol{x}_H) = f_{H_1}(\boldsymbol{x}_H) + f_{H_2}(\boldsymbol{x}_H) f_L(\boldsymbol{x}_L) \tag{2.4.31}$$

where $f_L : K_L \to \Re^l$ is a polynomial order less than or equal to *p*, $f_{H_1} : K_H \to \Re^m$ and $f_{H_2} : K_H \to \Re^{m \times l}$ may be of higher order. Given these conditions the following theorem (Johansen and Foss, 1993) may be stated:

Theorem 6 *- Suppose f(.) given in equation (2.4.31) is continuous and* Φ *is a bounded set. Then for any* $\varepsilon > 0$ *there is an* $\hat{f} \in \left\{ \hat{f} : K \to \Re^q \,\middle|\, \hat{f}(\boldsymbol{x}) = \sum_{I \in I_0} \hat{f}(\boldsymbol{x} - \xi_I)\hat{g}(\phi - \eta_I) \right\}$ *with* $\phi = \boldsymbol{x}_H$*, and countable and finite* I_0 *such that* $\left\| f - \hat{f} \right\|_\infty < \varepsilon$ ◊ (See Johannsen and Foss (1993) for proof.)

Additionally, for the special case were *p*=1 (i.e. local linear models), the approximation error is bounded by:

$$\left\| f - \hat{f} \right\|_\infty \le \frac{M}{2}\left(D(\eta_L, \Phi)\right)^2 + M\left\|K_L\right\|\left(D(\eta_L, \Phi)\right) = \varepsilon \tag{2.4.32}$$

where:

$$\left\|K_L\right\| = \sup_{\boldsymbol{x}_L \in K_L} \left\|\boldsymbol{x}_L\right\|_2.$$

On a more intuitive level we may state the presented results as follows: Components of the information space which contribute to approximation terms of higher order than the order of the local models used should be included in the reduced order space.

Thus the conditions and assumptions of the previous arguments have established a set of (non-unique) requirements, which allow us to construct a mapping $H : K \to \Phi$ that reduces the dimensions of the information space, but still permit the construction of an arbitrarily accurate approximation. For example, consider a discrete time system that is affine in its control inputs:

$$\begin{aligned} y[k] &= f(y[k-1], u[k-1]) \\ &= f_1(y[k-1]) + f_2(y[k-1])u[k-1] \\ &= f_{H_1}(\boldsymbol{x}_H) + f_{H_2}(\boldsymbol{x}_H) f_L(\boldsymbol{x}_L) \end{aligned}$$

then we may construct an approximation using interpolated first order models (p=1) as follows:

$$\hat{y}[k] = f(\boldsymbol{x}[k-1]) = \sum_{I \in I_0} \hat{f}(\boldsymbol{x}[k-1] - \xi_I)\hat{g}(\phi[k-1] - \eta_I) \quad I_0 = \{I \mid \eta_I = H(\xi_I), \xi_I \in \xi_L\}$$

where:

$$\begin{aligned} \boldsymbol{x}[k-1] &= [y[k-1], u[k-1]]^T \\ \phi[k-1] &= y[k-1] \end{aligned}$$

and the local models take the form:

$$\hat{f}(\boldsymbol{x}[k-1] - \xi_I) = (\boldsymbol{x}[k-1] - \xi_I)\hat{\Theta}_I \, .$$

Notice that the use of the reduced dimension space also simplifies the task of partitioning the set into various operating regimes.

As in section 2.4.4 we would like to devise a mechanism whereby a large domain may be covered by N local models but only $n<N$ models need be evaluated at once. Intuitively we recognize that we may discount local models whose validity functions are below a certain threshold, as they would not contribute significantly to the overall approximation. However, from a computational aspect, it is more efficient to not have to compute what would ultimately be unused model validity functions. Furthermore, it is difficult to determine what effects discounting individual local models have on the approximation error bound. The truncation radius mechanism presented earlier would appear to be a logical and viable approach to solving this problem while still permitting us to explicitly quantify the effects of discounting minimally contributing models on the overall approximation error bound.

Consider the following approximation construction:

$$\begin{aligned} \hat{f}(\boldsymbol{x}) &= \sum_{I \in I_0} \hat{f}(\boldsymbol{x} - \xi_I)\hat{g}(\phi - \eta_I) \\ I_0 &= \left\{I \mid \|\phi - \eta_I\|_2 \le \rho, \eta_I = H(\xi_I), \eta_I \in \eta_L, \xi_I \in \xi_L\right\} \end{aligned} \tag{2.4.33}$$

where the truncation radius ρ is the radius of an s-ball around ϕ, and $1 \le s \le n$. Furthermore we shall define the sets $\eta_{I_0} = \{\eta_I \mid I \in I_0\}$ and $\Phi_{I_0} = \left\{\phi \mid \|\phi - \eta_I\|_2 \le \rho, \eta_I \in \eta_{I_0}\right\}$ noting that I_0 and thus both η_{I_0} and Φ_{I_0} are functions of $\boldsymbol{x}$. If the truncation radius ρ is chosen sufficiently large such that the density of η_{I_0} in Φ_{I_0} is the

same as η_L in Φ then the approximation error bound provided by equation (2.4.32) still holds. As the truncation radius is reduced the density of η_{I_0} in Φ_{I_0} will diminish causing the distance $D(\eta_{L_0}, \Phi_{I_0})$ to increase. This results in $D(\eta_{L_0}, \Phi_{I_0}) > D(\eta_L, \Phi)$. Thus the approximation error bound, for the case of local linear models, at any point $\boldsymbol{x}$, may be obtained by substituting $D(\eta_{L_0}, \Phi_{I_0})$ into (2.4.32) resulting in:

$$\left\| f - \hat{f} \right\|_\infty \leq \frac{M}{2} \left(D(\eta_{L_0}, \Phi_{L_0}) \right)^2 + M \left\| K_L \right\| \left(D(\eta_{L_0}, \Phi_{L_0}) \right) \tag{2.4.34}$$

where:

$$\left\| K_L \right\| = \sup_{\boldsymbol{x}_L \in K_L} \left\| \boldsymbol{x}_L \right\|_2 .$$

Equation (2.4.34) is a function of $\boldsymbol{x}$ because both η_{I_0} and Φ_{I_0} are functions of $\boldsymbol{x}$. However, if the lattice ξ_L is evenly spaced over the entire domain *K*, and the mapping $H : K \to \Phi$ is constructed by using the projection $\phi = \boldsymbol{x}_H$, then the distance $D(\eta_{L_0}, \Phi_{I_0})$ will remain constant for all $\boldsymbol{x} \in K$ [7]. The approximation error bound would thus also remain constant and could be calculated at any point $\boldsymbol{x} \in K$.

In conclusion then, we have seen that the LMN approach, from an approximation point of view, thus incorporates many of the capabilities of the MLP and RBF approaches but gives us two additional advantages. Firstly, in contrast to Gabor and Wavelet approaches, it provides us with the ability to easily incorporate *a-priori* or extract *a-posteriori* knowledge about a system. Secondly it accommodates a mechanism whereby the effects of the curse of dimensionality may be potentially reduced by trading off the local model complexity against the number of dimensions used in constructing the approximation sampling lattice.

2.5 DISCUSSION AND CONCLUSIONS

In this chapter we began by considering neural networks as information processing systems and breaking their analysis into computational, algorithmic and implementation levels. The basic components of an ANN system were described and, at a superficial level, the relationship between them and their biological counterparts were discussed. Although it is not the intent of this work to model biological neural networks one should not dismiss the biological connection (Arbib, 1995). The study of neurobiology can be fertile ground for the development of ideas and concepts that are then transplanted into an engineering framework. This approach may become ever more important as the emphasis in ANN research shifts from network architecture definition to dynamic construction, optimisation and modification of such architecture (Quinlan, 1998.)

Next, emphasis is turned to the feed forward neural network problem, and how such feed forward networks are used to describe non-linear systems in the context of this work. It is important to note that the method described is but one approach to the problem at hand and that many other architectures are used to model non-linear dynamic systems. Typically neural network approaches to control can be divided into two main categories. The first, as used in this work, is the use of a feed forward network coupled to some time history model of the

[7] This assumes that there are no ill effects on the distance metric as one approaches the edges of the domain *K*.

system. Here an ARX model is the method used to capture the system dynamics. Another method, which is structurally similar but uses multiple local Laguerre models instead, is described in Sbarbaro and Johansen (1997). The second major category uses recurrent neural networks, which build on feed forward network theory by attempting to model system dynamics by feeding back network states within or around a feed forward network. Although it can be proven that such network structures can solve the differential approximation problem (Zibikowski, 1994) such results are purely existence results and are fraught with difficulties when attempting practical implementations. However, a number of researchers (Pearlmutter, 1989, Pineda, 1987, Williams and Zisper, 1988, Bailer-Jones *et al*, 1998) have successfully developed recurrent architectures for modelling of dynamic systems.

The remainder of the chapter presented and discussed various approximation theory results and how they relate to different kinds of network architectures. First the Stone - Weierstrass theorem provided us with a set of criterion that given functions must satisfy in order to guarantee the existence of a uniform approximation constructed from these given functions. Kolomogorov's representational theorem was used as the basis for an argument permitting approximation construction using MLP networks. Unfortunately these results provide little information as to precisely how the MLP network in question should be constructed or even how any particular construction will behave with regards to issues such as approximation interpolation (generalisation) and overall approximation error. Although significant advances have been made in this area (Barron 1994, Suzuki, 1998) it is still unclear exactly how best to construct a general MLP network.

These shortcomings are addressed by the multi-dimensional sampling approach used in RBF networks. It is shown that this approach will provide a network that will interpolate an approximation constructed from a finite number of samples and, provided certain assumptions and conditions are met, guarantee that the approximation error remains within certain calculable bounds. The analysis provides us with sufficient insight into the approximation mechanism that a limited but useful design procedure can be stated. This procedure requires only limited smoothness information about the function to be approximated. Clearly this approach has a number of advantages, but it suffers from two main disadvantages; the difficulty associated with including *a-priori* information and its susceptibility to the curse of dimensionality.

LMN networks address both these problems and the final section of the chapter thus concentrated on the approximation capabilities of these networks. Interestingly, an RBF network, with normalisation, may be viewed as a LMN where the local models are constants. The approach therefore retains many of the advantages of RBF networks. However, local models may be of any order or construction.

The arguments presented have centred on the use of Gaussian functions for both RBF and LMN network construction. Although these functions have a number of appealing properties other functions, such as inverse polynomials or splines may be used instead, however, this would require the revaluation of some of the results presented thus far. The primary motivation behind doing this would be the reduction of computational overhead or to minimise the probability of reactivation - a side effect introduced by the normalisation of the basis functions (Murray-Smith and Johansen, 1997.) The remainder of this work will assume that Gaussian basis functions are used in RBF and LMN network construction.

Another potential advantage of the LMN approach is its behaviour at the edges of the approximation domain *K*. An LMN approximation will extrapolate outside this region using the closest local model to the current point in the information space. This extrapolation is of the same order as the closest local model. When used as plant models in control systems such behaviour may be more favourable when compared to RBF networks whose output will tend to zero as the information vector moves outside the domain *K*. Normalised RBF networks will, under such conditions, tend to a constant value. The implicit assumption behind this observation is that the extrapolated model's sensitivity maintains some degree of accuracy whereas an RBF based model's sensitivity would be driven to zero. This characteristic becomes important when considering the stability of a closed loop control system incorporating such networks to estimate plant sensitivity.

Although the LMN approach provides a number of advantages over the MLP and RBF methods there are also disadvantages. To construct an RBF network using the design procedure in section 2.4.4, the designer need only know the extent of the domain *K,* the value of F_{max} and the required spectral support β. The latter two values provide sufficient information about the smoothness of the approximated function that the approximation error on the domain *K* may be determined from equation (2.4.20). This smoothness information may be determined relatively easily from system data. Conversely, the equivalent smoothness information used in calculating the LMN approximation error (equation (2.4.32)), is contained in the maximum induced operator norm *M*. Unfortunately this value can not be directly determined without significant knowledge of the function being approximated. Therefore, the required density of the set η_L, and thus the number of models cannot be directly determined from the approximation error equation.

It is interesting to note that there is a strong relationship, even equivalency under certain conditions, between fuzzy systems and the RBF and LMN architectures. This has practical relevance in that the techniques developed in one paradigm may be used to manipulate models developed in a different paradigm. Furthermore a particular paradigm may be more adept at dealing with one particular aspect of the modelling problem, for example the use of fuzzy a-priori knowledge may be beneficial in pre-structuring a network. The reader is referred to Hunt *et al* (1996) and (Foss and Johansen, 1993) for precise details.

This chapter has presented numerous mathematical results and facts about various approximation techniques, all of which fall under the broad subject of artificial neural networks. This semi- rigorous mathematical presentation has been provided to help gain insight and understanding into the mechanisms underlying the methods used in the remainder of this work, and to support the conclusion that ANN provide a viable and practical solution to the non-linear function approximation problem. Thus continued effort into the application of these techniques to non-linear dynamic systems modelling is warranted.

CHAPTER 3

NEURAL NETWORKS - STRUCTURE AND IMPLEMENTATION

3.1 INTRODUCTION

In the previous chapter, three basic mathematical formulations for ANN structures were shown to satisfy the FNN approximation problem. We concluded the chapter by recognising that the computation ability of these approaches provided a satisfactory solution to the problem at hand. In this chapter, we will concentrate on the more pragmatic aspects of general information processing systems; the algorithmic formulation and implementation.

We have chosen to implement the presented algorithms as a set of methods contained in a simple neural network class library coded in the Matlab language. Network objects created using this class library can be manipulated from the Matlab command line. To facilitate the simulation of dynamic systems a Simulink block set was also created which uses a number of m-file S-functions to create and process network objects by calling the aforementioned class library methods. All the implementation material presented has thus been optimised for this environment, specifically, because of Matlab's emphasis on using vectorized code, the algorithms have been vectorized whenever possible.

We shall begin by looking at, for each network type, the third element of an ANN system (section 2.2.), namely, the structure or activation rule used to implement the approximation. Next, the fourth and fifth elements are covered by looking at how the network parameters are adapted using various learning rules. Included in this section is a discussion about experimental conditions and structure optimisation. Finally, the process of extracting the network Jacobian information will be presented followed by conclusions.

3.2 NETWORK STRUCTURE - THE ACTIVATION RULE

3.2.1 MULTI-LAYER PERCEPTRON NETWORK STRUCTURE

The MLP network is based on the approximation construction discussed in section 2.4.3. It has a multi-layer structure (Figure 3.2.1) consisting of one input layer, any number of hidden layers and one output layer. The input layer or first layer would contain as many units as there are inputs, while the output layer sizes according to the number of network outputs. The hidden layers may contain any number of units. Generally the number of

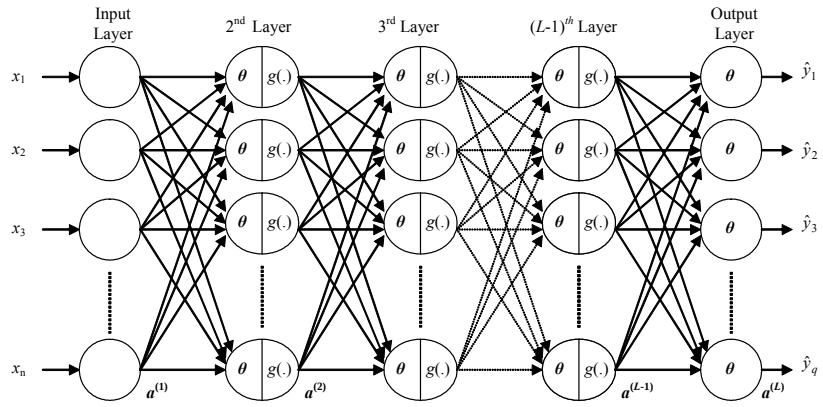


Figure 3.2.1. Multi-Layer Perceptron Network Structure.

```
10  net.activation{1}=[x;1];                          % 1st layer is input
20  for i=2:net.numlayers;                            % i+1 layer
30      net.activation{i}=[activation_function(net.weights{i}*net.activation{i-1});1];
40  end
50  yhat=net.activation{net.numlayers}(1:end-1);      % Remove bias term in Output
```

Listing 3.2.1 MLP Activation Rule Implementation.

hidden layers is limited to one or two as this number is sufficient to construct the function approximation. A layer may receive excitation from any preceding layer but, as in this work, each layer is typically excited only by the layer directly preceding it. The sole purpose of the input layer is to distribute the input signals to the first hidden layer and performs no mathematical manipulations on the incoming signals. Hidden layers use a linear integration function coupled with a sigmoid or sigmoid like function. The output layer may frequently use linear integration and activation functions, resulting in an output that is a weighted sum of the last hidden layer unit activations. Combining all these elements allows us to construct an expression for the activation rule. Let the column vectors $\boldsymbol{x}$, $\hat{\boldsymbol{y}}$ and $\boldsymbol{a}^{(l)}$ represent the input, output and activation state of the l^{th} layer of the network respectively. Note that $\boldsymbol{x} = \boldsymbol{a}^{(1)}$ and $\hat{\boldsymbol{y}} = \boldsymbol{a}^{(L)}$ for a network containing L+1 layers. Further let $\boldsymbol{\theta}_j^{(l)}$ denote the weight vector of the j^{th} unit in the l^{th} layer where:

$$\boldsymbol{\theta}_j^{(l)} = \left[\theta_{j,1}^{(l)}, \theta_{j,2}^{(l)}, \cdots, \theta_{j,m_{l-1}}^{(l)}, \theta_j^{(l)}\right]^T \quad 1 < l \le L \tag{3.2.1}$$

and $\theta_{j,i}^{(l)}$ is the scalar weight connecting the j^{th} unit of the l^{th} layer to the i^{th} unit of the (l-1)th layer which contains m units, and $\theta_j^{(l)}$ is the bias associated with the j^{th} unit. Using equation (3.2.1) we may define a weight matrix for each layer:

$$\Theta^{(l)} = \left[\boldsymbol{\theta}_1^{(l)}, \boldsymbol{\theta}_2^{(l)}, \boldsymbol{\theta}_3^{(l)}, \cdots \boldsymbol{\theta}_{m_l}^{(l)}\right]^T \quad 1 < l \le L\,. \tag{3.2.2}$$

Lastly, we create a vector of activation functions as follows:

$$G^{(l)}(\cdot) \equiv \begin{bmatrix} g(\boldsymbol{\theta}_1^{(l)T}[\boldsymbol{a}^{(l-1)^T},1]^T) \\ g(\boldsymbol{\theta}_2^{(l)T}[\boldsymbol{a}^{(l-1)^T},1]^T) \\ \vdots \\ g(\boldsymbol{\theta}_{m_l}^{(l)T}[\boldsymbol{a}^{(l-1)^T},1]^T) \\ 1 \end{bmatrix} = \begin{bmatrix} g\left(\Theta^{(l)}[\boldsymbol{a}^{(l-1)^T},1]^T\right) \\ 1 \end{bmatrix} \quad 1 < l \le L . \tag{3.2.3}$$

Note that at any point in time $G^{(l)}(\cdot)$ will take on the value $[\boldsymbol{a}^{(l)^T},1]^T$ and that the total number of units evaluated during each forward pass may be expressed as:

$$N = \sum_{l=2}^{L} m_l . \tag{3.2.4}$$

Combining equations (3.2.2) and (3.2.3) permits us to write a single expression for the MLP activation rule:

$$\begin{gathered} \hat{\boldsymbol{y}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) = \Theta^{(L)} G^{(L-1)}\left(\Theta^{(L-1)} G^{(L-2)}\left(\Theta^{(L-2)} \cdots G^{(2)}\left(\Theta^{(2)}[\boldsymbol{x}^T,1]^T\right)\cdots\right)\right) \\ \Theta^{(l)} \in \Re^{m_l \times (m_{l-1}+1)}, G^{(l)} \in \Re^{(m_l+1)\times 1}. \end{gathered} \tag{3.2.5}$$

Implementation of equation (3.2.5) is straight forward and is shown in Listing 3.2.1

To relate the theoretical foundations presented in section 2.4.3 to the MLP structure presented above, consider a network containing an input layer with *n* inputs, 2 hidden layers with m_2 and m_3 units respectively and an output layer with *q* outputs. Equation (3.2.5) thus becomes:

$$\begin{gathered} \hat{\boldsymbol{y}} = \Theta^{(4)} G^{(3)}\left(\Theta^{(3)} G^{(2)}\left(\Theta^{(2)}[\boldsymbol{x}^T,1]^T\right)\right) \\ \boldsymbol{x} \in \Re^{n\times 1}, \Theta^{(4)} \in \Re^{q\times(m_3+1)}, \Theta^{(3)} \in \Re^{m_3\times(m_2+1)}, \Theta^{(2)} \in \Re^{m_2\times(n+1)}. \end{gathered}$$

Expanding the matrix operations as follows:

$$\hat{\boldsymbol{y}} = \sum_{i=1}^{m_3} \Theta^{(4)}_{[1\cdots q],i} g\left(\sum_{j=1}^{m_2} \Theta^{(3)}_{i,j} g\left(\Theta^{(2)}_{j,[1\cdots n]}\boldsymbol{x} + g^{-1}\left(\Theta^{(3)}_{j,m_2+1}\right)\right) + g^{-1}\left(\Theta^{(4)}_{[1\cdots q],m_3+1}\right)\right),$$

and simplifying the notation by making the following substitutions:

$$\begin{aligned} a_i &= \Theta^{(4)}_{[1\cdots q],i}, & a_j &= \Theta^{(3)}_{i,j} \\ b_i &= 1, & b_j &= \Theta^{(2)}_{j,[1,\cdots n]} \\ c_i &= g^{-1}\left(\Theta^{(4)}_{[1\cdots q],m_3+1}\right), & c_j &= g^{-1}\left(\Theta^{(3)}_{j,m_2+1}\right) \end{aligned}$$

results in:

$$\begin{aligned} \hat{\boldsymbol{y}} &= \sum_{i=1}^{m_3} a_i g\left(b_i\left(\sum_{j=1}^{m_2} a_j g\left(b_j\boldsymbol{x} + c_j\right)\right) + c_i\right) \\ &= \sum_{i=1}^{m_3} \chi_i\left(\sum_{j=1}^{m_2} \phi_{ij}\left(x_j\right)\right). \end{aligned} \tag{3.2.6}$$

Clearly (3.2.6) matches the functional form required in equation (2.4.6) of Theorem 3 in section 2.4.3, implying that this four-layered MLP network may approximate, with arbitrary error, any continuous function. It is worth while noting however, that there is no indication of how to choose values for m_2 and m_3 in order to remain within some arbitrary error bound.

3.2.2 RADIAL BASIS FUNCTION NETWORK STRUCTURE

The Gaussian Radial Basis Function Network implements the approximation described by equation (2.4.17) and (2.4.18), restated here in a multi output form as:

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{I \in I_o} \boldsymbol{\theta}_I g(\boldsymbol{x} - \xi_I) \quad I_o = \left\{ I \mid \left\| \boldsymbol{x} - \xi_I \right\|_2 \le \rho, \boldsymbol{x} \in K \subset \Re^n, \xi_I \in \xi_L \right\}$$
$$\boldsymbol{\theta}_I = \boldsymbol{c}(\xi_I)\ ,\ g(\boldsymbol{x} - \xi_I) = e^{-\left((\boldsymbol{x}-\xi_I)^T \Sigma (\boldsymbol{x}-\xi_I)\right)} \tag{3.2.7}$$

where ξ_I is a vertex in the lattice $\xi_L = \{\Delta_1 i_1 \boldsymbol{e}_1 + \Delta_2 i_2 \boldsymbol{e}_2 + \cdots + \Delta_n i_n \boldsymbol{e}_n)\}$ which has sample spacing Δ_j along each of the basis vectors $\boldsymbol{e}_j$ for $\Re^n$, $L = \{i_1, \ldots, i_n\}$ is an *n*-tuple of integers, and ρ is a *n*-ball around $\boldsymbol{x}$. The approximation properties and design procedure are described in section 2.4.4. The general network construction, shown in Figure 3.2.2, consists of three layers. As in the MLP the input and output layer size according to the number of inputs and outputs respectively. The number of hidden units corresponds to the number of elements in the set *L* and may be expressed as:

$$N = \prod_{j=1}^{n} \max i_j \quad \forall i_j \in L\ . \tag{3.2.8}$$

Clearly *N* will depend on the selected lattice spacing or sampling mesh size and the dimensions of the set *K*. In this configuration the vertices of the sampling mesh and the variances may be interpreted as the weight matrices associated with an elliptical integration function. These weights define the input to hidden layer connectivity, while the j^{th} row of the column vector $\boldsymbol{\theta}_I$ is the weight associated with the connection between the hidden unit centred at ξ_I and the j^{th} output. The activation rule is thus a direct implementation of equation (3.2.7). Note however, only the units that are influenced by the elements of the set I_0 need actually be computed at any point in time.

In the special case where the basis vectors $\boldsymbol{e}_j$ are orthogonal, we refer to the network as an axis orthogonal network. Although this type of network may not always provide the most efficient structure, in terms of the

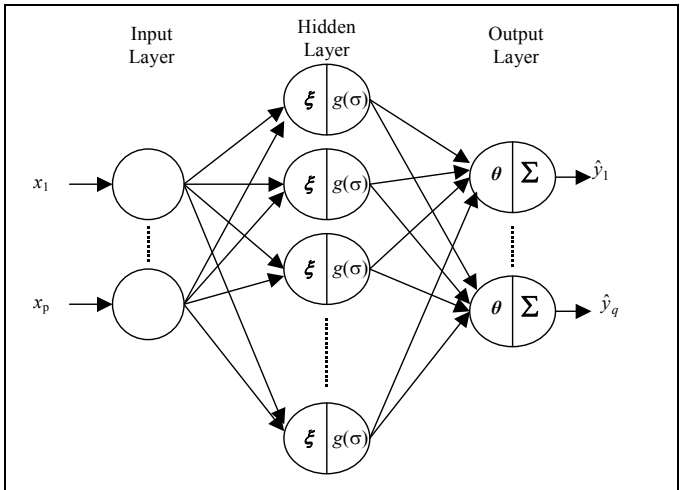


Figure 3.2.2. General RBF Network Structure.

number of units required to achieve a particular approximation, it allows us to exploit the linearly separable nature of the Gaussian basis functions. This has the potential to greatly reduce the number of calculations that must be performed when implementing the network in a serial processing environment. When each dimension of the input space is orthogonal, the matrix $\boldsymbol{\Sigma}$ in equation (3.2.7) is diagonal, permitting each of the N hidden unit activations to be expressed as a tensor product:

$$\begin{aligned} g(\boldsymbol{x}-\xi_I) &= \prod_{j=1}^{n} e^{-\frac{({}_jx-{}_j\xi_I)^2}{\sigma_j^2}} \qquad \forall \xi_I \in \xi_L \\ &= \prod_{j=1}^{n} a({}_jx, {}_j\xi_I, \sigma_j) \end{aligned} \tag{3.2.9}$$

where ${}_j\xi_I$ is the j^{th} element of the vector ξ_I. Careful consideration of equation (3.2.9) reveals that when calculating all N activations, the term $a({}_jx, {}_j\xi_I, \sigma_j)$ will be equal $N/\max i_j$ times for each I and j combination. By collecting these common terms and performing a single exponential evaluation, we may reduce the number of exponentials calculated to:

$$N_o = \sum_{j=1}^{n} \max i_j \quad \forall i_j \in L. \tag{3.2.10}$$

This may be easily visualised by studying a simple example. Consider a network with two inputs and a single output. Let the sampling lattice be divided into three regions along input one and two regions along input two. This lattice representation is pictured in Figure 3.2.3(a) with its equivalent layered representation in Figure 3.2.3(b). It is clear that, for this example, $a(x_1, {}_1\xi_{1,1}, \sigma_1) = a(x_1, {}_1\xi_{1,2}, \sigma_1)$ is common to both the calculations of $g(\boldsymbol{x}-\xi_{1,1})$ and $g(\boldsymbol{x}-\xi_{1,2})$, similarly $a(x_2, {}_2\xi_{1,2}, \sigma_2) = a(x_2, {}_2\xi_{2,2}, \sigma_2) = a(x_2, {}_2\xi_{3,2}, \sigma_2)$ is common to $g(\boldsymbol{x}-\xi_{1,2})$, $g(\boldsymbol{x}-\xi_{2,2})$ and $g(\boldsymbol{x}-\xi_{3,2})$. The common $a({}_jx, {}_j\xi_I, \sigma_j)$ terms are equal because all elements, except the j^{th}, of ξ_I are ignored when the calculation is performed. Therefore, by replacing ${}_j\xi_I$ with the j^{th} element of I multiplied by Δ_j, where Δ_j is the spacing of the lattice ξ_L along the dimension j, we may evaluate a single exponential which will equal all the common terms generated by appropriate combinations of I and j. Thus, in the presented example:

$$\begin{aligned} a(x_1, 1\Delta_1, \sigma_1) &= a(x_1, {}_1\xi_{1,1}, \sigma_1) = a(x_1, {}_1\xi_{1,2}, \sigma_1) \\ a(x_1, 2\Delta_1, \sigma_1) &= a(x_1, {}_1\xi_{2,1}, \sigma_1) = a(x_1, {}_1\xi_{2,2}, \sigma_1) \\ a(x_1, 3\Delta_1, \sigma_1) &= a(x_1, {}_1\xi_{3,1}, \sigma_1) = a(x_1, {}_1\xi_{3,2}, \sigma_1) \\ a(x_2, 1\Delta_2, \sigma_2) &= a(x_2, {}_2\xi_{1,1}, \sigma_2) = a(x_2, {}_2\xi_{2,1}, \sigma_2) = a(x_2, {}_2\xi_{3,1}, \sigma_2) \\ a(x_2, 2\Delta_2, \sigma_2) &= a(x_2, {}_2\xi_{1,2}, \sigma_2) = a(x_2, {}_2\xi_{2,2}, \sigma_2) = a(x_2, {}_2\xi_{3,2}, \sigma_2). \end{aligned}$$

Also, notice that, as predicted by equation (3.2.10), there need be only five exponentials evaluated, but consistent with equation (3.2.8), there are six activation values, each of which is calculated by the product described in equation (3.2.9). Although in this example the number of exponentials is only reduced by one, the reduction becomes far more dramatic as the number of regions in the lattice is increased. For example, a two dimensional system with twenty regions associated with each input would require forty exponentials but would contain four hundred activations - an order of magnitude reduction.

Evaluating only those activations influenced by the set I_0 can still further reduce the number of calculations. In Figure 3.2.3(a) the grey region[8] represents the *p*-ball (a circle in the 2 dimensional case) of radius ρ surrounding an input vector ending at the point $\{x_1, x_2\}$. By projecting this region back onto the axes it is clear that only the functions $a(x_2, 2\Delta_2, \sigma_2)$ and $a(x_1, 2\Delta_1, \sigma_1)$ provide any significant contribution to the output. The total number of calculations required to generate the activations contributing to the summation in equation

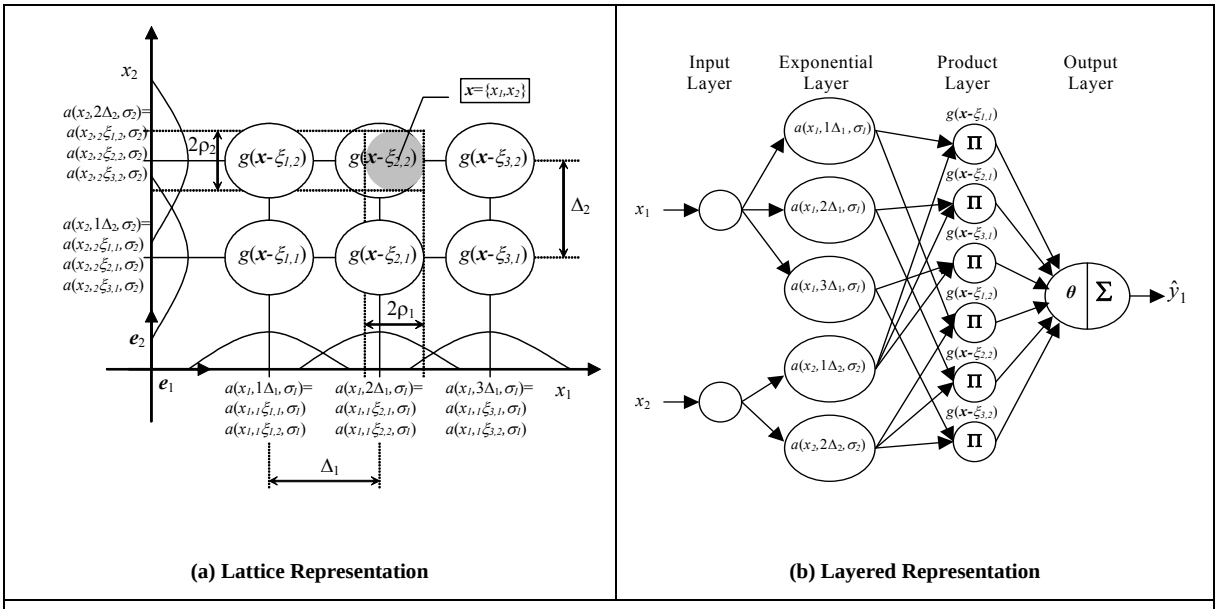


Figure 3.2.3. Axis Orthogonal RBF Network Example.

(3.2.7) is thus two exponentials and one product.

From the above discussion, we may easily develop an algorithm to efficiently calculate all unit activations that may significantly contribute to the output given a particular input. However, in order to calculate the network output, we must also perform the summation shown in equation (3.2.7). This equation may be similarly expressed using matrix operations. Let us assume that for some particular input the set I_0 contains *l* vector elements. We may then construct Θ_{I_0} and $G(\boldsymbol{x}, \xi_{I_0})$ by joining the *l* output weight vectors $\boldsymbol{\theta}_I$, and the *l* calculated activations as follows:

[8] In practice a significantly larger region would be used so that more than one vertex is encompassed. However, for the purposes of illustrating the concept, while maintaining a simple diagram, the region has been made fictitiously small.

$$\Theta_{I_0} = \left[\boldsymbol{\theta}_{I_1} \mid \boldsymbol{\theta}_{I_2} \mid \cdots \mid \boldsymbol{\theta}_{I_l}\right]^T, \quad \forall I_{[1\cdots l]} \in I_0, \Theta_{I_0} \in \Re^{l \times q}$$
$$G(\boldsymbol{x}, \xi_{I_0}) = \begin{bmatrix} g(\boldsymbol{x} - \xi_{I_1}) \\ g(\boldsymbol{x} - \xi_{I_2}) \\ \vdots \\ g(\boldsymbol{x} - \xi_{I_l}) \end{bmatrix} \tag{3.2.11}$$

resulting in:

$$\hat{\boldsymbol{y}} = \Theta_{I_0}^T G(\boldsymbol{x}, \xi_{I_0}) . \tag{3.2.12}$$

To obtain the construction in (3.2.11) an efficient mechanism for simultaneously calculating a weight index number for each corresponding unit activation must be devised.

An implementation that performs the calculations above while simultaneously attempting to minimise memory and computational resources, is shown in Listing 3.2.2. The code begins by initialising two variables in line[9] ten. The first variable is a vector that will combine the exponential function results, eventually becoming the final activation vector `actv.` The second, `idxw,` will become a vector containing the weight indexes that correspond to the activations contained in `actv.` The code then loops (line 20 through 180) through each input to the network.

```
10  actv=1; idxw=0;                                % Initialize variables
20  for j=1:net.ninp                               % Loop through each network input
30      expn=(u(j)-net.center{j}).*net.invsigma{j}; % Find exponents for jth input
40      idxe=find(abs(expn)<net.nthld1);           % Determine exponentials activated
50      expn=expn(idxe); actv=actv(:)*exp(-expn.^2); % Multiply with active units
60      vct1=((idxe-(j>1)).*net.vcprd(j));         % Calculate new indexes
70      vct1=vct1(ones(size(idxw,1),1),:);         % Copy vct1 to rows(idxw) rows
80      matx=idxw(:,ones(1,size(idxe,2)));         % Copy idxw to cols(idxe) columns
130     idxw=vct1(:)+matx(:);                      % Update weight index vector
140     idxa=find(actv>net.nthld2);                % Make truncation region a ball
150     actv=actv(idxa);                           % Remove inactivated units
160     idxw=idxw(idxa);                           % Remove inactivated weight indexes
180 end                                            % End for loop
200 yhat=net.weights(:,idxw)*actv(:);              % Output result
```

Listing 3.2.2. Axis Orthogonal RBF Activation Rule Implementation.

The loop begins in line 30 by calculating a vector that contains the square root of the negative exponent for each exponential function of the j^{th} input. That is:

$$\texttt{expn=(u(j)-net.center\{j\}).*net.invsigma\{j\}} = (x_j - \Delta_j[1, \cdots, \max i_j])(1/\sigma_j) .$$

The absolute of the j^{th} value of `expn` will be less than a certain threshold if the j^{th} component of the input vector is close enough to the centre of the j^{th} exponential function. It is easy to show that this threshold `(net.thld1)` is related to the truncation radius ρ_j, and the spatial variance σ_j, by $\texttt{net.thld1} = (\rho_j / \sigma_j)$. Thus, if the components of `expn` are less than this threshold, then the corresponding exponential functions fall inside the

[9] Line numbers increment in steps of ten except where code still to be presented is to be inserted.

truncation radius and must be evaluated. Line 40 extracts the indexes of the values in `expn` that meet this condition and stores the results back in `expn`.

Line 50 calculates the exponentials for the `expn` values and performs an outer vector product with the current vector of activations. On the first iteration the product is simply the exponential results times unity. If the system has a single input (i.e. one-dimensional) then the result is a vector of the desired activations. If the system has two inputs then the result, after the second iteration, is a matrix where each entry represents an activation value at the lattice vertices. Note that the region of excited units would be square. For systems with three or more inputs, the matrix resulting from second and/or subsequent iteration is first converted to a vector by stacking the matrix columns on top of one another. This is has the effect of converting the first two or more dimensions of information to an equivalent single dimension system. The outer product then augments the system with the next dimension's contribution to the final activation. Lines 150 and 160 are used to ensure that, as each dimension is processed, only activated units inside a *p*-ball (ellipsoid) are processed on the next iteration. The mechanism is easily seen in the two-dimensional case where, as noted previously, the region of excited units represented by the activation matrix is square (rectangular). The smallest values of activation will be found toward the corners of this region, where the result is the product of two small numbers. By defining another post product threshold below which the activation is ignored, causes the corners of the region to be ignored. The resulting region is thus circular (elliptical). The threshold value, `net.thld2,` is related to the truncation radius and variance by:

$$\texttt{net.thld2} = e^{-\sum_j \left(\rho_j / \sigma_j\right)^2} .$$

Lines 60 to 80, together with lines 130 and 160, are used to construct a vector of indexes that identify all the activated units. Once this index vector is constructed the weights corresponding to all active units may be extracted in a single statement. The final network output can then calculated, as in line 200, by performing the matrix operations of equation (3.2.12).

Simply put, the index vector code calculates a scalar index into an *n*-dimensional array. The index represents the ordinal number for any particular element of an *n*-dimensional array if the array were stored in linear memory by sequentially concatenating each dimension. For example, appending the columns for the first layer in an end-to-end fashion, and then appending the second layer columns in the same manner, would store a three-dimensional array. The index value for any entry may be calculated directly from the element's subscript values. Note that the activation vector created in line 50 is a linear representation of selected entries of the *n*-dimensional array of activation values.

The code works by creating a new column vector whose first element is a one, followed by the cumulative product of the vector defining the number of regions in the first *n*-1 dimensions. Next, the element's subscripts are placed in a row vector and each element, except the first, is decremented by one. The index value is the inner product of the two vectors. For example, using the three dimensional array (system) previously introduced, let us suppose that each of the dimensions (inputs) has been divided into seven regions, then the entry in location [2,4,6] would have an index value of $[2,(4-1),(6-1)][1,(1x7),(1x7x7)]^{T} = 268$.

Continuing this example, let us assume an input vector such that the 1st, 2nd, and 3rd exponentials are excited by input component one, the 3rd and 4th exponentials are excited along dimension two, and the 6th and 7th exponentials are excited by component three. From the information given, and assuming a square truncation region, we know that the indexes may range from 1 to 343 and that for the particular input vector under consideration there will be 3x2x2=12 units activated. Table 3.2.1 shows how the code goes about generating the index vector for this example. The initialised variables are shown in the first row. The second row shows the variables on the first iteration of the loop. We see that `idxe` is assigned the vector of subscripts for the excited exponentials. In line 60 each element is multiplied (because this is the first iteration they are not decremented) by the first element of `net.vcprod` and the result is stored in `vct1`. Line 70 then copies this row vector as many times as there are elements in `idxw` and stores the resulting matrix back in `vct1`. This prepares `vct1` to be summed with the previous terms of the inner product created for lower ordinal subscripts. Naturally during the first iteration there are no previous terms and the vector is simply copied back onto itself. Next, `matx` is prepared by line 80, which copies the current `idxw` column vector as many times as there are elements in `idxe`. This prepares the earlier terms of the inner product to be added with `vct1`. Clearly, during the first iteration, zeroing each element will compensate for the lack of any previous terms. This is achieved by simply initialising `idxw` to the scalar value of zero. Note that `vct1` and `matx` will always have the same dimensions. In line 130 the final step converts both matrices to column vectors by concatenating columns, and then performs the sum of the inner product. The result is stored in `idxw` for use during the next iteration. The second iteration values, in row three of the table, show how `vct1` and `matx` grow as higher dimensions are processed. Note also that `idxe` is decremented on second and subsequent iterations before being multiplied with `net.vcprd`. The final weight index vector is shown at the bottom right corner of the table.

j	idxe	idxe-(j>1)	vcprd(j)	vct1	matx	idxw
[.]	[.]	[.]	[1 7 49]	[.]	[.]	[0]
1	[1 2 3]	[1 2 3]	[1]	[1 2 3]	[0 0 0]	[1; 2; 3]
2	[3 4]	[2 3]	[7]	[14 21; 14 21; 14 21]	[1 1; 2 2; 3 3]	[15; 16; 17; 22; 23; 24]
3	[6 7]	[5 6]	[49]	[245 294; 245 294; 245 294; 245 294; 245 294; 245 294]	[15 15; 16 16; 17 17; 22 22; 23 23; 24 24]	[260; 261; 262; 267; 268; 269; 309; 310; 311; 316; 317; 318]

Table 3.2.1. Example of Variable Evolution During Weight Index Calculation.

There are a number of approaches that may be utilised in implementing the axis orthogonal RBF activation rule. However, the presented method of calculating unit activation and indexes permits rapid construction of the activation and index vector, uses minimal memory, makes efficient use of computational resources, and is scalable to any number of dimensions. Furthermore, as we shall see later, the technique may be directly carried over to LMN networks and also facilitates the rapid computation of the network's Jacobian matrix.

3.2.3 LOCAL MODEL NETWORK STRUCTURE

Section 2.4.5 discusses the approximation ability of the Local Model Network where the approximation construction is described by equation (2.4.30). In this work we will confine our investigation to networks that use linear local models and Gaussian interpolation functions. Under these constraints we may restate the approximation construction as:

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{I \in I_0} \hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_I)\hat{g}(\phi - \eta_I) \tag{3.2.13}$$

where:

$$\hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_I) \in \left\{ \hat{\boldsymbol{f}} : K \to \Re^q \middle| \hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_I) = \hat{\Theta}_I^T(\boldsymbol{x} - \xi_I), \boldsymbol{x} \in K \subset \Re^n, \hat{\Theta}_I \in \Re^{n \times q}, I \in I_0 \right\} \tag{3.2.14}$$

$$I_0 = \left\{ I \mid \left\| \phi - \eta_I \right\|_2 \le \rho, \eta_I = H(\xi_I), \eta_I \in \eta_L, \xi_I \in \xi_L \right\} \tag{3.2.15}$$

together with the model validity functions:

$$\hat{g}(\phi - \eta_I) = \frac{g(\phi - \eta_I)}{\sum_{J \in I_0} g(\phi - \eta_J)} \tag{3.2.16}$$

where the interpolation function is expressed as:

$$g(\phi - \eta_I) = e^{-\left((\phi - \eta_I)^T \Sigma (\phi - \eta_I)\right)} \tag{3.2.17}$$

and $\xi_L = \left\{ \Delta_1 i_1 \boldsymbol{e}_1 + \Delta_2 i_2 \boldsymbol{e}_2 + \cdots + \Delta_n i_n \boldsymbol{e}_n) \right\}$ has spacing Δ_j along each of the basis vectors $\boldsymbol{e}_j$ for $\Re^n$, and $L = \{ i_1, \ldots, i_n \}$ is an n-tuple of integers. Furthermore it is assumed that the function being approximated may be expressed as:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_L, \boldsymbol{x}_H) = f_{H_1}(\boldsymbol{x}_H) + f_{H_2}(\boldsymbol{x}_H) f_L(\boldsymbol{x}_L) \tag{3.2.18}$$

where $\boldsymbol{x}^T = [\boldsymbol{x}_L^T, \boldsymbol{x}_H^T]$, $\boldsymbol{x}_L \in K_L \subset \Re^{n-s}$, $\boldsymbol{x}_H \in K_H \subset \Re^s$, $K_L, K_H \subset K \subset \Re^n$, and the mapping $H : K \to \Phi$ is constructed as the projection $\phi = \boldsymbol{x}_H$. The lattice η_L is constructed according to $\eta_L = H(\xi_L)$ implying that $\eta_L = \left\{ \Delta_{n-s+1} i_{n-s+1} \boldsymbol{e}_{n-s+1} + \Delta_{n-s+2} i_{n-s+2} \boldsymbol{e}_{n-s+2} + \cdots + \Delta_n i_n \boldsymbol{e}_n \right\}$ where $1 \le s \le n$. Lastly the resulting truncation radius ρ will form an *s*-ball around ϕ.

By substituting equation (3.2.16) into (3.2.13) and rearranging the result we may express the approximation as:

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) = \frac{1}{\sum_{J \in I_0} g(\phi - \eta_J)} \sum_{I \in I_0} \hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_I) g(\phi - \eta_I). \tag{3.2.19}$$

Comparing this to equation (3.2.7) in section 3.2.2 we see, if we replace the coefficients $\boldsymbol{\theta}_I$ by the local models $\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I)$ and the basis functions $g(\boldsymbol{x}-\xi_I)$ by the model validity functions $g(\phi-\eta_I)$, that the approximation construction is equivalent to a Gaussian RBF approximation except the output is normalised by the sum of the contributing model validity functions.

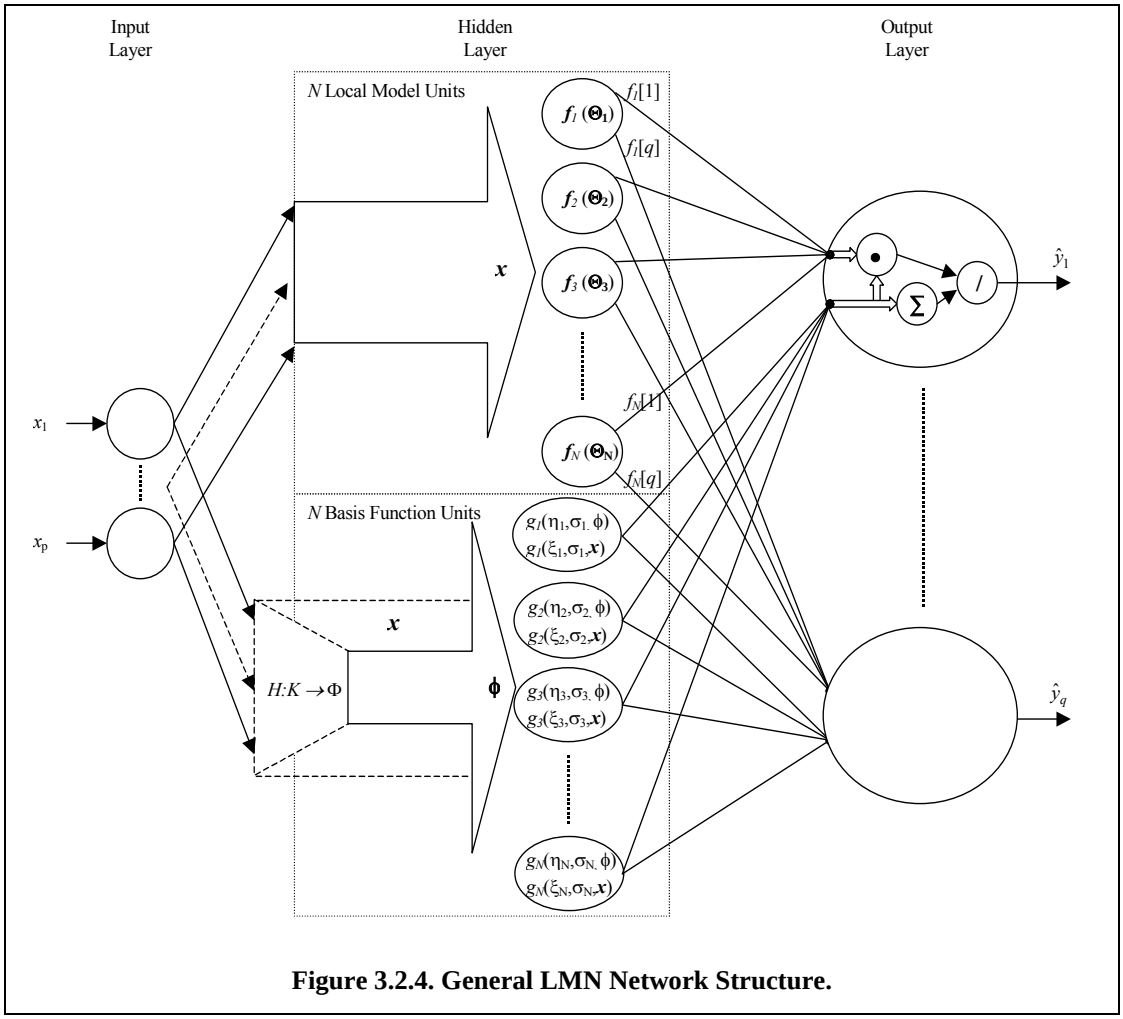


Figure 3.2.4. General LMN Network Structure.

The LMN can thus be viewed as a modified RBF network, however the sampling lattice density and dimension are now determined by the characteristics of ϕ. Such an approach leads to the network structure depicted in Figure 3.2.4. Consistent with the general RBF network construction we see that such an LMN structure has an input, hidden and output layer, with input and output layers sizing according to number of inputs and outputs respectively. However the hidden layer, input to hidden layer connectivity, and output unit calculations are significantly different to an equivalent RBF network.

The hidden layer now consists of two different types of units; namely the local model units and the corresponding basis function units. There is an equal number N of each unit type[10] where N may be expressed as:

$$N = \prod_{j=n-s+1}^{n} \max i_j \quad \forall i_j \in L. \tag{3.2.20}$$

Equation (3.2.20) clearly shows that dramatic reductions in N may result from decreasing s. As with RBF networks, in the special case where the basis vectors $\boldsymbol{e}_j$ are orthogonal, the resulting network is referred to as an axis orthogonal LMN network. For this type of network we may show, using the same approach as before, that although there are N basis function units it is possible to reduce the number of exponentials evaluated to:

$$N_o = \sum_{j=n-s+1}^{n} \max i_j \quad \forall i_j \in L. \tag{3.2.21}$$

The computational efficiencies resulting from the reduction in N and N_o are partially offset by the added burden of computing the local models and obviously this trade off is a key factor in choosing between RBF and LMN approximations.

The basis function units must be connected to the input units associated with $\boldsymbol{x}_H$ to construct the mapping H. There are two basic approaches to achieving this. The first approach divides the input to the basis function unit connections into two groups, each associated with $\boldsymbol{x}_H$ and $\boldsymbol{x}_L$ respectively. Only the first of these groups, which represents the lattice η_L, is connected to the basis function units. The second approach, the one utilized in this work, streamlines the code implementation but has a small computationally penalty associated with it. The approach works by specifying, during initial network definition, very large variances and sample spacing, in comparison to the extents of K, along each of the dimensions associated with $\boldsymbol{x}_L$. This causes a single exponential evaluation for each of these dimensions, the outcome of which is always unity. Because these outcomes are multiplied together with results from other dimensions, they play no role in the final model validity function values. However, if the mapping H were constructed in some manner other than the one described above this approach would be invalidated. In both methods, and identically to RBF networks, the weight matrices associated with the elliptic integration function are the lattice vertices and the Gaussian variances. The key difference is that in the first method the lattice points are defined by η_L whereas in the latter method lattice points are defined by ξ_L, where the components associated with $\boldsymbol{x}_L$ may be expressed by a single vector $\bar{\xi}_L = \left[\bar{\xi}_1, \bar{\xi}_2, \cdots, \bar{\xi}_{n-s}\right]$, $\bar{\xi}_j = (\max x_j + \min x_j)/2$, $\forall \boldsymbol{x} \in K, 1 \le j \le n-s$ and variances $\{\sigma_1, \sigma_2, \cdots \sigma_{n-s}\}$ are chosen such that $g(x_j - \bar{\xi}_j, \sigma_j) \approx 1$, $\forall \boldsymbol{x} \in K, 1 \le j \le n-s$.

[10]For strict conformance to neural network architecture there should be $N \times q$ local model units where each unit would have a single output. However, for both computational and diagrammatic purposes, it is much more convenient to consider each local model unit as having q outputs distributed to each of the output units.

In general, all inputs are connected to all local model units. However, if *a-priori* information indicates that any particular local model or group of local models does not make use of any particular input(s), then the unused connections may be omitted. Notice that the local models described by equation (3.2.14) may be written as:

$$\begin{aligned}\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I) &= \hat{\Theta}_I^T(\boldsymbol{x}-\xi_I) \\ &= \Theta_I^T\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}, \quad \Theta_I \in \Re^{(n+1)\times q}\end{aligned} \tag{3.2.22}$$

where the constant vector formed by the product $-\hat{\Theta}_I^T\xi_I$ has been absorbed into the n^{th}+1 column of Θ_I. The weights associated with the input to local model connections are thus the elements of Θ_I, the last column of which represents a bias, and each local model unit performs a linear integration function computation.

The activation function computation for each local model unit simply copies the integration function result to the output; functionally a multiplication by unity. Notice that if the mapping *H* is evaluated according to the second method described above, then $\phi = \boldsymbol{x}$ and $\eta_L = \xi_I$.

The output units combine the two elements of the hidden layer, using the computations shown in equation (3.2.19), to produce the final output. Each unit has two vector inputs containing *N* elements each. For the j^{th} output unit, the elements of the first input vector are the j^{th} output from each of the *N* hidden local model units. The second input vector's elements, which are the same in all the output units, are the *N* outputs from the basis function units. For both vector inputs the connection weights are unity. To generate the output the unit first performs the inner vector product (the integration function) between the two input vectors. Next, this scalar result is divided by the sum of the elements of the second input, the normalisation term, to produce the j^{th} output. This final step thus constitutes the output unit's activation function. Notice that the normalisation term is identical for each output unit, and thus need only be computed once in a typical serial programmatic implementation. The preceding discussion permits the overall mapping, after substitution of equation (3.2.22) into (3.2.13), to be expressed as:

$$\hat{\boldsymbol{y}} = \sum_{I\in I_0}\Theta_I^T\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}\hat{g}(\boldsymbol{x}-\xi_I)\,. \tag{3.2.23}$$

By judicious construction of matrices, it is possible to express equation (3.2.23) as a matrix equation. Assume that for some particular input the set I_0 contains l vector elements, then let:

$$\Theta_{I_0} = \left[\Theta_{I_1}^T \;\middle|\; \Theta_{I_2}^T \;\middle|\; \cdots \;\middle|\; \Theta_{I_l}^T\right]^T, \quad \forall I_{[1\cdots l]} \in I_0, \Theta_{I_0} \in \Re^{l(n+1)\times q}$$
$$G(\boldsymbol{x}, \xi_{I_0}) = \begin{bmatrix} \hat{g}(\boldsymbol{x} - \xi_{I_1}) \\ \hat{g}(\boldsymbol{x} - \xi_{I_2}) \\ \vdots \\ \hat{g}(\boldsymbol{x} - \xi_{I_l}) \end{bmatrix} \tag{3.2.24}$$

then:

$$\hat{\boldsymbol{y}} = \Theta_{I_0}^T \left(G(\boldsymbol{x}, \xi_{I_0}) \otimes [\boldsymbol{x}^T, 1]^T \right). \tag{3.2.25}$$

As was pointed out earlier the hidden layer element comprised of the basis function units is identical to a RBF network. This means that the discussion presented in section 3.2.2 is directly applicable to this element of LMN's and we may develop an almost identical activation rule implementation. Such an implementation is shown in Listing 3.2.3. Notice that lines 10 through 180 are identical to those used in Listing 3.2.2, however there are two implicit differences. Firstly, for comparable approximations, the vectors `net.center{j}`, `net.invsigma{j}` and consequently `expn` would generally be significantly reduced in length due to lower lattice density requirements. A shorter `expn` results in far fewer exponential calculations in the remainder of the loop. Secondly, the index vector, which previously identified the weights used in calculating the output, now identify local model units used in the output calculation.

Thus, once execution has proceeded to line 180 the `actv` vector will contain the basis unit interpolation

```
10  actv=1;idxm=0;                                  % Initialize variables
20  for j=1:net.ninp                                % Loop through each network input
30      expn=(u(j)-net.center{j}).*net.invsigma{j}; % Find exponents for jth input
40      idxe=find(abs(expn)<net.nthld1);            % Determine exponentials activated
50      expn=expn(idxe); actv=actv(:)*exp(-expn.^2); % Multiply with active units
60      vct1=((idxe-(j>1)).*net.vcprd(j));          % Calculate new indexes
70      vct1=vct1(ones(size(idxm,1),1),:);          % Copy vct1 to length(tmp2) rows
80      matx=idxm(:,ones(1,size(idxe,2)));          % Copy idxm to length(tmp1) cols
130     idxm=vct1(:)+matx(:);                       % Update model index vector
140     idxa=find(actv>net.nthld2);                 % Make truncation region a ball
150     actv=actv(idxa);                            % Remove inactive units
160     idxm=idxm(idxa);                            % Remove inactive model indexes
180 end                                             % End for loop
190 actv=permute(actv(:)./sum(actv(:)),[3,2,1]);    % Reshape actv and normalise
200 arry1=actv(ones(1,net.nout),...                 % Each model validity fcn times mdl wts
          [1, ones(1,net.ninp)],:).*net.weights(:,:,idxm);
210 arry2=[u',1];                                   % Append bias term to input
220 arry3=permute(sum(arry1.*...                    % Calc weighted output of each active mdl
          arry2(ones(1,net.nout),:,ones(1,length(idxm))),2),[1,3,2]);
250 yhat=sum(arry3,2);                              % Sum model outputs for network output
```

Listing 3.2.3. Axis Orthogonal LMN Activation Rule Implementation.

function results, and `idxm` contains the index vector identifying each local model contributing to the final result. From this point on (lines 190 to 250) the code becomes unique to the LMN activation rule implementation. In line 190 the `activ` vector is normalized to form a vector of model validity functions. It also undergoes a permutation operation to form a three dimensional array containing one row, one column, and l 'layers' where l is the number of contributing local models. Next in lines 200 to 220 three temporary three dimensional array variables are calculated. The first, `arry1`, represents:

$$\texttt{arry1} = \Theta_I^T \hat{g}(\boldsymbol{x} - \xi_I) \quad \forall I \in I_0 . \tag{3.2.26}$$

It is calculated by repeating the single row and column entry of `actv`, for each of the l layers, by the number of outputs and inputs+1 respectively, and then performing an element by element multiplication with the weight matrix for each of the contributing local models. The next temporary variable, `arry2`, augments a one onto the transpose of the input column as shown in equation (3.2.22). Lastly `arry3` represents:

$$\texttt{arry3} = \hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_I)\hat{g}(\boldsymbol{x} - \xi_I) \quad \forall I \in I_0 . \tag{3.2.27}$$

It is obtained by repeating `arry2` l times along dimension three and the number of output times along dimension one and then performing an element by element multiplication between `arry1` and `arry2`. This result is summed along dimension two and then permuted to form a matrix with the same number of rows and columns as network outputs and inputs respectively. The last step in obtaining the output, performed in line 250, is the final summation shown in equation (3.2.23). Summing `arry3` along the dimension representing the individual weighted local models, namely the second dimension, does precisely this.

The reader may be puzzled as to why, what appears to be a rather convoluted method has been used to obtain the final network outputs. The reasons are twofold. Firstly one may be tempted to use the matrix multiplication operators when calculating the local model results. This leads to using a 'for' loop which, when using the Matlab interpreter, can add significant computational overhead. The second reason, as we shall see later, is that by using the temporary variables defined we can avoid redundant calculations when computing the network Jacobian. These factors combined lead to an implementation with speed improvements in excess of 30% when compared to a direct matrix multiplication implementation.

3.3 TRAINING THE NETWORK PARAMETERS – THE LEARNING RULE

In a broad context the neural network system is used in this work to perform system identification. System identification is the process of experimentally determining a system model and consists of the following key steps: experiment planning, selection of model structure, criteria postulation, parameter estimation and model validation. The network environment influences the experiment planning stage and the plan must ensure that the network inputs meet certain criteria in order for the identification to be successful. The model structure is defined by the network structure or activation rule. The postulated criterion is constructed from the objective function while the parameter estimation task is defined by the method used to generate the learning rule. Finally the model validation step is required to ensure that the model predicts the identified system with sufficient fidelity and robustness in regions of the operating space that are not necessarily explicitly included during the parameter estimation process. There is a large body of work dealing with system identification and most of the

work presented in this section is drawn from these works. In particular, the reader is referred to Åström and Wittenmark (1995) and Ljung (1999) and the many references cited therein for a more detailed discussion.

3.3.1 THE OPTIMISATION PROBLEM

The process of training or adapting the network parameters combines the postulated criteria with the learning rule to update the systems parameters based on the systems current and past experiences. These two elements are inextricably linked by the fact that the learning rule results from the method selected to solve the non-linear optimisation problem defined by the objective function.

We therefore begin by defining a cost function or criterion that should be optimised. In discrete time systems this is often expressed as:

$$J(t,\theta) = \sum_{k=1}^{t} \ell(\varepsilon(k,\theta)) \tag{3.3.1}$$

where, for now we shall consider θ to be either a matrix or vector of parameters, $\varepsilon(k,\theta)$ is a generalised error measurement, *t* is the total number of discrete unit time steps processed where each step is represented by the integer value *k***.** The function $\varepsilon(k,\theta)$ is commonly chosen, as in this work, to be the prediction error:

$$\varepsilon(k,\theta) = \boldsymbol{y}[k] - \hat{\boldsymbol{y}}[k] \tag{3.3.2}$$

where $\boldsymbol{y}[k]$ is the observed system output and $\hat{\boldsymbol{y}}[k]$ is the network predicted system output parameterised by θ. We choose the objective function $\ell(\ )$ to be consistent with Gauss's principle of least squares which states that the unknown parameters of a model should be chosen such that 'the sum of the squares of the differences between the actually observed and the computed values, multiplied by numbers that measure the degree of precision, is a minimum.'

Thus let Λ^{-1} be a $q \times q$ diagonal matrix of weights describing the precision associated with each output. Furthermore, as the system being identified may be time varying let $\beta(t,k)$ be a factor that discounts the degree of precision as a function of time, then we may use the following quadratic function to map network outputs into a real number reflecting how well the network output is tracking its computation goal:

$$\ell(k,\theta) = \frac{1}{2}\beta(t,k)\varepsilon(k,\theta)^T \Lambda^{-1}\varepsilon(k,\theta)\,. \tag{3.3.3}$$

Finally, substituting equation (2.4.4) results in the following weighted least squares criteria or cost function:

$$\begin{aligned} J(t,\theta) &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\varepsilon(k,\theta)^T \Lambda^{-1}\varepsilon(k,\theta) \\ &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\left(\boldsymbol{y}[k] - \hat{\boldsymbol{y}}[k]\right)^T \Lambda^{-1}\left(\boldsymbol{y}[k] - \hat{\boldsymbol{y}}[k]\right) \\ &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\left(\boldsymbol{y}[k] - \hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)\right)^T \Lambda^{-1}\left(\boldsymbol{y}[k] - \hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)\right). \end{aligned} \tag{3.3.4}$$

Clearly, the better the network performs the smaller the value of the criteria. Combining the above, the optimisation problem to be solved in this work may be stated as:

$$\theta^*(t) = \arg\min_{\theta}\left(\frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\left(\boldsymbol{y}[k]-\hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)\right)^T\Lambda^{-1}\left(\boldsymbol{y}[k]-\hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)\right)\right). \tag{3.3.5}$$

Equation (3.3.5) is an unconstrained minimization problem. If other limitations are placed on any of the variables, for example the parameters are limited in their range, then these limitations should be stated as part of equation (3.3.5), which would clearly then become a constrained minimization problem.

3.3.2 ON-LINE VS BATCH MODE PROCESSING

Before presenting the methods used to solve equation (3.3.5) it is necessary to make a fundamental decision about whether the training should be performed after all the system data has been obtained or whether the algorithm should attempt to incrementally include new information as time progresses. The former approach is generally referred to as off-line or batch mode processing while the latter is called on-line or recursive identification.

Given the data $\boldsymbol{Z}(t) = \left[\boldsymbol{x}[1], \boldsymbol{y}[1], \boldsymbol{x}[2], \boldsymbol{y}[2], \cdots, \boldsymbol{x}[t], \boldsymbol{y}[t]\right]^T$ we can define the identification problem as:

$$\theta(t) = \mathrm{F}\left(t, \boldsymbol{Z}(t)\right) \tag{3.3.6}$$

where the function $\mathrm{F}(.)$ is implicitly defined by equation (3.3.5). In batch mode all the data is collected before attempting to solve for θ. The evaluation of $\mathrm{F}(.)$ thus involves an unforeseen amount of calculation. In an adaptive control application this is problematic as memory is limited and only a finite amount of computation time is available between system samples. To overcome this problem in recursive or on-line algorithms a fixed dimension variable representing the accumulated information state, $P(t)$, is defined. Thus:

$$\begin{aligned}\theta(t) &= \mathrm{h}\left(P(t)\right)\\ P(t) &= \mathrm{H}\left(t, P(t-1), \boldsymbol{y}(t), \boldsymbol{x}(t)\right).\end{aligned}$$

The functions $\mathrm{h}(.)$ and $\mathrm{H}(.)$ are explicit expressions evaluated using a fixed number of calculations. Furthermore, as the amount of information added with each new sample is, in general, small compared to the information already accumulated, the number reflecting the new information content is typically weighted by a factor $\gamma(t)$ reflecting the value of the new information and summed linearly to the existing parameters and information state:

$$\begin{aligned}\theta(t) &= \theta(t-1) + \gamma(t)\,\mathrm{Q}_\theta\left(P(t), \boldsymbol{y}(t), \boldsymbol{x}(t)\right)\\ P(t) &= P(t-1) + \mathrm{Q}_\mathrm{P}\left(P(t-1), \boldsymbol{y}(t), \boldsymbol{x}(t)\right).\end{aligned} \tag{3.3.7}$$

The weighting factor $\gamma(t)$ is called the update step size or adaption gain and provides a mechanism whereby the trade off between tracking ability and noise sensitivity can be controlled. The function $\mathrm{Q}_\theta(.)$ provides the direction that the update step should take. Equation (3.3.6) may be considered a general statement of the learning rule for off-line or batch processing while equation (3.3.7) is the analogous general statement of the recursive or on-line adaption learning rule.

A stated objective of this work is to use the algorithms developed in non-linear adaptive neurocontrol. Use of the term 'adaptive' implies then that training of the system should be performed on-line. One may question why it is

necessary to have adaptive capabilities. It could be argued that the varying nature of the plant may be included in the non-linearity of the controller design by adding another degree of freedom. Implicit in this idea is that the plant is well understood over its entire domain of operation. Furthermore, as we have seen in the previous chapter, the problem with this approach is that all the network architectures considered have aspects of their calculation that scale exponentially with the number of degrees of freedom suggesting that it is advantageous to keep the degrees of freedom to a minimum.

The argument above taken to the opposite extreme, that is to use an adaptive approach to continually update a linear model of the plant, also has its drawbacks. Consider a plant that has piecewise or rapidly varying parameters. In order for equation (3.3.7) to track these rapidly varying parameters the adaption gain must be increased in value thus making the algorithm sensitive to noise contained within the observed values. With large adaption gains previously identified regions of operation are rapidly overwritten with new information. The identification scheme has only a short memory of previously learnt information. Although the adaptive control community has developed methods to alleviate this problem, recursive identification of a non-linear plant having parameters that vary over vastly different time scales is difficult. By using non-linear neural networks in conjunction with on-line adaptive techniques one can memorize short time scale non-linear variations in the network models while the adaptive scheme can be used to adjust to long time scale parameter variation. This permits one to trade off algorithm memory requirements with tracking response while maintaining good noise sensitivity characteristics. This idea is closely related to the idea of parallel estimators suggested in Åström and Wittenmark (1995).

3.3.3 Network Parameterisation as a Linear Optimisation Problem

It is well known that an analytical solution exists for the optimisation problem described in (3.3.5) provided the function $\hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)$ is linear in the parameters θ and the output is written in the following form:

$$\hat{\boldsymbol{y}}[k] = \hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta) = \varphi^T[k]\boldsymbol{\theta}\,. \tag{3.3.8}$$

This problem is typically referred to as a linear regression problem and the term $\varphi[k]$ is referred to as the regression vector or regressor. Note that the output vector for the RBF network expressed in equations (3.2.12) may be expressed in the form of equation (3.3.8) provided the following substitutions are made:

$$\begin{aligned} \boldsymbol{\theta} &= \mathrm{col}\left(\Theta_{I_0}\right), \quad \boldsymbol{\theta} \in \Re^{lq} \\ \varphi^T[k] &= \frac{\partial \hat{\boldsymbol{y}}[k]}{\partial \boldsymbol{\theta}} = \left[I_{q\times q} \otimes G(\boldsymbol{x}[k],\xi_{I_0})\right]^T, \quad \varphi^T \in \Re^{q\times lq} \end{aligned} \tag{3.3.9}$$

where $I_{q\times q}$ is a $q\times q$ identity matrix. Similarly for the LMN network equation (3.2.25) may also be transformed where:

$$\begin{aligned} \boldsymbol{\theta} &= \mathrm{col}\left(\Theta_{I_0}\right), \quad \boldsymbol{\theta} \in \Re^{l(n+1)q} \\ \varphi^T[k] &= \frac{\partial \hat{\boldsymbol{y}}[k]}{\partial \boldsymbol{\theta}} = \left[I_{q\times q} \otimes \left(G(\boldsymbol{x}[k],\xi_{I_0}) \otimes [\boldsymbol{x}^T[k],1]^T\right)\right]^T, \quad \varphi^T \in \Re^{q\times l(n+1)q}\,. \end{aligned} \tag{3.3.10}$$

If, as in the case of MLP networks represented by equation (3.2.5), the function $\hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)$, is non-linear in θ but continuously differentiable of order one, then it may be transformed into a linear problem at any instant in time by observing that:

$$\begin{aligned}\hat{\boldsymbol{y}}[k] &= \left.\frac{\partial \hat{\boldsymbol{f}}(\boldsymbol{x}[k],\theta)}{\partial\theta}\right|_{\theta=\theta[k-1]} \theta[k-1] \\ &= \varphi^T(k,\theta)\theta[k-1].\end{aligned} \tag{3.3.11}$$

Note that the regressor remains a function of the parameters. To highlight this fact equation (3.3.11) is referred to as a pseudo-linear regression problem. To relate this to the parameters of the MLP network a more convenient weight vector is first defined for each layer:

$$\widetilde{\boldsymbol{\theta}}^{(l)} = \left[\boldsymbol{\theta}_1^{(l)T}, \boldsymbol{\theta}_2^{(l)T}, \boldsymbol{\theta}_3^{(l)T}, \cdots, \boldsymbol{\theta}_{m_l}^{(l)T}\right] = \left[\operatorname{col}\left(\Theta^{(l)T}\right)\right]^T, \quad \widetilde{\boldsymbol{\theta}}^{(l)} \in \Re^{1\times(m_{l-1}+1)m_l} \tag{3.3.12}$$

where all variables have the same meaning as those in equation (3.2.2). The MLP mapping can now be stated in the form of (3.3.11) given the following substitutions:

$$\begin{aligned}\boldsymbol{\theta} &= \begin{bmatrix}\widetilde{\boldsymbol{\theta}}^{(1)T} \\ \widetilde{\boldsymbol{\theta}}^{(2)T} \\ \vdots \\ \widetilde{\boldsymbol{\theta}}^{(L)T}\end{bmatrix}, \quad \boldsymbol{\theta} \in \Re^{\sum_{l=1}^{L}(m_{l-1}+1)m_l} \\ \varphi^T(k,\boldsymbol{\theta}) &= \left.\left[\frac{\partial\hat{\boldsymbol{y}}[k]}{\partial\widetilde{\boldsymbol{\theta}}^{(1)}}, \frac{\partial\hat{\boldsymbol{y}}[k]}{\partial\widetilde{\boldsymbol{\theta}}^{(2)}}, \cdots, \frac{\partial\hat{\boldsymbol{y}}[k]}{\partial\widetilde{\boldsymbol{\theta}}^{(L)}}\right]\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[k-1]}, \quad \varphi^T \in \Re^{q\times\left(\sum_{l=1}^{L}(m_{l-1}+1)m_l\right)}.\end{aligned} \tag{3.3.13}$$

Summarising equations (3.3.9), (3.3.10) and (3.3.13), we see that the learning rule for the RBF, LMN and MLP networks can be stated as linear regression or pseudo-linear regression problems respectively. The next three sections are therefore devoted to describing the techniques used in this work to solve the general recursive regression problem. To simplify the notation these discussions will not differentiate between $\varphi^T(k,\theta)$ and $\varphi^T[k]$, both of which will simply be referred to as $\varphi^T(k)$ or $\varphi^T(t)$ when $k=t$. Furthermore, the dimensions of $\boldsymbol{\theta}$ and $\varphi^T(k)$ will be generalized to r and $q\times r$ respectively, where r is considered to take on the appropriate value depending on the network being considered. Note that the general literature contains many varied methods to solve this problem. Those discussed here are used to illustrate the core ideas found in these methods and will culminate in a newly developed algorithm found in section 3.3.8. Algorithms actually implemented are designated by a superscript hash (#) character after the equation number.

3.3.4 The Steepest or Gradient Descent Method

We begin with the simplest approach, namely the steepest or gradient descent method. Here the parameter estimate is updated at each point in time, by moving in the negative direction of the gradient of the objective function. Therefore, applying the chain rule to equation (3.3.3) gives:

$$\begin{aligned}\frac{-\partial \ell(t,\theta)}{\partial \theta} &= \frac{-\partial}{\partial \theta}\left(\frac{1}{2}\beta(t,t)\big(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\big)^T \Lambda^{-1}\big(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\big)\right)\\ &= \frac{1}{2}\beta(t,t)\left(\left(\frac{\partial \hat{\boldsymbol{y}}(t)}{\partial \boldsymbol{\theta}(t-1)}\right)^T \Lambda^{-1}\big(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\big)+\big(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\big)^T \Lambda^{-1}\left(\frac{\partial \hat{\boldsymbol{y}}(t)}{\partial \boldsymbol{\theta}(t-1)}\right)\right)\\ &= \beta(t,t)\varphi(t)\Lambda^{-1}\big(\boldsymbol{y}(t)-\varphi^T(t)\boldsymbol{\theta}(t-1)\big).\end{aligned} \tag{3.3.14}$$

Assuming $\beta(t,t)$ remains constant, i.e. $\gamma = \beta(t,t)$, and that the accumulated information state, $P(t)$, represents the negative direction of the gradient of the objective function, the gradient descent learning rule may be stated as:

$$\begin{aligned}\boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1)+\gamma \boldsymbol{P}(t)\\ \boldsymbol{P}(t) &= \varphi(t)\Lambda^{-1}\big(\boldsymbol{y}(t)-\varphi^T(t)\boldsymbol{\theta}(t-1)\big).\end{aligned} \tag{3.3.15}^{\#}$$

This method is very inefficient when the hyper surface representing the objective function has long narrow valleys or large flat plateaus as the gradient term approaches zero in these regions. In an attempt to alleviate this problem another term, called momentum is frequently added:

$$\begin{aligned}\boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1)+\gamma \boldsymbol{P}(t)\\ \boldsymbol{P}(t) &= \varphi(t)\Lambda^{-1}\big(\boldsymbol{y}(t)-\varphi^T(t)\boldsymbol{\theta}(t-1)\big)+\mu P(t-1).\end{aligned} \tag{3.3.16}^{\#}$$

The momentum term attempts to keep the updates 'moving' by summing a weighted factor of the previous direction with the current direction. The idea is that components of the update that have changed from the previous iteration will tend to cancel but components in the same direction will sum together to 'push' the update further along the desired path. Although this approach can sometimes improve the convergence rate the method still remains inefficient. Furthermore, the 'best' values for γ and μ are problem dependent and must be selected empirically. The advantage of the algorithm is that, in spite of its lack of mathematical rigor, it is intuitively simple; note that at each time step the only complex calculation required is determining the matrix $\varphi(t)$.

3.3.5 THE RECURSIVE LEAST SQUARES (RLS) METHOD

In section 3.3.1 the selected criterion was called the weighted least squares criterion. It is well known that a mathematically sound analytical solution exists to this type of problem. Introducing the following matrix notations:

$$\begin{aligned}Y(t) &= \left[\boldsymbol{y}^T(1),\boldsymbol{y}^T(2),\cdots,\boldsymbol{y}^T(t)\right]^T\\ \hat{Y}(t) &= \left[\hat{\boldsymbol{y}}^T(1),\hat{\boldsymbol{y}}^T(2),\cdots,\hat{\boldsymbol{y}}^T(t)\right]^T\\ E_y(t) &= Y(t)-\hat{Y}(t), \quad \varepsilon_y(t) = \boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\\ \Phi(t) &= \begin{bmatrix}\varphi^T(1)\\ \varphi^T(2)\\ \vdots\\ \varphi^T(t)\end{bmatrix}\\ W_y(t) &= \begin{bmatrix}\beta(t,1)\Lambda^{-1} & & \\ & \ddots & \\ & & \beta(t,t)\Lambda^{-1}\end{bmatrix}, \quad \Lambda^{-1}\in \Re^{qxq}\end{aligned} \tag{3.3.17}$$

the cost function (3.3.4) then becomes:

$$\begin{aligned} J(\boldsymbol{\theta},t) &= \frac{1}{2}E_y^T(t)W_y(t)E_y(t) \\ &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\left(\left(\boldsymbol{y}[k]-\varphi^T[k]\boldsymbol{\theta}\right)^T\Lambda^{-1}\left(\boldsymbol{y}[k]-\varphi^T[k]\boldsymbol{\theta}\right)\right) \\ &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\left(\boldsymbol{y}^T[k]\Lambda^{-1}\boldsymbol{y}[k]-2\boldsymbol{\theta}^T\varphi[k]\Lambda^{-1}\boldsymbol{y}[k]+\boldsymbol{\theta}^T\varphi[k]\Lambda^{-1}\varphi^T[k]\boldsymbol{\theta}\right). \end{aligned} \tag{3.3.18}$$

Equation (3.3.18) may be differentiated directly with respect to $\boldsymbol{\theta}$:

$$\begin{aligned} \frac{\partial J(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}} &= \sum_{k=1}^{t}\beta(t,k)\left(\varphi[k]\Lambda^{-1}\varphi^T[k]\boldsymbol{\theta}-\varphi[k]\Lambda^{-1}\boldsymbol{y}[k]\right) \\ &= \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\varphi^T[k]\boldsymbol{\theta}-\sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\boldsymbol{y}[k]. \end{aligned} \tag{3.3.19}$$

To find the minimum we set equation (3.3.19) equal to zero:

$$\begin{aligned} &\frac{\partial J(\boldsymbol{\theta}^*,t)}{\partial\boldsymbol{\theta}^*} = P^{-1}(t)\boldsymbol{\theta}^*-\widetilde{Y}(t)=0 \\ &\therefore P^{-1}(t)\boldsymbol{\theta}^* = \widetilde{Y}(t), \quad \boldsymbol{\theta}^*\in\Re^r \end{aligned} \tag{3.3.20}$$

where:

$$\begin{aligned} P^{-1}(t) &= \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\varphi^T[k] = \Phi^T(t)W_y(t)\Phi(t) \\ \widetilde{Y}(t) &= \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\boldsymbol{y}[k] = \Phi^T(t)W_y(t)Y(t). \end{aligned} \tag{3.3.21}$$

Therefore the minimum of (3.3.18) occurs at:

$$\begin{aligned} \boldsymbol{\theta}^*(t) &= \left(\Phi^T(t)W_y(t)\Phi(t)\right)^{-1}\Phi^T(t)W_y(t)Y(t) \\ &= P(t)\widetilde{Y}(t). \end{aligned} \tag{3.3.22}$$

This is called the normal equation and is the basis for many off-line algorithms. To derive a recursive algorithm we begin by defining the variable $\lambda(t)$ as follows:

$$\lambda(t) = \begin{cases} \beta(t,k)/\beta(t-1,k) & 0<k<t \\ 1/\beta(t-1,k) & k=t \end{cases}. \tag{3.3.23}$$

Substituting (3.3.23) into (3.3.21) gives:

$$\begin{aligned} P^{-1}(t) &= \lambda(t)\sum_{k=1}^{t}\beta(t-1,k)\varphi(k)\Lambda^{-1}\varphi^T(k) \\ &= \lambda(t)\sum_{k=1}^{t-1}\beta(t-1,k)\varphi(k)\Lambda^{-1}\varphi^T(k)+\lambda(t)\beta(t-1,t)\varphi(t)\Lambda^{-1}\varphi^T(t) \\ &= \lambda(t)\sum_{k=1}^{t-1}\beta(t-1,k)\varphi(k)\Lambda^{-1}\varphi^T(k)+\varphi(t)\Lambda^{-1}\varphi^T(t) \\ &= \lambda(t)P^{-1}(t-1)+\varphi(t)\Lambda^{-1}\varphi^T(t). \end{aligned} \tag{3.3.24}$$

Similarly:

$$\begin{aligned}\widetilde{Y}(t) &= \lambda(t)\sum_{k=1}^{t-1}\beta(t-1,k)\varphi[k]\Lambda^{-1}\boldsymbol{y}[k] + \lambda(t)\beta(t-1,t)\varphi[t]\Lambda^{-1}\boldsymbol{y}[t] \\ &= \lambda(t)\widetilde{Y}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t))\,.\end{aligned} \tag{3.3.25}$$

From equation (3.3.24) we note that:

$$P^{-1}(t-1) = \frac{1}{\lambda(t)}\left(P^{-1}(t) - \varphi(t)\Lambda^{-1}\varphi^T(t)\right) \tag{3.3.26}$$

and from equation (3.3.22):

$$\widetilde{Y}(t-1) = P^{-1}(t-1)\boldsymbol{\theta}^*(t-1)\,. \tag{3.3.27}$$

Then substituting (3.3.24) through (3.3.27) back into (3.3.22) gives:

$$\begin{aligned}\boldsymbol{\theta}^*(t) &= P(t)\left(\lambda(t)\widetilde{Y}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\right) \\ &= P(t)\left(\lambda(t)P^{-1}(t-1)\boldsymbol{\theta}^*(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\right) \\ &= P(t)\left(\left(P^{-1}(t) - \varphi(t)\Lambda^{-1}\varphi^T(t)\right)\boldsymbol{\theta}^*(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\right) \\ &= \boldsymbol{\theta}^*(t-1) + P(t)\varphi(t)\Lambda^{-1}\left(\boldsymbol{y}(t) - \varphi^T(t)\boldsymbol{\theta}^*(t-1)\right) \\ &= \boldsymbol{\theta}^*(t-1) + P(t)\varphi(t)\Lambda^{-1}\left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right).\end{aligned} \tag{3.3.28}$$

In conventional RLS algorithms the matrix inversion required to calculate $P(t)$ in equation (3.3.28) is avoided by applying the matrix inversion lemma:

$$\left(A + BCD\right)^{-1} = A^{-1} - A^{-1}B\left(DA^{-1}B + C^{-1}\right)^{-1}DA^{-1}\,. \tag{3.3.29}$$

Thus, taking $A = \lambda(t)P^{-1}(t-1),\ B = D^T = \varphi(t),\ C = \Lambda^{-1}$ equation (3.3.24) gives:

$$P(t) = \frac{1}{\lambda(t)}\left(P(t-1) - P(t-1)\varphi(t)\left(\lambda(t)\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1)\right). \tag{3.3.30}$$

Moreover, if we let $K(t) = P(t)\varphi(t)\Lambda^{-1}$ then substituting (3.3.30) and simplifying gives:

$$K(t) = P(t-1)\varphi(t)\left(\lambda(t)\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}. \tag{3.3.31}$$

Summarizing equations (3.3.28), (3.3.30) and (3.3.31) gives the final recursive least squares algorithm:

$$\begin{aligned}\boldsymbol{\theta}^*(t) &= \boldsymbol{\theta}^*(t-1) + K(t)\left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right) \\ K(t) &= P(t-1)\varphi(t)\left(\lambda(t)\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1} \\ P(t) &= \frac{1}{\lambda(t)}\left(P(t-1) - K(t)\varphi^T(t)P(t-1)\right).\end{aligned} \tag{3.3.32}$$

Notice that to compute $K(t)$ in (3.3.32) it is still necessary to obtain the inverse of a $q \times q$ matrix. This is considerably less computational effort than would be required to compute the inverse[11] of $P(t)$ if the matrix inversion lemma had not been applied to equation (3.3.24).

3.3.6 THE EXPONENTIAL FORGETTING FACTOR WITH CONDITIONAL UPDATING METHOD

In a time varying system it is desirable to assign less weight to older measurements that are no longer representative of the system. This is done by selecting the values of $\beta(t,k)$ along the diagonal of the weight

[11] $\dim(P(t)) = \left(\dim(\boldsymbol{\theta}) \times \dim(\boldsymbol{\theta})\right)$

matrix W_y in (3.3.17). In particular if we choose $\beta(t,k) = \lambda^{t-k}$ were $\lambda < 1$ then we see that the current sample has unity weighting while older measurements are exponentially discounted. Furthermore, equation (3.3.23) reduces to $\lambda(t) = \lambda$ for all $k < t$ permitting λ to be substituted for $\lambda(t)$ in equation (3.3.32). The constant λ is often called the forgetting factor and the resulting RLS algorithm is called the exponential forgetting algorithm:

$$\begin{aligned} \boldsymbol{\theta}^*(t) &= \boldsymbol{\theta}^*(t-1) + K(t)\big(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\big) \\ K(t) &= P(t-1)\varphi(t)\big(\lambda\Lambda + \varphi^T(t)P(t-1)\varphi(t)\big)^{-1} \\ P(t) &= \frac{1}{\lambda}\big(P(t-1) - K(t)\varphi^T(t)P(t-1)\big). \end{aligned} \qquad (3.3.33)^{\#}$$

3.3.6.1 Relationship between Adaption Gain and the forgetting Factor

It is informative to compare the forgetting factor to the adaption gain in equation (3.3.7). By normalizing $P(t)$ such that $P(t) = \gamma(t)\hat{P}(t)$, and noting that $K(t) = \gamma(t)\hat{P}(t)\varphi(t)\Lambda^{-1}$, we see that the update part of equation (3.3.32) takes the same form as equation (3.3.7). Furthermore:

$$\hat{P}^{-1}(t) = \sum_{k=1}^{t}\varphi(k)\Lambda^{-1}\varphi^T(k) = \frac{\sum_{k=1}^{t}\beta(t,k)\varphi(k)\Lambda^{-1}\varphi^T(k)}{\sum_{k=1}^{t}\beta(t,k)} = P^{-1}(t)\gamma(t)$$

$$\therefore \quad \frac{1}{\gamma(t)} = \sum_{k=1}^{t}\beta(t,k)\,.$$

But from equation (3.3.23) we get:

$$\begin{aligned} &\sum_{k=1}^{t}\beta(t,k) = \sum_{k=1}^{t-1}\beta(t,k) + \beta(t,t) = \frac{\beta(t,k)}{\beta(t-1,k)}\sum_{k=1}^{t-1}\beta(t-1,k) + 1 \\ &\therefore \quad \frac{1}{\gamma(t)} = \frac{\lambda(t)}{\gamma(t-1)} + 1\,. \end{aligned} \qquad (3.3.34)$$

When using the exponential forgetting algorithm the forgetting factor becomes a constant allowing (3.3.34) to be reduced to:

$$\gamma = 1 - \lambda \qquad (3.3.35)$$

providing a simple relationship between the adaption gain and forgetting factor.

3.3.6.2 Persistent Excitation and The Estimator Wind-up Phenomenon

It is important to note that equation (3.3.22) must have a unique minimum, that is, the term $\left(\Phi^T(t)W_y(t)\Phi(t)\right)$ or $P^{-1}(t)$ must be invertible or have full rank. This condition is called an excitation condition and implies that certain experimental conditions must exist for the method to be successful. In particular, the inputs to an n^{th} order regression system are said to be persistently exciting of order *n* if the matrix:

$$\begin{aligned} C_n &= \lim_{t\to\infty}\frac{1}{t}\Phi^T\Phi \\ &= \lim_{t\to\infty}\frac{1}{t}\sum_{k=1}^{t}\varphi(k)\varphi^T(k) \end{aligned} \qquad (3.3.36)$$

is bounded positive definite. As the regression vector associated with dynamic systems typically contains elements of both past inputs and outputs, problems can arise if the system inputs are poorly excited, if the model is over parameterized, or when performing system identification under closed loop low order linear feedback conditions. Periods of input inactivity can lead to duplicated inputs and outputs and feedback may result in certain inputs being directly proportional to certain outputs. This causes columns of the matrix $P^{-1}(t)$ to approach or become linearly dependent resulting in poor conditioning or reduced rank respectively. A system exhibiting this problem is said to suffer from collinearity.

The absence of persistent excitation coupled with exponential forgetting can be particularly problematic causing a phenomenon called estimator windup. Consider the case where the regressor is constant, that is, $\varphi(t) = \varphi_0$ with some initial matrix $P_0^{-1} = \varphi_0 \Lambda^{-1} \varphi_0^T$. Then, if we let k begin at zero, equation (3.3.24) gives:

$$\begin{aligned} P^{-1}(t) &= \lambda^t P_0^{-1} + \sum_{k=1}^{t} \lambda^{t-k} \varphi_0 \Lambda^{-1} \varphi_0^T \\ &= \lambda^t P_0^{-1} + \alpha^{-1}(t) \varphi_0 \Lambda^{-1} \varphi_0^T \end{aligned} \tag{3.3.37}$$

where:

$$\alpha(t) = \frac{1-\lambda}{1-\lambda^t}. \tag{3.3.38}$$

Setting $A = \lambda^t P_0^{-1}$, $B = D^T = \varphi_0$, $C = \alpha^{-1}(t)\Lambda^{-1}$ and applying the matrix inversion lemma (3.3.29) to (3.3.37) gives:

$$P(t) = \frac{1}{\lambda^t}\left(P_0 - P_0 \varphi_0 \left(\lambda^t \alpha(t) \Lambda + \varphi_0^T P_0 \varphi_0 \right)^{-1} \varphi_0^T P_0 \right). \tag{3.3.39}$$

Also from (3.3.37) we get:

$$\begin{aligned} P^{-1}(t) &= \left(\lambda^t + \alpha^{-1}(t) \right) P_0^{-1} \\ \therefore P_0 &= P(t)\left(\lambda^t + \alpha^{-1}(t) \right). \end{aligned} \tag{3.3.40}$$

Substituting (3.3.40) into (3.3.39) gives:

$$P(t) = \alpha(t) P_0 \varphi_0 \left(\lambda^t \alpha(t) \Lambda + \varphi_0^T P_0 \varphi_0 \right)^{-1} \varphi_0^T P_0 . \tag{3.3.41}$$

Matrix $P(t)$ may be expressed as two terms:

$$P(t) = \widetilde{P}(t) + \beta(t) \varphi_0 \Lambda^{-1} \varphi_0^T \tag{3.3.42}$$

where:

$$\widetilde{P}(t) = \frac{1}{\lambda^t}\left(P_0 - P_0 \varphi_0 (\lambda^t \alpha(t) \Lambda + \varphi_0^T P_0 \varphi_0)^{-1} \varphi_0^T P_0 \right) - \beta(t) \varphi_0 \Lambda^{-1} \varphi_0^T . \tag{3.3.43}$$

The value $\beta(t)$ is chosen such that matrix $\widetilde{P}(t)$ has rank $n-1$ with $\widetilde{P}(t)\varphi_0 = 0$. This implies that the parameter information components represented by $\widetilde{P}(t)$ are orthogonal to φ_0. The second term therefore represents the update to the components of the parameter information which are not orthogonal to the regressor. Combining equations (3.3.41), (3.3.42) and (3.3.43) and post multiplying by φ_0 gives:

$$\alpha(t)P_0\varphi_0\left(\lambda^t\alpha(t)\Lambda+\varphi_0^TP_0\varphi_0\right)^{-1}\varphi_0^TP_0\varphi_0=\beta(t)\varphi_0\Lambda^{-1}\varphi_0^T\varphi_0\ . \tag{3.3.44}$$

Noting that $|\lambda|<1$, and examining equations (3.3.43), (3.3.38) and (3.3.44) respectively we see that:

$$\begin{aligned}&\lim_{t\to\infty}(\widetilde{P}(t))\to\infty \text{ as } \lambda^{-t}\\&\lim_{t\to\infty}(\alpha(t))\to(1-\lambda)\\&\lim_{t\to\infty}(\beta(t))\to(1-\lambda)\left(\varphi_0\Lambda^{-1}\varphi_0^T\right)^{-2}\ .\end{aligned} \tag{3.3.45}$$

Therefore, the 'orthogonal' part of $P(t)$ goes to infinity or 'winds up' while the 'non-orthogonal' part converges to a constant matrix. That is, when the regressor $\varphi(t)$ is constant, new information is obtained only for the parameter components that are not orthogonal to the regressor. The wind up phenomenon associated with the orthogonal parameter components causes $P(t)$ to become excessively large resulting in radical and / or oscillatory changes in parameter updates.

3.3.6.3 Conditional Updating

To avoid the windup phenomenon, the use of conditional updating attempts to measure when there is sufficient excitation and permits updating of $P(t)$ and the parameters only when the excitation exceeds some chosen threshold. A convenient dimensionless measure of the excitation is desired. Consider the following pre and post multiplied version of equation (3.3.42):

$$\varphi^T(t)P(t)\varphi(t)=\varphi^T(t)\widetilde{P}(t)\varphi(t)+\varphi^T(t)\beta(t)\varphi(t)\Lambda^{-1}\varphi^T(t)\varphi(t)\ . \tag{3.3.46}$$

If the regressor is constant then as $t\to\infty$ it is evident from the previous discussion, and substitution from (3.3.45), that equation (3.3.46) becomes:

$$\begin{aligned}\lim_{t\to\infty}(\varphi^T(t)P(t)\varphi(t))&=0+\varphi^T(t)\beta(t)\varphi(t)\Lambda^{-1}\varphi^T(t)\varphi(t)\\&=\varphi^T(t)(1-\lambda)\left(\varphi(t)\Lambda^{-1}\varphi^T(t)\right)^{-2}\varphi(t)\Lambda^{-1}\varphi^T(t)\varphi(t)\\&=(1-\lambda)\Lambda\ .\end{aligned}$$

Clearly, the following limit will be approached from above if there is insufficient excitation:

$$\Lambda^{-1}\varphi^T(t)P(t+1)\varphi(t)\to I_{qxq}(1-\lambda)\ . \tag{3.3.47}$$

Therefore, given some positive quantity υ, updating should only use those elements of the matrix equations which correspond to the i^{th} diagonal element on the left hand side of (3.3.47) which satisfy:

$$\text{diag}\left(\Lambda^{-1}\varphi^T(t)P(t)\varphi(t)\right)_i>(\upsilon+1)(1-\lambda)\ . \tag{3.3.48}^{\#}$$

3.3.6.4 Directional forgetting

Another approach to avoiding the windup phenomenon is to 'forget' only in the direction of the regressor, i.e. not to apply the forgetting factor to those elements of $P^{-1}(t)$ which are orthogonal to the regressor. Recalling (3.3.24), but temporarily setting $\lambda(t)=1,\ \Lambda^{-1}=I$ gives:

$$P^{-1}(t)=P^{-1}(t-1)+\varphi(t)\varphi^T(t)\ . \tag{3.3.49}$$

Let:

$$P^{-1}(t-1) = \widetilde{P}^{-1}(t-1) + \varphi(t)\Pi(t)\varphi^T(t) \tag{3.3.50}$$

where $\widetilde{P}^{-1}(t-1)\varphi(t) = 0$ then by pre and post multiplying $P^{-1}(t-1)$ by $\varphi^T(t)$ and $\varphi(t)$ respectively it is easy to show that:

$$\Pi(t) = (\varphi^T(t)\varphi(t))^{-1}\varphi^T(t)P^{-1}(t-1)\varphi(t)(\varphi^T(t)\varphi(t))^{-1}. \tag{3.3.51}$$

Substituting (3.3.50) into (3.3.49) and applying a forgetting factor to only the non-orthogonal part of (3.3.50) yields:

$$P^{-1}(t) = \widetilde{P}^{-1}(t-1) + \lambda\varphi(t)\Pi(t)\varphi^T(t) + \varphi(t)\varphi^T(t).$$

Simplifying by substituting $\Pi(t)$ from (3.3.51) and $\widetilde{P}^{-1}(t-1)$ from a rearranged (3.3.50) gives the new update equation:

$$P^{-1}(t) = P^{-1}(t-1) + (\lambda-1)\varphi(t)(\varphi^T(t)\varphi(t))^{-1}\varphi^T(t)P^{-1}(t-1)\varphi(t)(\varphi^T(t)\varphi(t))^{-1}\varphi^T(t) + \varphi(t)\varphi^T(t).$$

3.3.6.5 The Square Root Algorithm

The required inversion of $\left(\Phi^T(t)W_y(t)\Phi(t)\right)$, whether done using the matrix inversion lemma or solving the normal equations directly, is the basic source of collinearity related problems. One can try to enhance the numerical properties of this process so that poor conditioning is minimized. This is the key idea behind the orthogonalization or the square root algorithm approach. In this approach it is assumed that $P(t)$ can be represented as the product of matrices that are updated in an iterative manner.

Let $P(t)$ be represented by the Cholesky factorization, i.e. $P(t) = \overline{P}(t)\overline{P}^T(t)$. Construct a matrix:

$$\Gamma(t) = \begin{bmatrix} \sqrt{\lambda(t)\Lambda} & 0 \\ \overline{P}^T(t-1)\varphi(t) & \overline{P}^T(t-1) \end{bmatrix} \tag{3.3.52}$$

which is factored into the product of an orthogonal and upper triangular matrix by QR factorization to give:

$$\Gamma(t) = Q(t)\begin{bmatrix} R_{1,1}(t) & R_{1,2}(t) \\ 0 & R_{2,2}(t) \end{bmatrix} \tag{3.3.53}$$

where $R_{1,1}(t)$ and $R_{2,2}(t)$ are both upper triangular.

Multiplying (3.3.53) by its transpose and noting that $Q^T(t)Q(t) = I$ gives:

$$\begin{aligned} \overline{\Gamma}(t) = \Gamma^T(t)\Gamma(t) &= \begin{bmatrix} R_{1,1}^T(t) & 0 \\ R_{1,2}^T(t) & R_{2,2}^T(t) \end{bmatrix} Q^T(t)Q(t) \begin{bmatrix} R_{1,1}(t) & R_{1,2}(t) \\ 0 & R_{2,2}(t) \end{bmatrix} \\ &= \begin{bmatrix} R_{1,1}^T(t)R_{1,1}(t) & R_{1,1}^T(t)R_{1,2}(t) \\ R_{1,2}^T(t)R_{1,1}(t) & R_{1,2}^T(t)R_{1,2}(t) + R_{2,2}^T(t)R_{2,2}(t) \end{bmatrix} = \begin{bmatrix} \overline{\Gamma}_{1,1}(t) & \overline{\Gamma}_{1,2}(t) \\ \overline{\Gamma}_{2,1}(t) & \overline{\Gamma}_{2,2}(t) \end{bmatrix}. \end{aligned} \tag{3.3.54}$$

Similarly multiplying (3.3.52) by its transpose gives:

$$\begin{aligned}\overline{\Gamma}(t) &= \Gamma^T(t)\Gamma(t) = \begin{bmatrix} \left(\sqrt{\lambda(t)\Lambda}\right)^T & \varphi^T(t)\overline{P}(t-1) \\ 0 & \overline{P}(t-1) \end{bmatrix} \begin{bmatrix} \sqrt{\lambda(t)\Lambda} & 0 \\ \overline{P}^T(t-1)\varphi(t) & \overline{P}^T(t-1) \end{bmatrix} \\ &= \begin{bmatrix} \lambda(t)\Lambda + \varphi^T(t)P(t-1)\varphi(t) & \varphi^T(t)P(t-1) \\ P(t-1)\varphi(t) & P(t-1) \end{bmatrix} = \begin{bmatrix} \overline{\Gamma}_{1,1}(t) & \overline{\Gamma}_{1,2}(t) \\ \overline{\Gamma}_{2,1}(t) & \overline{\Gamma}_{2,2}(t) \end{bmatrix}\end{aligned} . \tag{3.3.55}$$

Notice that due to symmetry $\overline{\Gamma}_{1,1} = \overline{\Gamma}_{1,1}^T$ and $\overline{\Gamma}_{2,2} = \overline{\Gamma}_{2,2}^T$. Examination of (3.3.55) together with (3.3.32) and equating with (3.3.54) reveals:

$$\begin{aligned} K(t) &= P(t-1)\varphi(t)\left(\lambda(t)\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1} \\ &= \overline{\Gamma}_{2,1}(t)\overline{\Gamma}_{1,1}^{-1}(t) = \overline{\Gamma}_{2,1}(t)\overline{\Gamma}_{1,1}^{-T}(t) \\ &= R_{1,2}^T(t)R_{1,1}(t)R_{1,1}^{-1}(t)R_{1,1}^{-T}(t) \\ &= R_{1,2}^T(t)R_{1,1}^{-T}(t) \\ P(t) &= \frac{1}{\lambda(t)}\left(P(t-1) - P(t-1)\varphi(t)\left(\lambda(t)\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1)\right) \\ &= \frac{1}{\lambda(t)}\left(\overline{\Gamma}_{2,2}(t) - \overline{\Gamma}_{2,1}(t)\overline{\Gamma}_{1,1}^{-1}(t)\overline{\Gamma}_{1,2}(t)\right) = \frac{1}{\lambda(t)}\left(\overline{\Gamma}_{2,2}(t) - \overline{\Gamma}_{2,1}(t)\overline{\Gamma}_{1,1}^{-T}(t)\overline{\Gamma}_{1,2}(t)\right) \\ &= \frac{1}{\lambda(t)}\left(R_{1,2}^T(t)R_{1,2}(t) + R_{2,2}^T(t)R_{2,2}(t) - R_{1,2}^T(t)R_{1,1}(t)R_{1,1}^{-1}(t)R_{1,1}^{-T}(t)R_{1,1}^T(t)R_{1,2}(t)\right) \\ &= \frac{1}{\lambda(t)}\left(R_{2,2}^T(t)R_{2,2}(t)\right). \end{aligned} \tag{3.3.56}$$

Given that $P(t) = \overline{P}(t)\overline{P}^T(t)$ we deduce that $\overline{P}(t) = R_{2,2}^T(t)\big/\sqrt{\lambda(t)}$. The iterative update process may thus be stated as:

$$\begin{aligned} &\begin{bmatrix} R_{1,1}(t) & R_{1,2}(t) \\ 0 & R_{2,2}(t) \end{bmatrix} \overset{\text{QR}}{\leftarrow} \begin{bmatrix} \sqrt{\lambda(t)\Lambda} & 0 \\ \overline{P}^T(t-1)\varphi(t) & \overline{P}^T(t-1) \end{bmatrix} \\ &K(t) = R_{1,2}^T(t)R_{1,1}^{-T}(t) \\ &\boldsymbol{\theta}^*(t) = \boldsymbol{\theta}^*(t-1) + K(t)\left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right) \\ &\overline{P}(t) = \frac{R_{2,2}^T(t)}{\sqrt{\lambda(t)}}. \end{aligned} \tag{3.3.57}^{\#}$$

The most computationally expensive step used in this algorithm is the QR factorization. If the Housholder orthogonalization method (Golub and Van Loan, 1989) is used then a solution is found with $O(4(\dim(\boldsymbol{\theta}) + \dim(\boldsymbol{y}))^3/3)$ effort. There is still a matrix inversion to be performed in the calculation of $K(t)$, but the matrix to be inverted, R_{11}^T, is lower triangular. This permits a simple solution using backward substitution in $O(\dim(\boldsymbol{y})^2)$ effort. Numerical difficulties can be expected when the condition number of the matrix $R(t)$ approaches the inverse of the computational tolerance (usually very small.) In contrast, numerical problems are encountered when calculating $P(t)$ directly, if its condition number approaches the inverse of the *square root* of the computational tolerance.

If the system is not persistently excited the diagonal of $\overline{P}(t)$ will take on values approaching zero. This can be avoided by using conditional updating or by regularization which is discussed in the next section. Since $\overline{P}(t)$ is triangular its eigenvalues appear along its diagonal. A small positive valued diagonal matrix added to $\overline{P}(t)$ at each step thus forces the matrix to have positive eigenvalues preventing $\Gamma(t)$ from becoming rank deficient.

3.3.7 REGULARIZATION - THE CONSTANT TRACE AND KALMAN FILTER METHODS.

Another method of avoiding collinearity problems is to prevent $\left(\Phi^T(t)W_y(t)\Phi(t)\right)$ from becoming rank deficient. A term added to the cost function (3.3.18) that penalizes large movements in $\boldsymbol{\theta}$, thus stabilizing the parameter estimates, achieves this. Therefore let:

$$\begin{aligned}
\theta(t) &= \left[\boldsymbol{\theta}^T(1), \boldsymbol{\theta}^T(2), \cdots, \boldsymbol{\theta}^T(t)\right]^T \\
\widetilde{\theta}(t) &= \left[\widetilde{\boldsymbol{\theta}}^T(1), \widetilde{\boldsymbol{\theta}}^T(2), \cdots, \widetilde{\boldsymbol{\theta}}^T(t)\right]^T \\
E_\theta(t) &= \theta(t) - \widetilde{\theta}(t), \quad \varepsilon_\theta(t) = \boldsymbol{\theta}(t) - \widetilde{\boldsymbol{\theta}}(t) \\
W_\theta(t) &= \begin{bmatrix} \beta(t,1)\Lambda_\theta^{-1} & & \\ & \ddots & \\ & & \beta(t,t)\Lambda_\theta^{-1} \end{bmatrix}, \quad \Lambda_\theta^{-1} \in \Re^{rxr}
\end{aligned} \tag{3.3.58}$$

then:

$$J(\boldsymbol{\theta},t) = \frac{1}{2}E_y^T(t)W_y(t)E_y(t) + \frac{1}{2}E_\theta^T(t)W_\theta(t)E_\theta(t) \tag{3.3.59}$$

where $\Lambda_\theta^{-1}(t)$ is a matrix used to weight the relative importance of each parameter as well as the importance of parameter updates in relation to step length. Differentiating (3.3.59) directly with respect to $\boldsymbol{\theta}$ gives:

$$\frac{\partial J(\boldsymbol{\theta},t)}{\partial \boldsymbol{\theta}} = \left(\Phi(t)W_y(t)\Phi^T(t) + W_\theta(t)\right)\boldsymbol{\theta}(t) - \left(\Phi(t)W_y(t)Y(t) + W_\theta(t)\widetilde{\boldsymbol{\theta}}(t)\right). \tag{3.3.60}$$

Setting (3.3.60) to zero to determine the minimum and rearranging gives:

$$\boldsymbol{\theta}^*(t) = \left(\Phi(t)W_y(t)\Phi^T(t) + W_\theta(t)\right)^{-1}\left(\Phi(t)W_y(t)Y(t) + W_\theta(t)\widetilde{\boldsymbol{\theta}}(t)\right). \tag{3.3.61}$$

Comparing this to equation (3.3.22) it is clear that $W_\theta(t)$ can now be used to control the conditioning for the required inversion of $P^{-1}(t)$, however, regularization has also introduced errors in the parameter estimate in the form of the term $W_\theta(t), \widetilde{\boldsymbol{\theta}}(t)$, generally referred to as the bias error. The regularized constant trace algorithm heuristically makes use of these ideas. In this algorithm, during each iteration, the trace of $P(t)$ is kept constant by scaling and adding a small identity matrix. This causes the eigenvalues of $P(t)$ to remain artificially bounded. The algorithm takes the form:

$$\begin{aligned}
\boldsymbol{\theta}^*(t) &= \boldsymbol{\theta}^*(t-1) + K(t)\left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right) \\
K(t) &= P(t-1)\varphi(t)\left(\lambda\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1} \\
\overline{P}(t) &= \lambda^{-1}\left(P(t-1) - P(t-1)\varphi(t)\left(\Lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1)\right) \\
P(t) &= \left(\frac{\alpha_{\max} - \alpha_{\min}}{\mathrm{tr}\left(\overline{P}(t)\right)}\right)\overline{P}(t) + \alpha_{\min}I
\end{aligned} \tag{3.3.62}^{\#}$$

where $\alpha_{\max}$ and $\alpha_{\min}$ are the desired bounds for the maximum and minimum eigenvalues respectively. Typically these values are selected such that $\alpha_{\max}/\alpha_{\min} \approx 10^4$ and $\varphi^T\varphi(\alpha_{\max} - \alpha_{\min}) >> I$.

Unfortunately the optimal choice of $W_\theta(t)$ and $\widetilde{\boldsymbol{\theta}}(t)$ is unclear. There exists a large class of regularization algorithms, most of which differ primarily by the settings associated with these variables. One selection that is particularly revealing is to let $\widetilde{\boldsymbol{\theta}}(t) = \boldsymbol{\theta}(t-1)$. Then, from (3.3.58), it is clear that $\varepsilon_\theta(t) = \boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1))$, and the parameter update problem may be stated in the form of a state space system:

$$\begin{aligned} \boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1) + \varepsilon_\theta(t) \\ \boldsymbol{y}(t) &= \varphi^T(t)\boldsymbol{\theta}(t) + \varepsilon_y(t)\,. \end{aligned} \tag{3.3.63}$$

If it is assumed that $\varepsilon_\theta(t)$ and $\varepsilon_y(t)$ are uncorrelated white Gaussian process and measurement noise respectively, then applying the stochastic Kalman filter formulation to (3.3.63) gives:

$$\begin{aligned} &\boldsymbol{\theta}(t) = \boldsymbol{\theta}(t-1) + K(t)\left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right) \\ &K(t) = P(t-1)\varphi(t)\left(R_2(t) + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1} \\ &P(t) = P(t-1) - K(t)\varphi^T(t)P(t-1) + R_1(t) \\ &R_1(t) = \mathrm{E}\{\varepsilon_\theta(t)\varepsilon_\theta^T(t)\},\ R_2(t) = \mathrm{E}\{\varepsilon_y(t)\varepsilon_y^T(t)\} \end{aligned} \tag{3.3.64}$$

where $\mathrm{E}\{.\}$ denotes the mathematical expectation. Comparing (3.3.64) to (3.3.62) we see that $R_1(t)$ plays a role similar to $\alpha_{\min} I$ while $R_2(t) = \lambda(t)\Lambda$. From this we may deduce that, if $\lambda(t) = 1$ and the output is weighted by the inverse of its variance, then $\boldsymbol{\theta}(t)$ is Gaussian with a mean $\boldsymbol{\theta}^*(t)$ and $P(t)$ is the covariance matrix of the parameter estimation error. Thus we should set $\boldsymbol{\theta}^*(0)$ to be our best estimate of the parameter vector and $P(0)$ should reflect the confidence associated with this estimate. Kalman Filter theory also tells us that this algorithm, for a linear regression model, gives the optimal trade off between tracking ability and noise sensitivity in terms of a minimal *a-posteriori* parameter covariance matrix.

Although (3.3.64) provides insight into the choice of Λ^{-1} the selection of Λ_θ^{-1}, and hence W_θ, is unclear. This is a non-trivial problem which is dependent on the relationships between the parameters. These relationships are difficult to quantify in the case of ANN's and depend on the network structure as well as the underlying function generating the data mapping. In practice W_θ is typically 'tuned' by selecting a scaled version of the identity matrix. However, Hyötyniemi (1994, 1996) has provided some note-worthy results on how to define the organization of W_θ for dynamic systems having a specific regression structure using the theory of congruent systems.

3.3.8 The Recursive Singular Value Decomposition (SVD) Algorithm

As pointed out in previous sections, if the system input is not persistently exciting, RLS algorithms may perform poorly resulting in estimator windup. Furthermore RLS algorithms tend to be complex, prone to numerical ill conditioning and algorithm parameters may be difficult to select. In this section a new algorithm is developed. In it, an attempt is made to circumvent many of the problems already discussed.

We begin by reconsidering equation (3.3.59). In this equation there is a cost associated with the distance between some nominal $\widetilde{\boldsymbol{\theta}}(t)$ and the final solution $\boldsymbol{\theta}(t)$ weighted by the matrix $W_\theta(t)$. The conundrum that exists is that we cannot confidently specify $\widetilde{\boldsymbol{\theta}}(t)$ and $W_\theta(t)$ without knowing the ideal solution $\boldsymbol{\theta}^*(t)$. This may also be recognized in equation (3.3.61) by noting that, in essence, the choice of $\widetilde{\boldsymbol{\theta}}(t)$ and $W_\theta(t)$ *a-priori* is tantamount to determining a final bias error without knowing any information about the data being mapped. A different approach is clearly needed.

Rather than requiring a distance metric to be minimized let us assume that there exists some final, but initially unknown, error $\delta_\theta(t)$ between the attainable solution $\boldsymbol{\theta}(t)$ and the ideal solution $\boldsymbol{\theta}^*(t)$. Here the attainable solution is defined as that solution which can be achieved given the constructs of the underlying mapping and the quality of the excitation and measurements collected up to the current point in time. That is:

$$\delta_\theta(t) = \boldsymbol{\theta}^*(t) - \boldsymbol{\theta}(t).$$

The optimization should still however result in the ideal solution, therefore cost function (3.3.4) becomes:

$$\begin{aligned} J(\boldsymbol{\theta},t) &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\begin{pmatrix}\left(\boldsymbol{y}[k]-\varphi^T[k](\boldsymbol{\theta}[k]+\delta_\theta[k])\right)^T\Lambda^{-1}\\ \left(\boldsymbol{y}[k]-\varphi^T[k](\boldsymbol{\theta}[k]+\delta_\theta[k])\right)\end{pmatrix} \\ &= \frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\begin{pmatrix}\boldsymbol{y}[k]^T\Lambda^{-1}\boldsymbol{y}[k]\\ -2(\boldsymbol{\theta}[k]+\delta_\theta[k])^T\varphi[k]\Lambda^{-1}\boldsymbol{y}[k]\\ +(\boldsymbol{\theta}[k]+\delta_\theta[k])^T\varphi[k]\Lambda^{-1}\varphi^T[k](\boldsymbol{\theta}[k]+\delta_\theta[k])\end{pmatrix}. \end{aligned} \tag{3.3.65}$$

As before, equation (3.3.65) may be differentiated directly with respect to $\boldsymbol{\theta}$:

$$\frac{\partial J(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}} = \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\varphi^T[k](\boldsymbol{\theta}[k]+\delta_\theta[k]) - \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\boldsymbol{y}[k]). \tag{3.3.66}$$

Setting (3.3.66) equal to zero now gives:

$$\begin{aligned} &\frac{\partial J(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}} = P^{-1}(t)(\boldsymbol{\theta}(t)+\delta_\theta(t)) - \widetilde{Y}(t) = 0 \\ &\therefore \boldsymbol{\theta}^*(t) = (\boldsymbol{\theta}(t)+\delta_\theta(t)) = P(t)\widetilde{Y}(t), \quad (\boldsymbol{\theta}+\delta_\theta)\in\Re^r \end{aligned} \tag{3.3.67}$$

where, as before:

$$\begin{aligned} P^{-1}(t) &= \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\varphi^T[k] \\ \widetilde{Y}(t) &= \sum_{k=1}^{t}\beta(t,k)\varphi[k]\Lambda^{-1}\boldsymbol{y}[k]. \end{aligned} \tag{3.3.68}$$

Assume (all assumptions shall be addressed in more detail later) that $P^{-1}(t)$ can be factored into two components, one of which is approximately orthogonal to the regressor:

$$P^{-1}(t) = \widetilde{P}^{-1}(t) + \overline{P}^{-1}(t), \quad \widetilde{P}^{-1}(t)\varphi(t) \approx 0. \tag{3.3.69}$$

Further assume that the inverse can be expressed as the sum of the inverses of the previous factors:

$$P(t) = \widetilde{P}(t) + \overline{P}(t). \tag{3.3.70}$$

Combining equations (3.3.67) and (3.3.70) results in:

$$\boldsymbol{\theta}(t) + \delta_\theta(t) = \widetilde{P}(t)\widetilde{Y}(t) + \overline{P}(t)\widetilde{Y}(t)\,. \tag{3.3.71}$$

By definition $\widetilde{P}(t)$ is orthogonal to the regressor, but the regressor is not orthogonal to the parameter vector because $\hat{\boldsymbol{y}}(t) = \varphi^T(t)\boldsymbol{\theta}(t-1)$. Therefore this variable cannot be correlated with the parameter vector and we may conclude that:

$$\delta_\theta(t) = \widetilde{P}(t)\widetilde{Y}(t). \tag{3.3.72}$$

and:

$$\boldsymbol{\theta}(t) = \overline{P}(t)\widetilde{Y}(t)\,. \tag{3.3.73}$$

Proceeding as before we now define the *matrix* variable $\lambda(t)$ as follows:

$$\lambda(t) = \begin{cases} \beta(t,k)\beta(t-1,k)^{-1} & 0 < k < t \\ \beta(t-1,k)^{-1} & k = t \end{cases}.$$

Manipulation directly analogous to (3.3.24) therefore results in:

$$\begin{aligned} P^{-1}(t) &= \lambda(t)P^{-1}(t-1) + \varphi(t)\Lambda^{-1}\varphi^T(t) \\ \widetilde{Y}(t) &= \lambda(t)\widetilde{Y}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\,. \end{aligned} \tag{3.3.74}$$

If for any chosen constant scalar $\left(0 < \overline{\lambda} \le 1\right)$, there exist a $\beta(t,k)$ such that:

$$\lambda(t)P^{-1}(t-1) = \widetilde{P}^{-1}(t-1) + \overline{\lambda P}^{-1}(t-1) \tag{3.3.75}$$

then substituting (3.3.75) into (3.3.74) gives:

$$P^{-1}(t) = \widetilde{P}^{-1}(t-1) + \overline{\lambda P}^{-1}(t-1) + \varphi(t)\Lambda^{-1}\varphi^T(t)\,. \tag{3.3.76}$$

Substituting (3.3.74) into (3.3.73) gives:

$$\boldsymbol{\theta}(t) = \overline{P}(t)\left(\lambda(t)\widetilde{Y}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\right).$$

Shifting the time variable in (3.3.73) by one sample and rearranging gives $\widetilde{Y}(t-1) = \overline{P}^{-1}(t-1)\boldsymbol{\theta}(t-1)$. This can be substituted into the equation above to yield:

$$\boldsymbol{\theta}(t) = \overline{P}(t)\left(\lambda(t)\overline{P}^{-1}(t-1)\boldsymbol{\theta}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\right). \tag{3.3.77}$$

Consider now the factoring of $P(t)$ in assumptions (3.3.69) and (3.3.70). The matrix $P^{-1}(t)$ may be expanded using the Singular Value Decomposition (SVD) (Golub and Van Loan, 1989) to give:

$$P^{-1}(t) = U(t)\Sigma(t)V^T(t) \tag{3.3.78}$$

where $\Sigma(t)$ is the diagonal matrix of singular values $\sigma_i(t)$, and both $U(t)$ and $V(t)$ are orthonormal matrices. This decomposition exists even for non-square or singular systems. The orthonormal matrices have an additional, but very desirable, property. The columns of $U(t)$ which correspond to the larger singular value elements in $\Sigma(t)$, which we shall define as $\overline{U}(t)$ and $\overline{\Sigma}(t)$ respectively, form an orthonormal set of basis vectors for the range of $P^{-1}(t)$. Those columns of $V(t)$ whose corresponding elements in $\Sigma(t)$ are approximately zero, defined as $\widetilde{V}(t)$ and $\widetilde{\Sigma}(t)$ respectively, form an orthonormal set of basis vectors for the null space of $P^{-1}(t)$.

Furthermore, the smallest singular value is the 2-norm distance from the expanded matrix to the set of all rank deficient matrices. Thus, given some small tolerance $\delta > 0$, the numerical rank of $P^{-1}(t)$ is estimated to be $\hat{r}$ if:

$$\sigma_1 \geq \cdots \geq \sigma_{\hat{r}} \geq \delta \geq \sigma_{\hat{r}+1} \geq \cdots \geq \sigma_r \geq 0 . \tag{3.3.79}$$

Additionally if $U = [u_1, \cdots, u_n]$ and $V = [v_1, \cdots, v_n]$ are column partitionings then the least squares solution to (3.3.73) is given by:

$$\boldsymbol{\theta}(t) = \sum_{i=1}^{\hat{r}} \frac{u_i^T(t)\widetilde{Y}(t)}{\sigma_i(t)} v_i(t)$$

and the 2-norm of the residual is given by:

$$\rho_{LS}^2(t) = \left\| P^{-1}(t)\boldsymbol{\theta}(t) - \widetilde{Y}(t) \right\|_2^2 = \sum_{i=\hat{r}+1}^{r} \left(u_i^T(t)\widetilde{Y}(t) \right)^2 . \tag{3.3.80}$$

If $\sigma_{\hat{r}+1} \approx 0$, i.e. $P^{-1}(t)$ is poorly conditioned (or singular in the idealised theoretical case when $\sigma_{\hat{r}+1} = 0$), then any solution to a non-homogenous vector equation involving $P^{-1}(t)$ consists of that solution which exists in the range space of $P^{-1}(t)$ plus any linear combination of a solution in the null space of $P^{-1}(t)$. Given the prequel let us factor $P^{-1}(t)$ as follows:

$$\begin{aligned}
P^{-1}(t) &= U(t) \left[\begin{array}{ccc:ccc} \sigma_1 & & & & & \\ & \ddots & & & 0 & \\ & & \sigma_{\hat{r}} & & & \\ \hdashline & & & \sigma_{\hat{r}+1} & & \\ & 0 & & & \ddots & \\ & & & & & \sigma_r \end{array}\right] V^T(t) \\
&= U(t) \left[\begin{array}{c:ccc} 0 & 0 & & \\ \hdashline 0 & \sigma_{\hat{r}+1} & & \\ & & \ddots & \\ & & & \sigma_r \end{array}\right] V^T(t) + U(t) \left[\begin{array}{ccc:c} \sigma_1 & & & 0 \\ & \ddots & & \\ & & \sigma_{\hat{r}} & \\ \hdashline 0 & & & 0 \end{array}\right] V^T(t) \\
&= U(t)\widetilde{\Sigma}_0(t)V^T(t) + U(t)\overline{\Sigma}_0(t)V^T(t) \\
&= U(t) \left[\begin{array}{c:ccc} 0 & 0 & & \\ \hdashline 0 & \sigma_{\hat{r}+1} & & \\ & & \ddots & \\ & & & \sigma_r \end{array}\right] \left[0 \,\vdots\, \widetilde{V}(t) \right]^T + \left[\overline{U}(t) \,\vdots\, 0 \right] \left[\begin{array}{ccc:c} \sigma_1 & & & 0 \\ & \ddots & & \\ & & \sigma_{\hat{r}} & \\ \hdashline 0 & & & 0 \end{array}\right] V^T(t) \\
&= U(t)\widetilde{\Sigma}_0(t) \left[0 \,\vdots\, \widetilde{V}(t) \right]^T + \left[\overline{U}(t) \,\vdots\, 0 \right] \overline{\Sigma}_0(t)V^T(t) .
\end{aligned} \tag{3.3.81}$$

Now, because $\widetilde{V}(t)$ is the orthonormal basis of the null space of $P^{-1}(t)$, the first expanded term in the right hand side of the above equation, when multiplied by $\varphi(t)$, corresponds to the homogenous equation $P^{-1}(t)\varphi(t) = 0$. Defining $\widetilde{U}(t)$ and $\overline{V}(t)$ as those sub-matrices of $U(t)$ and $V(t)$, which have the same columns as $\widetilde{V}(t)$ and $\overline{U}(t)$ respectively, equation (3.3.81) may be simplified still further:

$$
\begin{aligned}
P^{-1}(t) &= \widetilde{U}(t)\begin{bmatrix} \sigma_{\hat{r}+1} & & 0 \\ & \ddots & \\ 0 & & \sigma_r \end{bmatrix}\widetilde{V}(t)^T + \overline{U}(t)\begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_{\hat{r}} \end{bmatrix}\overline{V}(t)^T \\
&= \widetilde{U}(t)\widetilde{\Sigma}(t)\widetilde{V}^T(t) + \overline{U}(t)\overline{\Sigma}(t)\overline{V}^T(t) \\
&= \widetilde{P}^{-1}(t) + \overline{P}^{-1}(t)\,.
\end{aligned}
\tag{3.3.82}
$$

Clearly then, $\widetilde{P}^{-1}(t)\varphi(t) \approx 0$, and the requirements of assumption (3.3.69) have been satisfied. Note there is a subtle but important difference between the implicit definitions of $\widetilde{\Sigma}_0, \overline{\Sigma}_0$ and $\widetilde{\Sigma}, \overline{\Sigma}$ in equations (3.3.81) and (3.3.82) respectively.

As the matrices $U(t)$ and $V(t)$ are orthogonal i.e. $U^T(t) = U^{-1}(t), \quad V^T(t) = V^{-1}(t)$ we may write $P(t)$ as:

$$
P(t) = V(t)\left[\begin{array}{ccc:ccc} 1/\sigma_1 & & & & 0 & \\ & \ddots & & & & \\ & & 1/\sigma_{\hat{r}} & & & \\ \hdashline & & & 1/\sigma_{\hat{r}+1} & & \\ & 0 & & & \ddots & \\ & & & & & 1/\sigma_r \end{array}\right] U^T(t)\,.
$$

Now, to verify assumption (3.3.70), a trivial analysis identical to (3.3.81) and (3.3.82) can be performed:

$$
\begin{aligned}
P(t) &= \widetilde{V}(t)\begin{bmatrix} 1/\sigma_{\hat{r}+1} & & 0 \\ & \ddots & \\ 0 & & 1/\sigma_r \end{bmatrix}\widetilde{U}^T(t) + \overline{V}(t)\begin{bmatrix} 1/\sigma_1 & & 0 \\ & \ddots & \\ 0 & & 1/\sigma_{\hat{r}} \end{bmatrix}\overline{U}^T(t) \\
&= \widetilde{V}(t)\widetilde{\Sigma}^{-1}(t)\widetilde{U}^T(t) + \overline{V}(t)\overline{\Sigma}^{-1}(t)\overline{U}^T(t) \\
&= \widetilde{P}(t) + \overline{P}(t)\,.
\end{aligned}
\tag{3.3.83}
$$

Revisiting equations (3.3.69) and (3.3.75) we see that:

$$
\begin{aligned}
\lambda(t) &= \left(P^{-1}(t-1) + \overline{\lambda}\overline{P}^{-1}(t-1) - \overline{P}^{-1}(t-1)\right)P(t-1) \\
&= I_{r\times r} + (\overline{\lambda} - I_{r\times r})\overline{P}^{-1}(t-1)P(t-1)\,.
\end{aligned}
\tag{3.3.84}
$$

This may be substituted back into equation (3.3.74) to give:

$$
P^{-1}(t) = P^{-1}(t-1) + (\overline{\lambda} - I)\overline{P}^{-1}(t-1) + \varphi(t)\Lambda^{-1}\varphi^T(t)\,. \tag{3.3.85}
$$

Using the decomposition (3.3.78), recalling that $U^T(t) = U^{-1}(t), V^T(t) = V^{-1}(t)$, and therefore that $U(t-1)U^T(t-1) = I_{r\times r}$, equation (3.3.84) may be restated as:

$$
\begin{aligned}
\lambda(t) &= U(t-1)U^T(t-1) + (\overline{\lambda} - I_{r\times r})U(t-1)\overline{\Sigma}_0(t-1)V^T(t-1)V(t-1)\Sigma^{-1}(t-1)U^T(t-1) \\
&= U(t-1)\Big(I + (\overline{\lambda} - I)\overline{\Sigma}_0(t-1)\Sigma^{-1}(t-1)\Big)U^T(t-1) \\
&= U(t-1)\left(I + (\overline{\lambda} - I)\begin{bmatrix}\overline{\Sigma}(t-1) & 0\\ 0 & 0\end{bmatrix}\Sigma^{-1}(t-1)\right)U^T(t-1) \\
&= U(t-1)\left(I + (\overline{\lambda} - I)\begin{bmatrix}I_{\hat{r}\times\hat{r}} & 0\\ 0 & 0\end{bmatrix}\right)U^T(t-1) \\
&= U(t-1)\begin{bmatrix}\overline{\lambda}I_{\hat{r}\times\hat{r}} & 0\\ 0 & I_{(r-\hat{r})\times(r-\hat{r})}\end{bmatrix}U^T(t-1).
\end{aligned}
\tag{3.3.86}
$$

Now substituting (3.3.86) into (3.3.77), and using the SVD expansion for $\overline{P}^{-1}(t-1)$ yields:

$$
\begin{aligned}
\boldsymbol{\theta}(t) &= \overline{P}(t)\left(\begin{matrix}U(t-1)\begin{bmatrix}\overline{\lambda}I_{\hat{r}\times\hat{r}} & 0\\ 0 & I_{(r-\hat{r})\times(r-\hat{r})}\end{bmatrix}U^T(t-1)U(t-1)\overline{\Sigma}_0(t-1)V^T(t-1)\boldsymbol{\theta}(t-1)\\ +\varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\end{matrix}\right) \\
&= \overline{P}(t)\Big(U(t-1)\overline{\lambda}\overline{\Sigma}_0(t-1)V^T(t-1)\boldsymbol{\theta}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\Big) \\
&= \overline{P}(t)\Big(\overline{\lambda}\overline{P}^{-1}(t-1)\boldsymbol{\theta}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\Big).
\end{aligned}
\tag{3.3.87}
$$

Rearranging (3.3.76) yields $\overline{P}^{-1}(t-1) = \overline{\lambda}^{-1}\Big(P^{-1}(t) - \widetilde{P}^{-1}(t-1) - \varphi(t)\Lambda^{-1}\varphi^T(t)\Big)$, which substituted into the previous equation gives:

$$
\boldsymbol{\theta}(t) = \overline{P}(t)\Big(\Big(P^{-1}(t) - \widetilde{P}^{-1}(t-1) - \varphi(t)\Lambda^{-1}\varphi^T(t)\Big)\boldsymbol{\theta}(t-1) + \varphi(t)\Lambda^{-1}\boldsymbol{y}(t)\Big).
$$

Finally, noting that $\hat{\boldsymbol{y}}(t) = \varphi^T(t)\boldsymbol{\theta}(t-1)$, this equation can be combined with equation (3.3.85) to give the recursive update algorithm:

$$
\begin{aligned}
P^{-1}(t) &= P^{-1}(t-1) + (\overline{\lambda} - I)\overline{P}^{-1}(t-1) + \varphi(t)\Lambda^{-1}\varphi^T(t) \\
\boldsymbol{\theta}(t) &= \overline{P}(t)\Big(P^{-1}(t) - \widetilde{P}^{-1}(t-1)\Big)\boldsymbol{\theta}(t-1) + \overline{P}(t)\varphi(t)\Lambda^{-1}\big(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\big)
\end{aligned}
\tag{3.3.88}
$$

where $\overline{P}^{-1}(t)$, $\widetilde{P}^{-1}(t)$ and $\overline{P}(t)$ are calculated using the SVD expansions in equations (3.3.82) and (3.3.83) respectively.

Algorithm (3.3.88) has some interesting properties. Note that if $\delta = 0$ in (3.3.79) then $\widetilde{P}^{-1}(t)$ would be an empty matrix and $\overline{P}^{-1}(t) = P^{-1}(t)$. This causes (3.3.88) to reduce to:

$$
\begin{aligned}
P^{-1}(t) &= \overline{\lambda}P^{-1}(t-1) + \varphi(t)\Lambda^{-1}\varphi^T(t) \\
\boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1) + P(t)\varphi(t)\Lambda^{-1}\big(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\big)
\end{aligned}
$$

which is the RLS algorithm in (3.3.24) and (3.3.28) before the application of the matrix inversion lemma. All the previous algorithms may thus be viewed as special cases of (3.3.88). The key difference is that now the selection of a single algorithm parameter, δ, allows us to exclude irrelevant information in the construction of the model parameters. Ever increasing values of δ will result in ever increasing amounts of potential information being

excluded from the model. This stems from the requirement that the orthogonality condition in (3.3.69) is only approximate, as opposed to absolute. The orthogonality condition formalises the fact that information represented by elements of $P^{-1}(t)$ which do not correlate with any model parameters can play no role in the final model. But the SVD decomposition has allowed us to separate the measured input information into mutually uncorrelated components through the orthonormal matrices $U(t)$ and $V(t)$. The singular values thus represent the amount of information along each direction of the subspaces spanned by $U(t)$ and $V(t)$. Interestingly, this is exactly what unsupervised neural networks using a method called principle component analysis (PCA) achieve, and this approach may provide an elegant mechanism to unify the analysis of unsupervised and feed forward neural networks.

By increasing the value of δ one successively increases the amount of information that is considered part of the null space of $P^{-1}(t)$. Therefore, by neglecting small or approximately orthogonal terms, those parameters which play little to no role in the underlying mapping will be ignored and the tendency for a model with excess parameters to over fit the data will be regulated. Furthermore, numerical inaccuracies and rounding errors which would generate small non-zero quantities, even if the two spaces are theoretically orthogonal, are automatically excluded. This results in an algorithm which requires no inversion of poorly conditioned matrices as the information (or lack thereof) leading to this condition is removed before the inversion resulting in $\overline{P}^{-1}(t)$. Regularization is therefore not required and the parameter estimate is bias free. The selection of δ is clearly chosen based on the permissible size of the residual as evidenced by equation (3.3.80).

Let us now consider the stability of the algorithm. The eigenvalues of the matrix term $\lambda(t)$ in equation (3.3.74) determines the stability of the equation for $P^{-1}(t)$. However, because $U^T(t-1) = U^{-1}(t-1)$, the result expressed in (3.3.86) is indeed an eigenvalue decomposition of $\lambda(t)$ with the eigenvectors being contained in $U(t-1)$ and all the eigenvalues being either $\overline{\lambda}$ for elements used in the model parameter updates, or 1 for elements associated with the residual. As $0 < \overline{\lambda} \leq 1$ we may conclude that equation (3.3.74) is bounded stable but contains elements which will not decay.

The stability of the parameter update equation is a little more complex. Considering equation (3.3.87) it is clear that the stability of the parameter update is determined by the eigenvalues of the term $\overline{P}(t)\overline{\lambda P}^{-1}(t-1)$. If one assumes that the data samples contain no new information then $\overline{P}^{-1}(t) = \overline{P}^{-1}(t-1)$ and the eigenvalues are just $\overline{\lambda}$ meaning that the parameter vector will decay toward zero under these conditions. For this reason updating should only occur if the prediction does not sufficiently closely represent the actual system output or if it is determined, using a technique such as conditional updating, that the input data is sufficiently rich in information.

Unfortunately the algorithm is very expensive to compute and requires more storage than the other methods mentioned. The computation of the SVD using the Golub-Reinsch SVD algorithm (Golub and Van Loan, 1989) requires $O(21\dim(\boldsymbol{\theta})^3)$ effort for the form shown in (3.3.88), however, this may be reduced to a limited extent.

Because $P^{-1}(t)$ is a symmetric positive definite matrix the orthonormal matrices $U(t)$ and $V(t)$, also called the left and right SVD factors respectively, are equal. This means, for each iteration, that only one SVD factor needs to be determined. This reduces the computational effort, if a new SVD is calculated during each iteration using the Golub-Reinsch SVD algorithm, to $O(12\dim(\boldsymbol{\theta})^3)$. Therefore setting $U(t)=V(t)$ and substituting the resulting decomposition into (3.3.88) results in the following algorithm:

$$\begin{aligned} U(t),\Sigma(t) &\overset{\text{SVD}}{\leftarrow} U(t-1)\begin{bmatrix}\overline{\lambda}\overline{\Sigma}(t-1) & 0\\ 0 & \widetilde{\Sigma}(t-1)\end{bmatrix}U^T(t-1)+\varphi(t)\Lambda^{-1}\varphi^T(t)\\ \boldsymbol{\theta}(t) &= \overline{U}(t)\Big(\overline{U}^T(t)-\overline{\Sigma}^{-1}(t)\overline{U}^T(t)\widetilde{U}(t-1)\widetilde{\Sigma}(t-1)\widetilde{U}^T(t-1)\Big)\boldsymbol{\theta}(t-1)\\ &\quad+\overline{U}(t)\overline{\Sigma}^{-1}(t)\overline{U}^T(t)\varphi(t)\Lambda^{-1}\big(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\big)\end{aligned} \qquad (3.3.89)^{\#}$$

Where $\overline{U},\widetilde{U},\overline{\Sigma},\widetilde{\Sigma},\overline{\Sigma}^{-1}$ are formed as shown in equations (3.3.81),(3.3.82) and (3.3.83). Note that in this form of the algorithm it is not necessary to explicitly form $P^{-1}(t)$, only $U(t)$ and the diagonal of $\Sigma(t)$ need be calculated and stored from iteration to iteration. This results in significant memory savings over the form shown in (3.3.88).

Another advantage of using only one SVD factor is that the symmetric nature of the matrices is guaranteed and numerical rounding errors cannot result in a non-symmetric $P^{-1}(t)$. Unfortunately the product $\varphi(t)\Lambda^{-1}\varphi^T(t)$ is still calculated which, as was evidenced in the square root algorithm, is undesirable from a numerical standpoint.

Future work might investigate how SVD rank-1 iterative updates and / or 'square root' type algorithms, such as the one described by Zhang (Zhang *et al.*, 1994) could be applied to (3.3.89). The application of other closely related and highly efficient algorithms such as Biermans U-D factorization method (Bierman, 1977), (Ljung, 1983), (Zhang and Li, 1999) to algorithm (3.3.89) should also be investigated. Also, the SVD approach should facilitate the use of a robust subset selection algorithm to reduce the required model order, which may be seen as a form of network structure optimisation.

3.3.9 Implementation of the Learning Rule

Implementation of the learning rule in this work is, for the most part, the straight forward application of the 'hash' equations in the preceding sections. However, there are a few customised details such as the use of dead zones to turn adaption on or off, network specific formulation of the problem for reduced computational effort, and the calculation of $\varphi(t)$ in equations (3.3.9), (3.3.10) and (3.3.13).

3.3.9.1 Adaption with a Dead Zone

All the learning algorithms discussed have, to greater or lesser extents, detrimental characteristics associated with continual updating when the incoming data contains little or no useful information. In section 3.3.6.3 the use of conditional updating attempted to detect when this was the case and turn off adaption if there was insufficient excitation. This problem may also be addressed by a more heuristic, but simpler and globally applicable, approach by simply turning off the adaption if the prediction error $\left|\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\right|$ is sufficiently small.

This technique is generally referred to as the dead zone technique. In this work a dead zone is applied to three different types of errors, requiring four different parameters to be set by the user:

1. Relative error – Error expressed as a percentage of the absolute value of the current target output. That is $e_r = |\boldsymbol{y}(t)|\varepsilon_r /100$ where ε_r is a value supplied by the user in percentage units.

2. Absolute error – A threshold value, which if the prediction error is below, is considered acceptable. That is $e_a = \varepsilon_a /100$ where ε_a is a value supplied by the user. This is the allowable error value when $|\boldsymbol{y}(t)| \approx 0$ and the relative error tends to zero.

3. Differential error – This error is defined as: $e_d = \min\left(\left|\frac{\varepsilon_{dr}}{100}\left(\boldsymbol{y}(t) - \boldsymbol{y}(t-1)\right)\right|, \frac{\varepsilon_{da}\Delta t}{100}\right)$ where the user supplied ε_{dr} and ε_{da} are the percentage relative differential error and absolute differential error values respectively, and Δt is the sampling interval. By using a first order approximation of the differential it is easily seen that the first value contained in the minimum function of e_d represents the absolute error that results in the predicted output at the next time step if the relative slope of the predicted output is in error by ε_{dr} percentage points of the actual slope. This first term thus gives an indication of the relative differential of the prediction error. Contrary to the absolute error above, we see that the absolute differential error term only plays a role as $|\boldsymbol{y}(t) - \boldsymbol{y}(t-1)|$ becomes larger, i.e. the differential of the output with respect to time is large. As this differential increases, a very small percentage error in the slope can generate very large output errors. The absolute differential error term thus places what is essentially an upper bound on the output error due to small differential errors at high rates of change in the output signal.

The three terms above can be combined into a single condition which, if true, permits the updating at any specific time to be avoided. Using the least stringent indicator of the three to indicate the cessation of adaption yields the following dead zone criteria:

$$|\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)| < \max\left(|\boldsymbol{y}(t)|\varepsilon_r /100, \varepsilon_a /100, \min\left(\left|\frac{\varepsilon_{dr}}{100}\left(\boldsymbol{y}(t) - \boldsymbol{y}(t-1)\right)\right|, \frac{\varepsilon_{da}\Delta t}{100}\right)\right). \tag{3.3.90}$$

A limited amount of hysteresis can also be applied to (3.3.90) to prevent rapid activation and deactivation of the adaption algorithm. If we express the hysteresis ε_{hy} as a percentage about the nominal equation above, then the 'turn off' condition becomes:

$$|\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)| < \left(1 - \frac{\varepsilon_{hy}}{100}\right)\max\left(|\boldsymbol{y}(t)|\varepsilon_r /100, \varepsilon_a /100, \min\left(\left|\frac{\varepsilon_{dr}}{100}\left(\boldsymbol{y}(t) - \boldsymbol{y}(t-1)\right)\right|, \frac{\varepsilon_{da}\Delta t}{100}\right)\right) \tag{3.3.91}$$

and the 'turn on' condition is:

$$|\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)| > \left(1 + \frac{\varepsilon_{hy}}{100}\right)\max\left(|\boldsymbol{y}(t)|\varepsilon_r /100, \varepsilon_a /100, \min\left(\left|\frac{\varepsilon_{dr}}{100}\left(\boldsymbol{y}(t) - \boldsymbol{y}(t-1)\right)\right|, \frac{\varepsilon_{da}\Delta t}{100}\right)\right). \tag{3.3.92}$$

3.3.9.2 NETWORK SPECIFIC FORMULATIONS

Observation of (3.3.9) and (3.3.10) reveals that the regressor $\varphi(t)$ is constructed using the Kronecker tensor product, from q (the number of outputs) copies of the term $G(\boldsymbol{x}[k],\xi_{I_0})$ and $\left(G(\boldsymbol{x}[k],\xi_{I_0})\otimes[\boldsymbol{x}^T[k],1]^T\right)$ for RBF and LMN networks respectively. This yields a sparse matrix construction which, if computed by direct application of the presented equations, is extremely inefficient. It turns out, for these network types, that by transposing the expression for the regression equation the calculations are greatly simplified. That is, for RBF's we should use the original equations (3.2.11) and (3.2.12), and for LMN's we use (3.2.24) and (3.2.25). Effectively this makes the computation of $K(t)$ and $P(t)$ dependent on the model structure only and multiple output systems have identical regression structures associated with each output. If the outputs of the corresponding identified system are scaled such that their variances are normalized then the inverse weighting matrix, Λ, is simply the identity matrix. The regressor $\varphi(t)$, and hence $K(t)$ and $P(t)$, are now dependent on the model structure only and the matrix term $\left(\lambda(t)\Lambda+\varphi^T(t)P(t-1)\varphi(t)\right)$, in all but the SVD algorithm, reduces to a diagonal matrix of identical terms. Under these conditions the system is equivalent to a single output system where only the prediction error calculation in the parameter update portion of the equation need be modified to reflect the multiple output nature of the network. The inversion in the calculation of $K(t)$ may also be reduced to division by a scalar value. Therefore, for RBF and LMN networks, equations (3.3.15)#, (3.3.16)#, (3.3.33)# (3.3.57)# and (3.3.62)# are respectively implemented as shown below:

Gadient descent algorithm:

$$\boldsymbol{\theta}(t)=\boldsymbol{\theta}(t-1)+\gamma\left(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\right)\varphi^T(t)\,, \tag{3.3.93}^{\#}$$

gradient descent with momentum algorithm:

$$\begin{aligned}\boldsymbol{\theta}(t)&=\boldsymbol{\theta}(t-1)+\gamma\,\boldsymbol{P}(t)\\ \boldsymbol{P}(t)&=\left(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\right)\varphi^T(t)+\mu\,\boldsymbol{P}(t-1)\,,\end{aligned} \tag{3.3.94}^{\#}$$

exponential forgetting factor algorithm:

$$\begin{aligned}\boldsymbol{\theta}(t)&=\boldsymbol{\theta}(t-1)+\left(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\right)K^T(t)\\ K(t)&=\frac{P(t-1)\varphi(t)}{\lambda+\varphi^T(t)P(t-1)\varphi(t)}\\ P(t)&=\frac{1}{\lambda}\left(P(t-1)-K(t)\varphi^T(t)P(t-1)\right),\end{aligned} \tag{3.3.95}^{\#}$$

square root algorithm (note that $R_{1,1}(t)$ is now a scalar variable):

$$\begin{bmatrix}R_{1,1}(t) & R_{1,2}(t)\\ 0 & R_{2,2}(t)\end{bmatrix}\overset{\text{QR}}{\leftarrow}\begin{bmatrix}\sqrt{\lambda(t)\Lambda} & 0\\ \overline{P}^T(t-1)\varphi(t) & \overline{P}^T(t-1)\end{bmatrix}$$

$$K(t)=R_{1,2}^T(t)/R_{1,1}(t) \tag{3.3.96}^{\#}$$

$$\begin{aligned}\boldsymbol{\theta}(t)&=\boldsymbol{\theta}(t-1)+\left(\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\right)K^T(t)\\ \overline{P}(t)&=\frac{R_{2,2}^T(t)}{\sqrt{\lambda(t)}}\,,\end{aligned}$$

constant trace algorithm:

$$\begin{aligned}
\boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1) + \left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right)K^T(t) \\
K(t) &= \frac{P(t-1)\varphi(t)}{\lambda + \varphi^T(t)P(t-1)\varphi(t)} \\
\overline{P}(t) &= \frac{1}{\lambda}\left(P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{1+\varphi^T(t)P(t-1)\varphi(t)}\right) \\
P(t) &= \left(\frac{\alpha_{\max} - \alpha_{\min}}{\mathrm{tr}\left(\overline{P}(t)\right)}\right)\overline{P}(t) + \alpha_{\min} I\ .
\end{aligned} \qquad (3.3.97)^{\#}$$

The parameter update equation for the SVD algorithm (3.3.89)# is also changed giving:

$$\begin{aligned}
U(t),\Sigma(t) &\overset{\mathrm{SVD}}{\leftarrow} U(t-1)\begin{bmatrix}\overline{\lambda}\overline{\Sigma}(t-1) & 0 \\ 0 & \widetilde{\Sigma}(t-1)\end{bmatrix}U^T(t-1) + \varphi(t)\Lambda^{-1}\varphi^T(t) \\
\boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1)\left(\overline{U}(t) - \widetilde{U}(t-1)\widetilde{\Sigma}(t-1)\widetilde{U}^T(t-1)\overline{U}(t)\overline{\Sigma}^{-1}(t)\right)\overline{U}^T(t) \\
&\quad + \left(\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right)\Lambda^{-1}\varphi^T(t)\overline{U}(t)\overline{\Sigma}^{-1}(t)\overline{U}^T(t)\ .
\end{aligned} \qquad (3.3.98)^{\#}$$

3.3.9.3 Calculation of the Regressor

Clearly, from equations (3.2.12) and (3.2.25) the regressor variables for RBF and LMN networks are simply $G(\boldsymbol{x}[k],\xi_{I_0})$ and $\left(G(\boldsymbol{x}[k],\xi_{I_0})\otimes[\boldsymbol{x}^T[k],1]^T\right)$ respectively. These variables are readily available during the activation rule computation and shall not be addressed further.

For MLP networks the regressor must be calculated by application of the famous back-propagation algorithm (Rumelhart and McClelland, 1986). The algorithm is essentially the repeated application of the chain rule to (3.2.5). Noting that $\widetilde{\boldsymbol{\theta}}^{(l)}$ in (3.3.13) is just the column form of $\Theta^{(l)}$ in (3.2.5), further assuming that the output activation function is not necessarily linear and finally explicitly showing the matrix form of the integration function, we can rewrite (3.2.5) in the following general functional form:

$$\hat{\boldsymbol{y}} = G^{(L)}\left(\mathrm{I}^{(L)}\left(\widetilde{\boldsymbol{\theta}}^{(L)}, G^{(L-1)}\left(\mathrm{I}^{(L-1)}\left(\widetilde{\boldsymbol{\theta}}^{(L-1)}, G^{(L-2)}\left(\mathrm{I}^{(L-2)}\left(\widetilde{\boldsymbol{\theta}}^{(L-2)}, \cdots G^{(2)}\left(\mathrm{I}^{(2)}\left(\widetilde{\boldsymbol{\theta}}^{(2)}, [\boldsymbol{x}^T,1]^T\right)\right)\cdots\right)\right)\right)\right)\right). \qquad (3.3.99)$$

Using matrix differentiation and repeated application of the chain rule to differentiate $\hat{\boldsymbol{y}}$ with respect to $\widetilde{\boldsymbol{\theta}}^{(l)}$ gives:

$$\frac{\partial\hat{\boldsymbol{y}}}{\partial\widetilde{\boldsymbol{\theta}}^{(l)}} = \frac{\partial\mathrm{I}^{(l)}}{\partial\widetilde{\boldsymbol{\theta}}^{(l)}}\frac{\partial G^{(l)}}{\partial\mathrm{I}^{(l)}}\frac{\partial\mathrm{I}^{(l+1)}}{\partial G^{(l)}}\frac{\partial G^{(l+1)}}{\partial\mathrm{I}^{(l+1)}}\cdots\frac{\partial\mathrm{I}^{(L-1)}}{\partial G^{(L-2)}}\frac{\partial G^{(L-1)}}{\partial\mathrm{I}^{(L-1)}}\frac{\partial\mathrm{I}^{(L)}}{\partial G^{(L-1)}}\frac{\partial G^{(L)}}{\partial\mathrm{I}^{(L)}}\frac{\partial\hat{\boldsymbol{y}}}{\partial G^{(L)}}\ . \qquad (3.3.100)$$

This represents precisely the sub matrix elements that make $\varphi^T(k,\boldsymbol{\theta})$ in equation (3.3.13). Careful examination of this equation reveals the following recursive relationship:

$$\frac{\partial\hat{\boldsymbol{y}}}{\partial G^{(l)}} = \frac{\partial\mathrm{I}^{(l+1)}}{\partial G^{(l)}}\frac{\partial G^{(l+1)}}{\partial\mathrm{I}^{(l+1)}}\frac{\partial\hat{\boldsymbol{y}}}{\partial G^{(l+1)}}\ . \qquad (3.3.101)$$

Therefore, if we solve (3.3.101) for the output layer, we can recursively solve it for any hidden layer. Setting $l+1=L$ then, by inspection, the last term reduces to $I_{q\times q}$. The middle term is the differential of the activation function with respect to the integration function. As there are no connections between units within the same layer this matrix must also be diagonal in form. Furthermore an expression, which we call $G'^{(l)}$, can be

```
00  dYdG=eye(numoutputs);
10  for i= net.numlayers:-1:2
20      dGdNet=diff_activ_fcn(net.activation{i}(1:end-1));
30      dYdNet=dYdG.*dGdNet(:,ones(1,size(dYdG,2));
40      dYdG=net.weights{i}(1:end-1,:)*dYdNet;       % Recursion
50      phi(indx(i,1):indx(i,2),:)=kron(dYdNet,net.activation{i-1});
60  end
```

Listing 3.3.1. Back Propagation Implementation.

obtained for each of the diagonal terms by analytically differentiating the activation function and then, as all the required information is available, evaluating it at each time point. The first term is the differential of the output layer's integration functions with respect to the last hidden layer's activation functions. Again, an analytic expression can be determined beforehand and evaluated during each iteration. The MLP networks in this work all use linear integration functions, the matrix form being $\mathrm{I}^{(l)} = \Theta^{(l)} G^{(l-1)}$, where $\Theta^{(l)}$ and $G^{(l-1)}$ are defined in (3.2.2) and (3.2.3). Clearly the differential of this function is simply the weight matrix $\Theta^{(l)}$. The last step to solving (3.3.100) is to recognize that the very first term in this equation is simply a reshaped form of $G^{(l-1)}$. Combining these facts and performing some mechanical algebraic manipulation results in the following final expressions:

$$\begin{aligned} &\frac{\partial \hat{\boldsymbol{y}}}{\partial G^{(l)}} = \Theta^{(l+1)} G'^{(l+1)} \frac{\partial \hat{\boldsymbol{y}}}{\partial G^{(l+1)}}, \quad \frac{\partial \hat{\boldsymbol{y}}}{\partial G^{(L)}} = I_{q\times q}, \quad 1 < l < L \\ &\frac{\partial \hat{\boldsymbol{y}}}{\partial \tilde{\boldsymbol{\theta}}^{(l)}} = G'^{(l)} \frac{\partial \hat{\boldsymbol{y}}}{\partial G^{(l)}} \otimes G^{(l-1)}, \quad G^{(1)} = [\boldsymbol{x}^T, 1]^T, \quad 1 < l < L . \end{aligned} \tag{3.3.102}$$

Listing 3.3.1 shows the implementation of (3.3.102). A few points to note are:

- The variable `dGdNet` (line 20) is equivalent to $G'^{(l)}$ except that differential with respect to the bias term is ignored by removing the last row of the activation function vector. The variable is also stored as a vector.
- The variable `dYdNet` is equivalent to $G'^{(l+1)} \partial \hat{\boldsymbol{y}} / \partial G^{(l+1)}$ and is reused in the recursion relationship (line 40) and the calculation of $\varphi(t)$ in line 50. This variable is also more efficiently calculated by performing an element wise multiplication (line 30) instead of a diagonal matrix multiplication.
- The variable `dYdG` is equivalent to $\partial \hat{\boldsymbol{y}} / \partial G^{(l)}$ and is preset to the appropriately sized identity matrix before the loop begins. As for `dGdNet` the differential with respect to the bias term is ignored by removing the last row of weight matrix.
- The starting and ending row indexes of `phi=` $\varphi(t)$ for each layer are precalculated and stored in `indx` to be reused during each iteration calculation in line 50.

3.3.9.4 GLOBAL VS. LOCAL LEARNING

In MLP networks the application of the learning rule is global in nature. That is, all adjustable parameters (the vector θ) in the network are updated simultaneously. This leads to large values of r implying significant computational effort, which scales according to the third power of r, at each iteration. In LMN networks, the implementation presented here updates only those local models that significantly contribute to the output. This can drastically reduce the computational effort associated with the update computation due to much smaller values of r associated with each local model and where now, increasing the number of contributing local models causes the computational effort to scale up linearly. Obviously, it is possible to globally train the LMN network as well, but local training of only subsections of the MLP does not appear to be possible. The global vs. local training trade off in LMN's has been addressed by Murray-Smith and Johansen (1997).

The RBF network implementation fits in between the MLP and LMN approach. Here the matrices used in the update equations are defined in a global sense but sub matrices are extracted and used based on the basis functions that remain activated once the truncation region is applied. This may be viewed as multiple *overlapping* local models, where the size of the local model and the degree of overlap is determined by the radius of the truncation region. This overlapping nature has implications for the update equations as sub elements of the matrices involved are updated at different times. The effects of this are unclear and may be particularly problematic to SVD based approaches. To this author's knowledge these effects have not been quantified.

3.3.10 STRUCTURE OPTIMISATION

The problem of structural optimisation is a complex one and a detailed exposition will not be attempted in this work. This section serves only as an introduction to the problem, recognizing that the problem is an important and significant issue, and to provide the reader with some reference and pointers of other work that has addressed this issue in more detail.

The goal of structure optimisation is to define a mechanism whereby the density of units and/or complexity of local models is minimized in a problem adaptive way. In section 3.3.1 a cost function was defined (Eq. (3.3.1)) which was minimized when the optimal set of parameters was obtained. Implicit in this cost function was a fixed model structure. The optimisation problem could be more generally stated as:

$$J^*(\theta, M) = \min_{\theta, M}\left(J(t, \theta, M)\right)$$

where M represents the network parameters that determine its structure. The optimisation of the network structure M is unfortunately a difficult non-convex problem and typically involves constructive (increasing the number of units as the systems trains) and/or destructive (removal of units) evolution of network components. This must be accomplished while maintaining network robustness and generalisation while using only the currently available training data. To guarantee a general solution is, in general, not possible and typically the devised methods implicitly include some form of knowledge about the problem under consideration.

The difficulty associated with this problem has lead to a wide variety of approaches which have also been tailored for the network type under consideration. Quinlan (1998) provides a review of many of the methods attempted, primarily for MLP networks, and places them in the biological context of the development of real brains. An example of a more quantitative engineering approach that makes use of the SVD to prune MLP networks is presented in Stepniewski and Jorgensen (1998). An example of structural optimisation for RBF networks includes the work on minimal resource-allocating networks (M-RAN) by Yingwei *et al* (1997, 1998) and (Sundararajan and Saratchandran, 2000), which was based on the work by Platt (1991), while the work of Schwenler *et al* (2001) also provides an overview of training all the parameters of an RBF network. As LMN's can be seen as an extension of RBF's many of the techniques suggested for RBF networks could potentially be extended to LMN's. The reader is referred to Zbikowski *et al* (1994) or (Murray-Smith and Johansen, 1997) for a review of techniques applied to LMN's.

3.4 DETERMINING THE NETWORK JACOBIAN

In this section we concentrate on determining the network Jacobian defined by:

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} \equiv \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial x_1} & \cdots & \frac{\partial \hat{y}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_q}{\partial x_1} & \cdots & \frac{\partial \hat{y}_q}{\partial x_n} \end{bmatrix}. \tag{3.4.1}$$

This quantity can be very useful in adaptive control and dynamic system identification as it directly relates the systems output sensitivity to input variations. This is commonly related to plant sensitivity or system parameters which are used for control law synthesis. When a neural network is used to model a non-linear system this quantity in effect provides an instantaneous linearization of the identified non-linear system.

3.4.1 DETERMINING MLP NETWORK JACOBIAN INFORMATION

Determining the network Jacobian information in an MLP network is simple when one recognizes that the input layer can be treated as any other hidden layer and the recursion relationship (3.3.102) can be applied one more time to get:

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial [\boldsymbol{x}^T, 1]^T} = \frac{\partial \hat{\boldsymbol{y}}}{\partial G^{(1)}} = \Theta^{(2)} G'^{(2)} \frac{\partial \hat{\boldsymbol{y}}}{\partial G^{(2)}}. \tag{3.4.2}$$

The last row associated with the bias term can be ignored, as is done in the normal back propagation algorithm. The implementation is thus built into the back propagation algorithm in Listing 3.3.1, and shows up in line 40, or variable `dYdG,` which contains the required Jacobian information when execution of the loop (10-60) is complete.

3.4.2 DETERMINING RBF JACOBIAN INFORMATION

Consider the network approximation described by equation (3.2.7). To determine the Jacobian of such a network we may directly differentiate with respect to the input vector:

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} \equiv \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial x_1} & \cdots & \frac{\partial \hat{y}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_q}{\partial x_1} & \cdots & \frac{\partial \hat{y}_q}{\partial x_n} \end{bmatrix} = \sum_{I \in I_0} \boldsymbol{\theta}_I \frac{dg(\boldsymbol{x} - \xi_I)}{d\boldsymbol{x}} \quad I_o = \left\{ I \mid \left\| \boldsymbol{x} - \xi_I \right\|_2 \leq \rho, \boldsymbol{x} \in K \subset \Re^n, \xi_I \in \xi_L \right\}. \tag{3.4.3}$$

Expanding the differential gives:

$$\begin{aligned} \frac{dg(\boldsymbol{x} - \xi_I)}{d\boldsymbol{x}} &= \frac{d}{d\boldsymbol{x}} \left(e^{-\left((\boldsymbol{x} - \xi_I)^T \Sigma (\boldsymbol{x} - \xi_I) \right)} \right) \\ &= -2(\boldsymbol{x} - \xi_I)^T \Sigma g(\boldsymbol{x} - \xi_I) . \end{aligned} \tag{3.4.4}$$

Substituting (3.4.4) into (3.4.3) results in an expression for the RBF network Jacobian:

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} = -2 \sum_{I \in I_0} \boldsymbol{\theta}_I (\boldsymbol{x} - \xi_I)^T \Sigma g(\boldsymbol{x} - \xi_I) . \tag{3.4.5}$$

We note that all the terms in equation (3.4.5) are calculated or are available while computing the activation rule. The only additional computational costs are those associated with the explicit product terms contained in equation (3.4.5). Furthermore, as a significant proportion of computation is performed during the *activation rule* computation, the overhead associated with the Jacobian calculation is minimal whether the network is being trained or not. This observation is in stark contrast to the MLP where a significant proportion of the Jacobian calculation is performed while *training* the network. This means that once an RBF network has been trained and the training algorithms have been switched off, *all* the processing resources associated with the training are made available for other tasks. For the MLP, only *limited* processing power can be relinquished as it is still being used in evaluating the Jacobian.

We may also express equation (3.4.5) using only matrix operations. Let us assume that the set I_0 contains l vector elements and that ${}_j\xi_{I_k}$ signifies the j^{th} element of the vector which describes the k^{th} lattice vertex where $1 \leq k \leq l$, $I_k \in I_0 \subset L$ and $\xi_{I_k} \in \xi_L$. We may now construct matrices X_{I_0} by stacking l instances of $\boldsymbol{x}^T$, ξ_{I_0} by stacking each ξ_I^T for all $I \in I_0$, and for notational convenience D_{I_0} :

$$\begin{aligned} X_{I_0} &= \begin{bmatrix} \boldsymbol{x}^T \\ \boldsymbol{x}^T \\ \vdots \\ \boldsymbol{x}^T \end{bmatrix} \quad \xi_{I_0} = \begin{bmatrix} \xi_{I_1}^T \\ \xi_{I_2}^T \\ \vdots \\ \xi_{I_l}^T \end{bmatrix} \\ D_{I_o} &= (X_{I_0} - \xi_{I_0}) \Sigma . \end{aligned} \tag{3.4.6}$$

Using the constructions in equation (3.2.11), (3.4.6) and defining the function diag(.), which constructs a null matrix of appropriate size and places the vector argument along the main diagonal, we may rewrite equation (3.4.5) entirely as a matrix equation:

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} = -2\Theta_{I_0}^T \operatorname{diag}\left(G(\boldsymbol{x}, \xi_{I_0}) \right) D_{I_0} . \tag{3.4.7}$$

Let us now consider how we might implement equation (3.4.7) in an axis orthogonal network arrangement so that we maximise the benefits produced by the activation rule implementation of section 3.2.2. Clearly Θ_{I_0} and $G(\boldsymbol{x},\xi_{I_0})$ are constructed when calculating the activation rule and will not be further addressed. The unresolved part of the problem is the construction of D_{I_0} in an incremental manner while stepping through each input dimension. For an axis orthogonal structure, we may expand this term into its individual components as follows:

$$D_{I_0} = \begin{bmatrix} \frac{(x_1 -_1 \xi_{I_1})}{\sigma_1^2} & \frac{(x_1 -_1 \xi_{I_2})}{\sigma_1^2} & \cdots & \frac{(x_1 -_1 \xi_{I_l})}{\sigma_1^2} \\ \frac{(x_2 -_2 \xi_{I_1})}{\sigma_2^2} & \frac{(x_2 -_2 \xi_{I_2})}{\sigma_2^2} & \cdots & \frac{(x_2 -_2 \xi_{I_l})}{\sigma_2^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{(x_n -_n \xi_{I_1})}{\sigma_n^2} & \frac{(x_n -_n \xi_{I_2})}{\sigma_n^2} & \cdots & \frac{(x_n -_n \xi_{I_l})}{\sigma_n^2} \end{bmatrix}^T . \tag{3.4.8}$$

Clearly, the above matrix will be constructed row by row but it is still unclear how the ${}_j\xi_I$ terms are obtained. We recall from section 3.2.2 that, for the j^{th} input of an axis orthogonal network, the term ${}_j\xi_I$ is repeated a number of times because all the elements except the j^{th} are ignored. This implies that the matrix in (3.4.8) may contain a number of equal terms. Unfortunately, looking at equation (3.4.8), it is not immediately apparent as to how this may be achieved. This is best demonstrated by studying the three input example presented earlier; Assume an input vector such that the 1st, 2nd, and 3rd exponentials are excited by input component one, the 3rd and 4th exponentials are excited along dimension two, and the 6th and 7th exponentials are excited by component three. This results in a set I_0 as follows:

$$I_0 = \left\{ \begin{bmatrix}1\\3\\6\end{bmatrix}^T \begin{bmatrix}2\\3\\6\end{bmatrix}^T \begin{bmatrix}3\\3\\6\end{bmatrix}^T \begin{bmatrix}1\\4\\6\end{bmatrix}^T \begin{bmatrix}2\\4\\6\end{bmatrix}^T \begin{bmatrix}3\\4\\6\end{bmatrix}^T \begin{bmatrix}1\\3\\7\end{bmatrix}^T \begin{bmatrix}2\\3\\7\end{bmatrix}^T \begin{bmatrix}3\\3\\7\end{bmatrix}^T \begin{bmatrix}1\\4\\7\end{bmatrix}^T \begin{bmatrix}2\\4\\7\end{bmatrix}^T \begin{bmatrix}3\\4\\7\end{bmatrix}^T \right\}. \tag{3.4.9}$$

Ignoring the x and σ terms in (3.4.8) we may, for the current example, substitute the *I* subscripts from (3.4.9) to generate the following matrix:

$$\begin{bmatrix} 1\xi[1,3,6] & 1\xi[2,3,6] & 1\xi[3,3,6] & 1\xi[1,4,6] & 1\xi[2,4,6] & 1\xi[3,4,6] & 1\xi[1,3,7] & 1\xi[2,3,7] & 1\xi[3,3,7] & 1\xi[1,4,7] & 1\xi[2,4,7] & 1\xi[3,4,7] \\ 2\xi[1,3,6] & 2\xi[2,3,6] & 2\xi[3,3,6] & 2\xi[1,4,6] & 2\xi[2,4,6] & 2\xi[3,4,6] & 2\xi[1,3,7] & 2\xi[2,3,7] & 2\xi[3,3,7] & 2\xi[1,4,7] & 2\xi[2,4,7] & 2\xi[3,4,7] \\ 3\xi[1,3,6] & 3\xi[2,3,6] & 3\xi[3,3,6] & 2\xi[1,4,6] & 3\xi[2,4,6] & 3\xi[3,4,6] & 3\xi[1,3,7] & 3\xi[2,3,7] & 3\xi[3,3,7] & 3\xi[1,4,7] & 3\xi[2,4,7] & 3\xi[3,4,7] \end{bmatrix}. \tag{3.4.10}$$

As in section 3.2.2 we may replace ${}_j\xi_I$ with the j^{th} element of *I* multiplied by Δ_j, where Δ_j is the spacing of the lattice ξ_L along the dimension *j*, resulting in (3.4.8), for the current example, taking the form:

$$D_{I_0} = \begin{bmatrix} \frac{(x_1-1\Delta_1)}{\sigma_1^2} & \frac{(x_1-2\Delta_1)}{\sigma_1^2} & \frac{(x_1-3\Delta_1)}{\sigma_1^2} & \frac{(x_1-1\Delta_1)}{\sigma_1^2} & \frac{(x_1-2\Delta_1)}{\sigma_1^2} & \frac{(x_1-3\Delta_1)}{\sigma_1^2} & \frac{(x_1-1\Delta_1)}{\sigma_1^2} & \frac{(x_1-2\Delta_1)}{\sigma_1^2} & \frac{(x_1-3\Delta_1)}{\sigma_1^2} & \frac{(x_1-1\Delta_1)}{\sigma_1^2} & \frac{(x_1-2\Delta_1)}{\sigma_1^2} & \frac{(x_1-3\Delta_1)}{\sigma_1^2} \\ \frac{(x_2-3\Delta_2)}{\sigma_2^2} & \frac{(x_2-3\Delta_2)}{\sigma_2^2} & \frac{(x_2-3\Delta_2)}{\sigma_2^2} & \frac{(x_2-4\Delta_2)}{\sigma_2^2} & \frac{(x_2-4\Delta_2)}{\sigma_2^2} & \frac{(x_2-4\Delta_2)}{\sigma_2^2} & \frac{(x_2-3\Delta_2)}{\sigma_2^2} & \frac{(x_2-3\Delta_2)}{\sigma_2^2} & \frac{(x_2-3\Delta_2)}{\sigma_2^2} & \frac{(x_2-4\Delta_2)}{\sigma_2^2} & \frac{(x_2-4\Delta_2)}{\sigma_2^2} & \frac{(x_2-4\Delta_2)}{\sigma_2^2} \\ \frac{(x_3-6\Delta_3)}{\sigma_3^2} & \frac{(x_3-6\Delta_3)}{\sigma_3^2} & \frac{(x_3-6\Delta_3)}{\sigma_3^2} & \frac{(x_3-6\Delta_3)}{\sigma_3^2} & \frac{(x_3-6\Delta_3)}{\sigma_3^2} & \frac{(x_3-6\Delta_3)}{\sigma_3^2} & \frac{(x_3-7\Delta_3)}{\sigma_3^2} & \frac{(x_3-7\Delta_3)}{\sigma_3^2} & \frac{(x_3-7\Delta_3)}{\sigma_3^2} & \frac{(x_3-7\Delta_3)}{\sigma_3^2} & \frac{(x_3-7\Delta_3)}{\sigma_3^2} & \frac{(x_3-7\Delta_3)}{\sigma_3^2} \end{bmatrix}^T . \tag{3.4.11}$$

The common terms and pattern of construction now become apparent facilitating the development of an algorithm to construct D_{I_0} independent of the number of inputs or active units. A Matlab realisation of such an algorithm that is succinct, efficient, and optimally compatible with the activation rule implementation discussed in section 3.2.2, is shown in Listing 3.4.1. For convenience, the activation rule calculations have been included in italicised grey type.

```
10  actv=1; idxw=0; difs=0;                          % Initialize variables
20  for j=1:net.ninp                                 % Loop through each network input
30      expn=(u(j)-net.center{j}).*net.invsigma{j};  % Find exponents for jth input
40      idxe=find(abs(expn)<net.nthld1);             % Determine exponentials activated
50      expn=expn(idxe); actv=actv(:)*exp(-expn.^2); % Multiply with active units
60      vct1=((idxe-(j>1)).*net.vcprd(j));           % Calculate new indexes
70      vct1=vct1(ones(size(idxw,1),1),:);           % Copy vct1 to rows(idxw) rows
80      matx=idxw(:,ones(1,size(idxe,2)));           % Copy idxw to cols(idxe) columns
90      vct2=expn.*net.isigma{j}(idxe);              % Calculate new jacobian info
100     arry=difs(:,:,ones(1,size(idxe,2)));         % Copy difs matrix cols(idxe) times
110     arry(j,:,:)=vct2(ones(size(idxw,1),1),:);    % Copy vct2 to rows(idxw) rows
120     difs=arry(:,:);                              % Update Jacaobian info matrix
130     idxw=vct1(:)+matx(:);                        % Update weight index vector
140     idxa=find(actv>net.nthld2);                  % Make truncation region a ball
150     actv=actv(idxa);                             % Remove inactivated units
160     idxw=idxw(idxa);                             % Remove inactivated weight indexes
170     difs=difs(:,idxa);                           % Remove unactivated jacobian info
180 end                                              % End for loop
190 dydu=-2.*net.weights(:,idxw)*...                 % Calculate Jacobian
              (actv(:,ones(1,net.ninp)).* difs');
200 yhat=net.weights(:,idxw)*actv(:);                % Output result
```

Listing 3.4.1. Axis Orthogonal RBF Jacobian Calculation Implementation.

The code augmentation begins in line 10 where `difs,` the variable for storing the transpose of the D_{I_0} matrix, is set to zero. Next, in line 90, the variable `vct2,` is, for each of the j^{th} input contributing exponentials, set equal to the exponent value used in the activation rule calculation, divided by its corresponding variance. This calculation is the only additional *mathematical* operation required within the loop and actually determines *all* the terms that the j^{th} input contributes to D_{I_0}.

Lines 100, 110 and 120 manipulate `difs` from the previous iteration, together with `vct2` from the current iteration, to ensure the correct structure for the final D_{I_0}. Line 100 repeats the previous iteration `difs` matrix, storing each copy along the third dimension of the variable `arry`. The number of entries, and hence copies of `difs`, in this third dimension is equal to the number of exponentials along the j^{th} input that fall inside the truncation radius. (i.e. The number of elements in `idxe`) Next, line 110 appends a new j^{th} row onto each matrix entry in `arry` and copies `vct2` into each of these rows. Line 120 appends, row wise, each matrix entry along the third dimension of `arry`, to form the new `difs` matrix. The last step within the loop, line 170, is to remove any rows in `difs` that may be associated with units whose activations fall below the activation threshold when the

truncation region is changed from a cube to a ball. To clarify the above-mentioned process Table 3.4.1[12] has been included, demonstrating, for the previously discussed example, how the variables evolve during loop iteration.

j	idxe (100)	idxw (110)	arry (110)	difs (120)
[.]	[.]	[0]	[.]	[0]
1	[1 2 3]	[0]	$-\begin{bmatrix} \frac{1\Delta_1}{\sigma_1^2} \\ \frac{2\Delta_1}{\sigma_1^2} \\ \frac{3\Delta_1}{\sigma_1^2} \end{bmatrix}$	$-\begin{bmatrix} \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \end{bmatrix}$
2	[3 4]	$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	$-\begin{bmatrix} \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \\ \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} \\ \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \\ \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} \end{bmatrix}$	$-\begin{bmatrix} \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} & \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \\ \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} \end{bmatrix}$
3	[6 7]	$\begin{bmatrix} 15 \\ 16 \\ 17 \\ 22 \\ 23 \\ 24 \end{bmatrix}$	$-\begin{bmatrix} \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} & \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \\ \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} \\ \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} \\ \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} & \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \\ \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} \\ \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} \end{bmatrix}$	$-\begin{bmatrix} \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} & \frac{1\Delta_1}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} & \frac{1\Delta_1)}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} & \frac{1\Delta_1)}{\sigma_1^2} & \frac{2\Delta_1}{\sigma_1^2} & \frac{3\Delta_1}{\sigma_1^2} \\ \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{3\Delta_2)}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{3\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} & \frac{4\Delta_2}{\sigma_2^2} \\ \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{6\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} & \frac{7\Delta_3}{\sigma_3^2} \end{bmatrix}$

Table 3.4.1. Example of Variable Evolution During Jacobian Calculation. (x = [0,0,0]).

Once the loop iteration is complete, the resulting `difs` matrix is the transpose of the desired D_{I_0} matrix. The final step in the calculation is to perform the products shown in equation (3.4.7). This step, accomplished in line 190, makes use of the activations contained in `actv`, calculated during the activation rule computation, the final `difs` matrix and the current weight matrix $\Theta_{I_0}^T$ contained in `net.weights(:,idxw)`. Note that the matrix $\text{diag}\left(G(\boldsymbol{x},\xi_{I_0})\right)$ is not specifically constructed, but the product $\text{diag}\left(G(\boldsymbol{x},\xi_{I_0})\right)D_{I_0}$ may be more efficiently and directly implemented by the code `(actv(:,ones(1,net.ninp)).* difs')`.

[12] The table assumes firstly that the input vector is [0,0,0] and secondly that no units are discarded when converting from a cubic to a spherical truncation region.

3.4.3 DETERMINING LMN JACOBIAN INFORMATION

As in section 3.4.2 we may determine the LMN Jacobian by direct differentiation of equation (3.2.13) with respect to the input vector. Thus, by applying the product rule we may state the following:

$$\begin{aligned}\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} &\equiv \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial x_1} & \cdots & \frac{\partial \hat{y}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_q}{\partial x_1} & \cdots & \frac{\partial \hat{y}_q}{\partial x_n} \end{bmatrix} \\ &= \sum_{I\in I_0}\left(\frac{d\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I)}{d\boldsymbol{x}}\hat{g}(\boldsymbol{x}-\xi_I)\right)+\sum_{I\in I_0}\left(\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I)\frac{d\hat{g}(\boldsymbol{x}-\xi_I)}{d\boldsymbol{x}}\right)\end{aligned} \tag{3.4.12}$$

where all variables are the same as those defined in section 3.2.3. Notice that the second method of implementing the mapping $H: K \to \Phi$ discussed in section 3.2.3 allows us to ignore the effects of H because $\phi = \boldsymbol{x}$ and $\eta_L = \xi_I$. This simplifies the equations as we are not required to specifically consider ϕ, or the lattice η_L in the construction of equation (3.4.12).

For the specific case of local linear models, we can expand the first term under the summation to get:

$$\sum_{I\in I_0}\left(\frac{d\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I)}{d\boldsymbol{x}}\hat{g}(\boldsymbol{x}-\xi_I)\right)=\sum_{I\in I_0}\hat{\Theta}_I^T\hat{g}(\boldsymbol{x}-\xi_I). \tag{3.4.13}$$

Simplifying the notation by replacing $\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I)$, $\hat{g}(\boldsymbol{x}-\xi_I)$ and $g(\boldsymbol{x}-\xi_I)$ with $\hat{\boldsymbol{f}}_I$, $\hat{g}_I$ and g_I respectively, then applying the quotient rule and performing the required differentiations, the second term under the summation simplifies as follows:

$$\begin{aligned}\sum_{I\in I_0}\hat{\boldsymbol{f}}_I\frac{d\hat{g}_I}{d\boldsymbol{x}} &= \sum_{I\in I_0}\hat{\boldsymbol{f}}_I\frac{d}{d\boldsymbol{x}}\left(\frac{g_I}{\sum_{J\in I_0}g_J}\right)=\sum_{I\in I_0}\hat{\boldsymbol{f}}_I\left(\frac{\sum_{J\in I_0}g_J\frac{dg_I}{d\boldsymbol{x}}-g_I\frac{d}{d\boldsymbol{x}}\left(\sum_{J\in I_0}g_J\right)}{\left(\sum_{J\in I_0}g_J\right)^2}\right) \\ &= \sum_{I\in I_0}\hat{\boldsymbol{f}}_I\left(\frac{\sum_{J\in I_0}g_J}{\left(\sum_{J\in I_0}g_J\right)^2}2(\xi_I-\boldsymbol{x})^T\Sigma g_I-\frac{g_I}{\left(\sum_{J\in I_0}g_J\right)^2}\sum_{J\in I_0}2(\xi_J-\boldsymbol{x})^T\Sigma g_J\right) \\ &= -2\sum_{I\in I_0}\hat{\boldsymbol{f}}_I\hat{g}_I\left((\boldsymbol{x}-\xi_I)^T\Sigma-\sum_{J\in I_0}(\boldsymbol{x}-\xi_J)^T\Sigma\hat{g}_J\right).\end{aligned} \tag{3.4.14}$$

Substituting equations (3.4.13) and (3.4.14) back into equation (3.4.12) results in:

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}}=\sum_{I\in I_0}\hat{\Theta}_I^T\hat{g}(\boldsymbol{x}-\xi_I)-2\sum_{I\in I_0}\hat{\boldsymbol{f}}(\boldsymbol{x}-\xi_I)\hat{g}(\boldsymbol{x}-\xi_I)\left((\boldsymbol{x}-\xi_I)^T\Sigma-\sum_{J\in I_0}(\boldsymbol{x}-\xi_J)^T\Sigma\hat{g}(\boldsymbol{x}-\xi_J)\right). \tag{3.4.15}$$

As in previous sections we can rewrite this equation as a set of matrix operations. Defining the following matrices:

$$
\begin{aligned}
X_{I_0} &= \begin{bmatrix} \boldsymbol{x}^T \\ \hline \boldsymbol{x}^T \\ \hline \vdots \\ \hline \boldsymbol{x}^T \end{bmatrix} \quad \xi_{I_0} = \begin{bmatrix} \xi_{I_1}^T \\ \hline \xi_{I_2}^T \\ \hline \vdots \\ \hline \xi_{I_l}^T \end{bmatrix} \\
D_{I_o} &= (X_{I_0} - \xi_{I_0})\Sigma \\
G(\boldsymbol{x}, \xi_{I_0}) &= \begin{bmatrix} \hat{g}(\boldsymbol{x} - \xi_{I_1}) \\ \hat{g}(\boldsymbol{x} - \xi_{I_2}) \\ \vdots \\ \hat{g}(\boldsymbol{x} - \xi_{I_l}) \end{bmatrix} \\
F(\boldsymbol{x}, \xi_{I_0}) &= \left[\hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_{I_1}) \;\middle|\; \hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_{I_2} \;\middle|\; \cdots \;\middle|\; \hat{\boldsymbol{f}}(\boldsymbol{x} - \xi_{I_l}) \right]
\end{aligned}
\tag{3.4.16}
$$

then:

$$
\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} = \sum_{I \in I_0} \hat{\Theta}_I^T \hat{g}(\boldsymbol{x} - \xi_I) - 2F(\boldsymbol{x}, \xi_{I_0}) \operatorname{diag}\left(G(\boldsymbol{x}, \xi_{I_0})\right)\left(D_{I_0} - \boldsymbol{I}_l G(\boldsymbol{x}, \xi_{I_0})^T D_{I_0}\right) \tag{3.4.17}
$$

where $\boldsymbol{I}_l$ is a column vector containing *l* rows of one.

By expressing the result as shown above we observe a number of similarities with expressions developed in previous sections permitting much of the earlier implementation to be transferred directly to Listing 3.4.2. Observe that D_{I_0} defined in (3.4.16) is precisely the same as that defined in equation (3.4.6). The method of evaluation is thus exactly the same as in section 3.4.2 and the implementation on lines 10, 90-120 and 170 is identical to that shown in Listing 3.4.1.

The first term in (3.4.15) is almost identical to the term `arry1` first described in equation (3.2.26), the only difference being that the weight matrix used does not include the column of bias values. Ignoring the last column of `arry1` and summing it along the third dimension gives the desired result. Next we see that the expression $F(\boldsymbol{x}, \xi_{I_0}) \operatorname{diag}\left(G(\boldsymbol{x}, \xi_{I_0})\right)$ results in term `arry3` described in equation (3.2.27) and can be reused without any further modification. The calculation of the vector $G(\boldsymbol{x}, \xi_{I_0})^T D_{I_0}$ is directly calculated and stored in a temporary variable called `arry4` from the previously obtained `difs` and `actv` variables. Rather than perform the vector multiplication of $\boldsymbol{I}_l$ with `arry4` to obtain $\left(D_{I_0} - \boldsymbol{I}_l G(\boldsymbol{x}, \xi_{I_0})^T D_{I_0}\right)$ it is more efficient to repeatedly stack the row vector `arry4` on its self and perform the subtraction directly. All the operations described in this paragraph are combined in lines 230 and 240 of Listing 3.4.2 to get the Jacobian matrix result shown in equation (3.4.17).

```
10  actv=1;idxm=0;difs=0;                              % Initialize variables
20  for j=1:net.ninp                                   % Loop through each network input
30      expn=(u(j)-net.center{j}).*net.invsigma{j};    % Find exponents for jth input
40      idxe=find(abs(expn)<net.nthld1);               % Determine exponentials activated
50      expn=expn(idxe); actv=actv(:)*exp(-expn.^2); % Multiply with active units
60      vct1=((idxe-(j>1)).*net.vcprd(j));             % Calculate new indexes
70      vct1=vct1(ones(size(idxm,1),1),:);             % Copy vct1 to length(tmp2) rows
80      matx=idxm(:,ones(1,size(idxe,2)));             % Copy idxm to length(tmp1) cols
90      vct2=expn.*net{n}.isigma{j}(idxe);             % Calculate new jacobian info
100     arry=difs(:,:,ones(1,size(idxe,2)));           % Copy difs matrix cols(idxe) times
110     arry(j,:,:)=vct2(ones(size(idxm,1),1),:);      % Copy vct2 to rows(idxm) rows
120     difs=arry(:,:);                                % Update Jacaobian info matrix
130     idxm=vct1(:)+matx(:);                          % Update model index vector
140     idxa=find(actv>net.nthld2);                    % Make truncation region a ball
150     actv=actv(idxa);                               % Remove inactive units
160     idxm=idxm(idxa);                               % Remove inactive model indexes
170     difs=difs(:,idxa);                             % Remove inactive jacobian info
180 end                                                % End for loop
190 actv=permute(actv(:)./sum(actv(:)),[3,2,1]);       % Reshape actv and normalise
200 arry1=actv(ones(1,net.nout),...                    % Each model validity fcn times mdl wts
          [1, ones(1,net.ninp)],:).*net.weights(:,:,idxm);
210 arry2=[u',1];                                      % Append bias term to input
220 arry3=permute(sum(arry1.*...                       % Calc weighted output of each active mdl
          arry2(ones(1,net.nout),:,ones(1,length(idxm))),2),[1,3,2]);
230 arry4=(difs*actv(:))';                             % Inner most summation in Jacobian calc
240 dydu= sum(arry1(:,1:end-1,:),3)-...                % Calculate final Jacobian Matrix
          2*arry3*(difs'-arry4(ones(1,length(idxm)),:));
250 yhat=sum(arry3,2);                                 % Sum model outputs for network output
```

Listing 3.4.2. Axis Orthogonal LMN Jacobian Calculation Implementation.

3.5 CONCLUSION

The focus of this chapter has been on the specific structural and algorithmic details of the MLP, RBF and LMN networks. Initially the network structures or activation rules were considered and it was shown how, for each network type, simple Matlab code could be used to implement these rules. These implementations were optimised, where possible, for extraction of the network Jacobian information and it was revealed how the efficiencies of an axis orthogonal implementation could be exploited for RBF and LMN networks. Furthermore, it was demonstrated how LMN's could be constructed or expressed in a general neural network framework and their commonalty with RBF networks was highlighted.

In section 3.3 the training of the various network parameters, or the learning rule, was discussed. The learning rule was couched in the form of a non-linear optimisation problem and the commonality with the field of system identification was highlighted. The differing formulations of the problem for batch mode, or off-line, versus on-line, or recursive, approaches was discussed and a rationale for using on-line training was provided. Next it was shown how, by judicious choices of variable construction, the parameter update of all three network types could

be expressed as a linear or pseudo-linear regression problem. This important step provided the foundation for treating the training of all three network types as common, allowing various learning rules to be explored without detailed consideration for the particular network application. The training methods considered included the steepest or gradient descent method and a number of variations and extensions of the recursive least squares approach. The variations discussed, such as the exponential forgetting factor method, attempted to highlight short comings, specifically numerical conditioning issues, and demonstrated how they may be solved. These problems lead to the inevitable discussion of the use of regularization. The important relationship to the Kalman Filter was also included in this sub section.

The final training algorithm presented was a new approach based on the singular value decomposition (SVD.) This method was developed in an attempt to address some of the issues involved with the RLS based algorithms. Although the algorithm has shortcomings in terms of computational effort and the fact that a 'square root' formulation is not used, a number of potential advantages were described, in particular the stability of the information matrix and parameter update equations. Other possible advantages could include natural extensions to perform robust parameter subset selection. The author believes that potential solutions to the mentioned problems exist, although they were not explored in this work. These include the use of rank one SVD update algorithms and square root formulations.

The penultimate section involved with training algorithms dealt with implementation specific issues. These included a description of using dead zones during training and network specific formulations of the algorithm to reduce computational effort. Another important topic of this section described how the regressor variable was constructed for each of the network types. In particular, for MLP's, the famous back propagation algorithm was concisely described, using a matrix formulation, and a Matlab code implementation was presented. This subsection concluded with a short discussion of the trade-off associated with global versus local learning. The final section associated with training algorithms introduced the idea of structural optimisation and a number of references were provided for more detailed reading regarding this topic.

The third and final major section of the chapter described the extraction of the network Jacobian information for each of the network types under consideration. Of particular interest here is the derivation of the expressions for the LMN Jacobian. The implementation details for the Jacobian calculation associated with both the RBF and LMN network are shown to be very similar and can be obtained with minimal computational effort.

To aid in the understanding of how the major equations described in this chapter are exercised we close this chapter with a flow chart, shown in Figure 3.5.1, describing the computational sequence used in this work when identifying a function mapping.

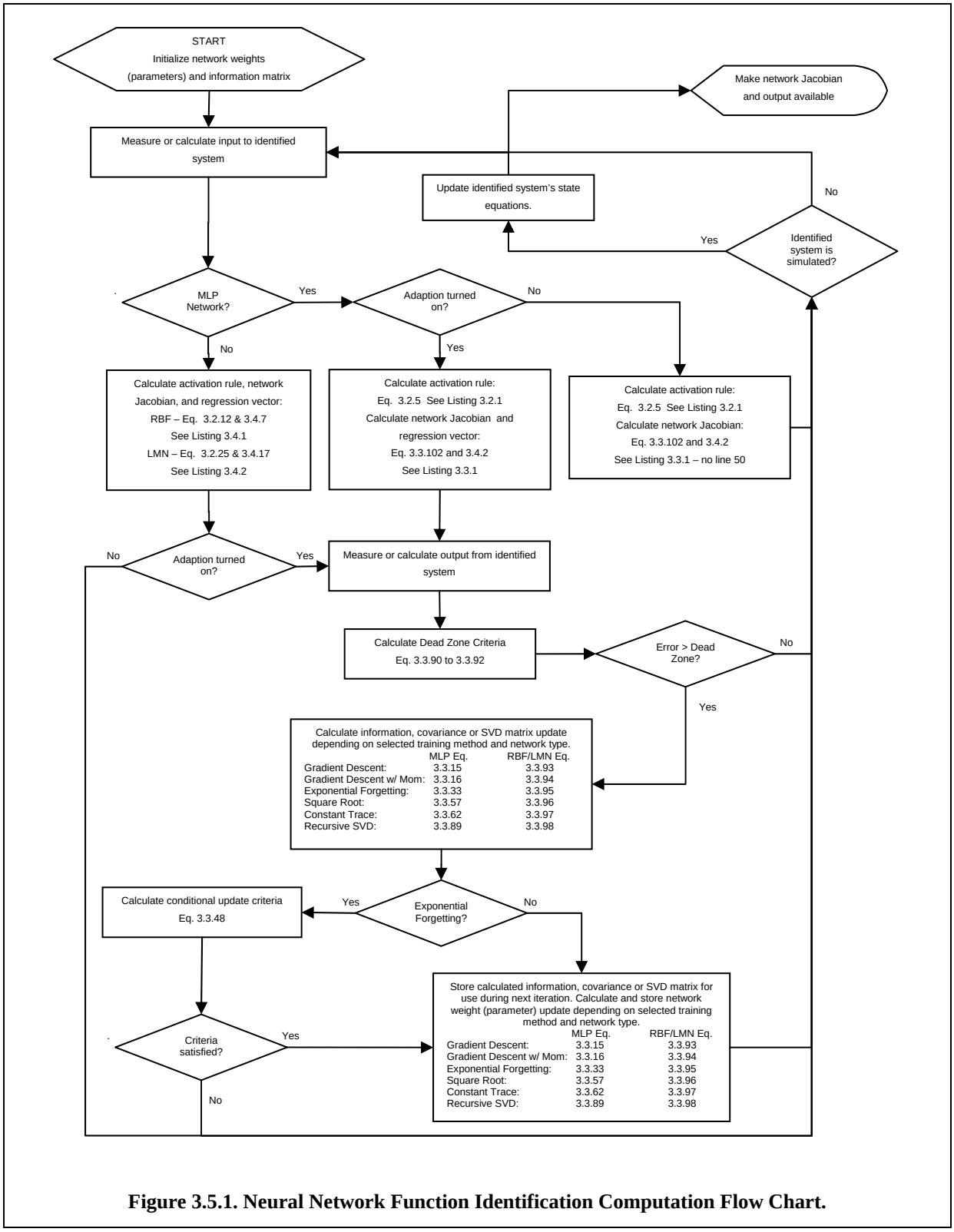


Figure 3.5.1. Neural Network Function Identification Computation Flow Chart.

CHAPTER 4

CONTROL USING NEURAL NETWORKS - NEUROCONTROL

4.1 INTRODUCTION

Until this point the work in this dissertation has focused purely on neural network issues. Attention now turns to the application of these networks in a feedback control environment.

Neurocontrol can be broadly classified (Suykens and Bersini, 1996) into five main areas, namely; Neural adaptive control, Neural optimal control, Reinforcement learning, Internal model control and predictive control and finally NLq stability theory. In this work, our objective is not to design neuro-controllers but only to demonstrate the techniques described in previous chapters. For this reason we focus exclusively on the first classification, i.e. neural adaptive control. The design methodology can be broken down into five main steps:

- The mathematical description or model of the plant that is to be controlled.
- The choice of a controller structure or control law formulation.
- The formulation of an adaptation mechanism to update the controller parameters so that the control objective can be attained.
- A stability analysis of the closed loop system.
- Determination of conditions required for the convergence of the parameters to an acceptable solution.

To begin the chapter however, we first attempt to place this work in context by providing a brief history of feedback control culminating in the advent of neurocontrol. Next the discussion moves to the first three of the design steps given above in the sections entitled, Plant Description, Control Law Formulation and Adaptation Mechanisms. The last of these three sections also shows how the network Jacobian information is used, explaining the motivation behind the development of the Jacobian algorithms in chapter three. The final two design steps above are theoretical and complex in nature and are considered beyond the scope of this work. This does not mean that these steps should be ignored, indeed they are vital to the successful, practical and safe implementation of a real-world controller, even though they are purely theoretical in nature. Finally, the chapter closes with conclusions.

4.2 HISTORICAL BACKGROUND

The use of feedback control can be traced back more than 2000 years to Greece, where mechanisms for regulating water level were devised. Probably the most recognized early control system was the fly ball governor for steam engine speed control devised by James Watt in 1769. In 1868 James Clark Maxwell analysed a differential equation model of a governor by linearizing about an equilibrium point and showed that stability depended upon the state of a characteristic equation having negative real parts. E.J. Routh improved on this work in 1877 and is credited with devising the first mathematically based stability criteria for higher order systems. Prior to World War II Bode and Nyquist developed methods for analysing more complex feedback amplifiers and the PID controller was developed.

During World War II the demand for control of more sophisticated systems led to an explosion in what was now recognized as the independent discipline of control engineering. Notably during this period Evans developed the root locus method of stability analysis, work which may be considered an extension of Maxwell and Routh's earlier work. The use of the Laplace transform and the s-domain continued to flourish after the war. Russian mathematicians meanwhile, particularly Lyapunov, independently produced significant work utilizing time-domain formulations.

With the advent of the space age in the 1950's, western theorists began to explore the more sophisticated approaches made possible by the digital computer from which arose the important work of Kalman and Bellman in utilizing ordinary differential equation (ODE) models. Optimal control, first introduced by Weiner and Phillips during World War II, was now extended using the calculus of variations, to non-linear systems and the Z-transform and difference equations gave birth to modern digital control.

Non-linear and / or time varying systems were, and to this day, remain challenging to analyse in a general framework. Adaptive control was one of the techniques developed, beginning in the 1950's, in an attempt to answer this challenge, primarily in connection with the design of autopilots for high-performance aircraft. Initially adaptive control was based mainly on heuristic ideas and early dramatic failures resulted in the approach falling out of favour. As state space, stability theory and stochastic approaches evolved in the 1960's renewed interest in adaptive methods appeared. Dynamic programming (introduced by Bellman) enhanced the understanding of adaptive techniques and the fundamental contributions of Tsypkin showed that many schemes for learning and adaptive control could be expressed in a common framework. Increased understanding of system identification and robustness coupled with proofs for stability of adaptive systems sparked a renaissance in the field and commercially available adaptive controllers began to appear in the 1980's.

Meanwhile, in computer science and other related fields researchers were intrigued by, and studying systems that could emulate the behaviour of biological systems. The field of Artificial Intelligence (AI) was being shaped by contributions such as Zadeh's seminal paper on fuzzy logic in 1965. Others, beginning with Hebb, Rosenblatt, McCulloch and Pitts, and later Hopfield, Carpenter and Gossberg, Werbos, Albus, Kohonen, Hinton, and, Rumelhart and McClelland, were laying the foundation for what would become the field of neural networks. Concurrently, methods such as Genetic Algorithms and Expert Systems were also being developed.

Slowly the fields encompassed by feedback control, adaptive control and stochastic control became grouped under the general heading of conventional control. In the 1970's the fields of AI, neural networks, genetic algorithms and conventional control began to merge resulting in what is today regarded as intelligent control. More specifically, the amalgamation of control theory with neural network paradigms is referred to as neurocontrol.

4.3 PLANT DESCRIPTION

A general mathematical form for multi-input multi-output (MIMO) discrete[13] time plant description is the state space representation:

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{h}(\boldsymbol{x}[k], \boldsymbol{u}[k]) \\ \boldsymbol{y}[k] &= \boldsymbol{g}(\boldsymbol{x}[k], \boldsymbol{u}[k]) \end{aligned}$$

where $\boldsymbol{x}[k] \in \Re^n$, $\boldsymbol{u}[k] \in \Re^p$ and $\boldsymbol{y}[k] \in \Re^q$ are vectors describing the plant state, inputs and outputs respectively at the k^{th} sample in time, and the functions $\boldsymbol{h} : \Re^{n+p} \to \Re^n$ and $\boldsymbol{g} : \Re^{n+p} \to \Re^q$ are vector valued functions. To simplify what follows we shall restrict this description to a single-input single-output (SISO) plant where the function $\boldsymbol{g}(.)$ is assumed to be a function of $\boldsymbol{x}[k]$ only. Furthermore, these functions are minimally assumed to be continuous. This gives:

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{h}(\boldsymbol{x}[k], u[k]) \\ y[k] &= g(\boldsymbol{x}[k]) \,. \end{aligned} \tag{4.3.1}$$

We now need to determine under what conditions this description can be made equivalent to the problem definition described in section 2.4.1, which formed the archetype I/O representation environment for the neural network approximation problem. The derivation below closely follows that presented in van Breemen (1997).

Input and output sequences obtained over the next *n* samples of future time may be denoted at the k^{th} sample time by:

$$\begin{aligned} \boldsymbol{y}_n[k] &= \big(y[k], y[k+1], \ldots, y[k+n-1]\big)^T \\ \boldsymbol{u}_n[k] &= \big(u[k], u[k+1], \ldots, u[k+n-1]\big)^T \,. \end{aligned} \tag{4.3.2}$$

Recursively iterating the state update equation gives:

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{h}\big(\boldsymbol{x}[k], u[k]\big) && = \boldsymbol{h}_1\big(\boldsymbol{x}[k], \boldsymbol{u}_1[k]\big) \\ \boldsymbol{x}[k+2] &= \boldsymbol{h}\big(\boldsymbol{x}[k+1], u[k+1]\big) = \boldsymbol{h}\big(\boldsymbol{h}\big(\boldsymbol{x}[k], u[k]\big), u[k+1]\big) && = \boldsymbol{h}_2\big(\boldsymbol{x}[k], \boldsymbol{u}_2[k]\big) \\ &\vdots \\ \boldsymbol{x}[k+n] &= \boldsymbol{h}\big(\boldsymbol{x}[k+n-1], u[k+n-1]\big) && = \boldsymbol{h}_n\big(\boldsymbol{x}[k], \boldsymbol{u}_n[k]\big) \,. \end{aligned} \tag{4.3.3}$$

Similarly, the sequence $\boldsymbol{y}_n[k]$ can be expressed by:

$$\begin{aligned} y[k] &= g\big(\boldsymbol{x}[k]\big) && = g_0\big(\boldsymbol{x}[k]\big) \\ y[k+1] &= g\big(\boldsymbol{x}[k+1], u[k+1]\big) = g\big(\boldsymbol{h}_1\big(\boldsymbol{x}[k], \boldsymbol{u}_1[k]\big)\big) && = g_1\big(\boldsymbol{x}[k], \boldsymbol{u}_1[k]\big) \\ &\vdots \\ y[k+n-1] &= g\big(\boldsymbol{h}_{n-1}\big(\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]\big)\big) && = g_{n-1}\big(\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]\big) \end{aligned} \tag{4.3.4}$$

[13] Equivalent continuous time expressions can be obtained for all the work described in this chapter, however, only discrete time implementations will be discussed.

or:

$$\boldsymbol{y}_n[k] = \boldsymbol{g}_n\left(\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]\right) \tag{4.3.5}$$

where $\boldsymbol{g}_n = \left(g_0(.), g_1(.), \cdots, g_{n-1}(.)\right)$. Introducing an extended function we see that (4.3.5) can be expressed as:

$$\begin{bmatrix} \boldsymbol{y}_n[k] \\ \boldsymbol{u}_{n-1}[k] \end{bmatrix} = \begin{bmatrix} \boldsymbol{g}_n\left(\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]\right) \\ \boldsymbol{u}_{n-1}[k] \end{bmatrix} = \boldsymbol{G}\left(\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]\right) \tag{4.3.6}$$

where $\boldsymbol{G} : \Re^{2n-1} \to \Re^{2n-1}$. As the input and output dimensions of $\boldsymbol{G}(.)$ are the same then the function expressed in (4.3.6) might be inverted to get:

$$\begin{bmatrix} \boldsymbol{x}[k] \\ \boldsymbol{u}_{n-1}[k] \end{bmatrix} = \boldsymbol{G}^{-1}\left(\boldsymbol{y}_n[k], \boldsymbol{u}_{n-1}[k]\right) = \begin{bmatrix} \boldsymbol{G}_{\boldsymbol{x}}^{-1}\left(\boldsymbol{y}_n[k], \boldsymbol{u}_{n-1}[k]\right) \\ \boldsymbol{G}_{\boldsymbol{u}}^{-1}\left(\boldsymbol{y}_n[k], \boldsymbol{u}_{n-1}[k]\right) \end{bmatrix} \tag{4.3.7}$$

implying:

$$\boldsymbol{x}[k] = \boldsymbol{G}_{\boldsymbol{x}}^{-1}\left(\boldsymbol{y}_n[k], \boldsymbol{u}_{n-1}[k]\right). \tag{4.3.8}$$

The function $\boldsymbol{G}_{\boldsymbol{x}}^{-1}(.)$ is called the observer of the system expressed in (4.3.1).

Consider now the Inverse Function Theorem, which states (Parker 1997): "*If $f(x)$ is a continuously differentiable function of the Euclidian n-space to itself and at a point x_0 the matrix with the entry $\partial f_i / \partial x_j$ in the i^{th} row and j^{th} column is non-singular, then there is a continuously differentiable function f^{-1} defined in the neighbourhood of $f(x_0)$ which is an inverse function for $f(x)$ at all points near x_0.*"

That is, if the determinant of the Jacobian of $\boldsymbol{G}(.)$, evaluated at the point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1})$, is not equal to zero then the inverse $\boldsymbol{G}^{-1}(.)$ exists in the region around $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1})$. The Jacobian matrix can be obtained by simple matrix differentiation of (4.3.6) which, together with the fact that the future input variable sequence to the plant at the k^{th} sample $\boldsymbol{u}_{n-1}[k]$ is independent[14] of the state vector at the k^{th} sample, gives:

$$\begin{aligned} \left.\frac{\partial \boldsymbol{G}\left(\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]\right)}{\partial[\boldsymbol{x}[k], \boldsymbol{u}_{n-1}[k]]}\right|_{[\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1}]} &= \left.\begin{bmatrix} \dfrac{\partial \boldsymbol{y}_n[k]}{\partial \boldsymbol{x}[k]} & \dfrac{\partial \boldsymbol{y}_n[k]}{\partial \boldsymbol{u}_{n-1}[k]} \\ \dfrac{\partial \boldsymbol{u}_{n-1}[k]}{\partial \boldsymbol{x}[k]} & \dfrac{\partial \boldsymbol{u}_{n-1}[k]}{\partial \boldsymbol{u}_{n-1}[k]} \end{bmatrix}\right|_{[\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1}]} \\ &= \left.\begin{bmatrix} \dfrac{\partial \boldsymbol{y}_n[k]}{\partial \boldsymbol{x}[k]} & \dfrac{\partial \boldsymbol{y}_n[k]}{\partial \boldsymbol{u}_{n-1}[k]} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}\right|_{[\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1}]}. \end{aligned} \tag{4.3.9}$$

For this matrix to be non-singular the following condition must be satisfied:

$$\det\left(\left.\frac{\partial \boldsymbol{y}_n[k]}{\partial \boldsymbol{x}[k]}\right|_{[\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1}]}\right) \neq 0. \tag{4.3.10}$$

[14] This is strictly true for the open loop plant but, under closed loop conditions, this assumption may be violated and a more complex condition will result. This fact is not addressed in the source reference, but has implications for the identification and stability of the closed loop system.

If condition (4.3.10) is satisfied, then using equations (4.3.7), (4.3.8) and (4.3.3) we see that the state vector at sample $(k+n)$ can be expressed for some local region around $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{n-1})$ by:

$$\boldsymbol{x}[k+n] = \boldsymbol{h}_n\left(\boldsymbol{G}_{\boldsymbol{x}}^{-1}\left(\boldsymbol{y}_n[k], \boldsymbol{u}_{n-1}[k]\right), \boldsymbol{u}_n[k]\right).$$

Substituting this expression into the output equation of (4.3.1) gives the following locally equivalent I/O system description:

$$\begin{aligned} y[k+n] &= g(\boldsymbol{x}[k+n]) \\ &= g\left(\boldsymbol{h}_n\left(\boldsymbol{G}_{\boldsymbol{x}}^{-1}\left(\boldsymbol{y}_n[k], \boldsymbol{u}_{n-1}[k]\right), \boldsymbol{u}_n[k]\right)\right) \\ &= f(\boldsymbol{y}_n[k], \boldsymbol{u}_n[k]) . \end{aligned}$$

Shifting the time indexes and substituting the original sequences in (4.3.2) yields:

$$y[k] = f(y[k-1], \ldots, y[k-n], u[k-1], \ldots, u[k-n]) .$$

This is the same form as equation (2.4.1) with $n_a = n_b$, $n_k = 0$, and $e[k] = 0$. Mechanical algebraic extensions of the above derivation can be made to include all these variables but are not illustrative and are omitted. The condition (4.3.10) can also be readily extended for p input, q output MIMO systems which we present here without derivation:

$$\det\left(\left.\frac{\partial \boldsymbol{y}_{j,n}[k]}{\partial \boldsymbol{x}[k]}\right|_{[\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}_{1,n-1}, \bar{\boldsymbol{u}}_{2,n-1}, \ldots, \bar{\boldsymbol{u}}_{p,n-1}]}\right) \neq 0, \qquad 1 \le j \le q \tag{4.3.11}$$

where $\boldsymbol{y}_{j,n}[k]$ is the future output time sequence defined in (4.3.2) for the j^{th} output and $\bar{\boldsymbol{u}}_{i,n-1}$ is the selected operating region for the i^{th} future input time sequence.

4.4 CONTROL LAW FORMULATION

As stated earlier, the objective for the control law formulation in this work is only to demonstrate the techniques described in previous chapters. Consequently only two control law formulations are presented here, namely; Series Inverse control and the Minimum Degree Pole Placement (MDPP) design. These approaches were chosen for their simplicity and ability to elucidate the key points of earlier chapters. Again, in the interests of simplicity, mainly SISO systems will be considered although many of the techniques presented can be extended to MIMO cases.

4.4.1 SERIES INVERSE CONTROL

Assume a plant can be described by a mapping $H : U \to Y$, where $U \subseteq \Re^p$ is the input signal space and $Y \subseteq \Re^q$, $\boldsymbol{y} \in Y$ is the output signal space. Further assume there exists an inverse mapping of H given by $G : Y_m \to U$, where $Y_m \subseteq \Re^q$ is called the reference signal space. The plant described by H can then be made to track a reference signal $\boldsymbol{y_m} \in Y_m$ by setting $\boldsymbol{u} = G(\boldsymbol{y_m})$, $\boldsymbol{u} \in U$ and noting that:

$$\boldsymbol{y} = H(\boldsymbol{u}) = H(G(\boldsymbol{y_m})) = \boldsymbol{y_m} . \tag{4.4.1}$$

Key to this approach is determining when the inverse exists and whether it can be formed in such a way that it can be represented by a neural network.

The condition for existence can be determined for square i.e. $p = q$, MIMO systems. Let the plant be represented by the following I/O description:

$$\begin{array}{rcl} y_1[k+1] & = & h_1(\boldsymbol{y}^T_{1,n_{y1}}[k],\ldots,\boldsymbol{y}^T_{p,n_{yp}}[k],\boldsymbol{u}^T_{1,n_{u1}}[k],\ldots,\boldsymbol{u}^T_{p,n_{up}}[k]) \\ \vdots & & \vdots \\ y_p[k+1] & = & h_p(\boldsymbol{y}^T_{1,n_{y1}}[k],\ldots,\boldsymbol{y}^T_{p,n_{yp}}[k],\boldsymbol{u}^T_{1,n_{u1}}[k],\ldots,\boldsymbol{u}^T_{p,n_{up}}[k]) \end{array} = \begin{array}{l} h_1(\Phi[k],u_1[k],\ldots,u_p[k]) \\ \vdots \\ h_p(\Phi[k],u_1[k],\ldots,u_p[k]) \end{array} \quad (4.4.2)$$

where:

$$\begin{aligned} \boldsymbol{y}_{i,n_{yi}}[k] &= \left(y_i[k],y_i[k-1],\ldots,y_i[k-n_{y_i}+1]\right)^T \\ \boldsymbol{u}_{i,n_{ui}}[k] &= \left(u_i[k],u_i[k-1],\ldots,u_i[k-n_{u_i}+1]\right)^T \\ \Phi[k] &= \left(\boldsymbol{y}^T_{1,n_{y1}}[k],\ldots,\boldsymbol{y}^T_{p,n_{yp}}[k],\boldsymbol{u}^T_{1,n_{u1}-1}[k],\ldots,\boldsymbol{u}^T_{p,n_{up}-1}[k]\right)^T \\ \boldsymbol{u}_{i,n_{ui}-1}[k] &= \left(u_i[k-1],\ldots,u_i[k-n_{u_i}+1]\right)^T . \end{aligned} \quad (4.4.3)$$

Following the archetype (4.4.1) the inverse control problem is now, for $1 \le i \le p$, to determine:

$$\begin{aligned} y_i[k+1] &= h_i(\Phi[k],u_1[k],\ldots,u_p[k]) \\ &= h_i\left(\Phi[k],\boldsymbol{G}\left(\Phi[k],y_{m_1}[k],\ldots,y_{m_p}[k]\right)\right) \\ &= y_{m_i}[k] \end{aligned}$$

where the desired control law is:

$$\left(u_1[k],\ldots,u_p[k]\right) = \boldsymbol{G}\left(\Phi[k],y_{m_1}[k],\ldots,y_{m_p}[k]\right). \quad (4.4.4)$$

Using an extended function gives:

$$\begin{aligned} \begin{bmatrix} \Phi[k] \\ y_1[k+1] \\ \vdots \\ y_p[k+1] \end{bmatrix} &= \begin{bmatrix} \Phi[k] \\ h_1(\Phi[k],u_1[k],\ldots,u_p[k]) \\ \vdots \\ h_p(\Phi[k],u_1[k],\ldots,u_p[k]) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{H}_\Phi\left(\Phi[k],u_1[k],\ldots,u_p[k]\right) \\ \boldsymbol{H}_y\left(\Phi[k],u_1[k],\ldots,u_p[k]\right) \end{bmatrix} \\ &= \boldsymbol{H}(\Phi[k],u_1[k],\ldots,u_p[k]) \\ &= \boldsymbol{H}\left(\Phi[k],\boldsymbol{G}\left(\Phi[k],y_{m_1}[k],\ldots,y_{m_p}[k]\right)\right) = \begin{bmatrix} \Phi[k] \\ y_{m_1}[k] \\ \vdots \\ y_{m_p}[k] \end{bmatrix}. \end{aligned}$$

Clearly, if the inverse of $\boldsymbol{H}(.)$ exists then the function $\boldsymbol{G}(.)$ can be realised. Therefore, obtaining the Jacobian of $\boldsymbol{H}(.)$ about some operating point $[\overline{\Phi},\overline{u}_1,\ldots,\overline{u}_p]$, and using the Inverse Function Theorem we can test for the local existence of $\boldsymbol{G}(.)$. Thus:

$$\left.\frac{\partial \boldsymbol{H}(\Phi[k],u_1[k],\ldots,u_p[k])}{\partial[\Phi[k],u_1[k],\ldots,u_p[k]]}\right|_{[\overline{\Phi},\overline{u}_1,\ldots,\overline{u}_p]} = \left.\begin{bmatrix} \dfrac{\partial \Phi[k]}{\partial \Phi[k]} & \dfrac{\partial \Phi[k]}{\partial[u_1[k],\ldots,u_p[k]]} \\ \dfrac{\partial \boldsymbol{H}_y(\Phi[k],u_1[k],\ldots,u_p[k])}{\partial \Phi[k]} & \dfrac{\partial \boldsymbol{H}_y(\Phi[k],u_1[k],\ldots,u_p[k])}{\partial[u_1[k],\ldots,u_p[k]]} \end{bmatrix}\right|_{[\overline{\Phi},\overline{u}_1,\ldots,\overline{u}_p]}$$

$$= \left.\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \dfrac{\partial \boldsymbol{H}_y(\Phi[k],u_1[k],\ldots,u_p[k])}{\partial \Phi[k]} & \dfrac{\partial \boldsymbol{H}_y(\Phi[k],u_1[k],\ldots,u_p[k])}{\partial[u_1[k],\ldots,u_p[k]]} \end{bmatrix}\right|_{[\overline{\Phi},\overline{u}_1,\ldots,\overline{u}_p]}$$

resulting in the following condition:

$$\det\left(\left.\frac{\partial \boldsymbol{H}_y(\Phi[k],u_1[k],\ldots,u_p[k])}{\partial[u_1[k],\ldots,u_p[k]]}\right|_{[\overline{\Phi},\overline{u}_1,\ldots,\overline{u}_p]}\right) \neq 0\,,$$

or expanding to show the individual terms gives:

$$\left.\begin{vmatrix} \dfrac{\partial h_1}{\partial u_1[k]} & \dfrac{\partial h_1}{\partial u_2[k]} & \cdots & \dfrac{\partial h_1}{\partial u_p[k]} \\ \dfrac{\partial h_2}{\partial u_1[k]} & \dfrac{\partial h_2}{\partial u_2[k]} & \cdots & \dfrac{\partial h_2}{\partial u_p[k]} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial h_p}{\partial u_1[k]} & \dfrac{\partial h_p}{\partial u_2[k]} & \cdots & \dfrac{\partial h_p}{\partial u_p[k]} \end{vmatrix}\right|_{[\overline{\Phi},\overline{u}_1,\ldots,\overline{u}_p]} \neq 0\,. \tag{4.4.5}$$

Summarising, the series inverse control law shown in (4.4.4) may be used to control the system in equation (4.4.2), given the definitions in (4.4.3), provided the condition shown in (4.4.5) is satisfied. We note that the form of the system in (4.4.2) and the control law in (4.4.4) can be solved using a neural network approximation.

The approach can be enhanced by observing that the existence condition is dependent only on the plant dynamics being invertible. If the reference signal is derived from a dynamic reference model described by $S : R \to Y_m$ mapping some demand signal $\boldsymbol{r} \in R \subseteq \Re^q$ then (4.4.1) may be modified as follows:

$$\boldsymbol{y} = H(\boldsymbol{u}) = H(G(\boldsymbol{y_m})) = H(G(S(\boldsymbol{r}))) = H(\widetilde{G}(\boldsymbol{r})) = \boldsymbol{y_m} = S(\boldsymbol{r})\,. \tag{4.4.6}$$

That is, the reference dynamics can be absorbed into $\boldsymbol{G}(.)$, without violating the analysis above, to yield the following control law (henceforth referred to as the SIC law):

$$\begin{aligned} \left(u_1[k],\ldots,u_p[k]\right) &= \widetilde{\boldsymbol{G}}\left(\Phi[k],\boldsymbol{r}_{1,n_1}[k],\ldots,\boldsymbol{r}_{p,n_{r_p}}[k]\right) \\ \boldsymbol{r}_{i,n_{r_i}}[k] &= \left(r_i[k],r_i[k-1],\ldots,r_i[k-n_{r_i}+1]\right)^T\,. \end{aligned} \tag{4.4.7}$$

The inverse controller essentially "sees" the forward dynamics of the reference system in series with the inverse dynamics of the plant. The reference signal does not appear in (4.4.7) because ideally $\boldsymbol{y}[k]$ tracks $\boldsymbol{y_m}[k]$ making it unnecessary to use both variables.

4.4.2 Minimum Degree Pole Placement Design

The pole placement design procedure has been covered extensively in Åström and Wittenmark (1995, 1997) and the minimum degree pole placement approach covered here closely follows their exposition. Only the basic

technique is shown, however, it may be considered the basis for many other linear methods such as LQG, moving average and minimum variance controllers. The method presented is simple, illustrative and if properly applied, can be practically useful.

Assume that a SISO plant can be described by the I/O representation (see section 4.3 for conditions) of equation (2.4.1). Additionally, assume that at any instant in time the function $f(.)$ of equation (2.4.1) can be represented by two polynomials in the forward shift operator[15] [16] q. That is:

$$\begin{aligned} y[k+n_a]+a_1y[k+n_a-1]+\cdots+a_{n_a}y[k] &= b_0u[k+n_b]+b_1u[k+n_b-1]+\cdots+b_{n_b}u[k] \\ A(q)y[k] &= B(q)u[k] \\ A(q) &= \left(q^{n_a}+a_1q^{n_a-1}+\ldots+a_{n_a}\right) \\ B(q) &= \left(b_0q^{n_b}+b_1q^{n_b-1}+\ldots+b_{n_b}\right). \end{aligned} \tag{4.4.8}$$

In a time invariant linear system the coefficients of $A(q)$ and $B(q)$ are constant but in this work they are permitted to change with time. However, for the purposes of the discussion in this section, we shall assume that they are constant and available. Furthermore, it is assumed that $A(q)$ and $B(q)$ do not have any common factors (i.e. they are relatively prime) and that the coefficient of the highest power term in $A(q)$ is always one, that is, $A(q)$ is monic. Finally, we require that the degree of $A(q)$ (designated $\deg A$) is greater than the degree of $B(q)$, i.e. $n_a > n_b$. The difference in degree, $n_k = \deg A - \deg B$, is called the pole excess and represents the integer part of the ratio of plant time delay to sampling period.

Given the plant description of the previous paragraph it is hypothesized that such a plant may be controlled by the general two degree of freedom linear control law described by:

$$R(q)u[k] = T(q)r[k] - S(q)y[k] \tag{4.4.9}$$

where R, S and T are all forward shift operator polynomials. This control law will be referred to as the RST law in the sequel. Rearranging (4.4.9) and removing the explicit notation for the q operator the two degrees of freedom, $-S/R$ and T/R representing negative feedback and positive feedforward transfer operators respectively, are easily seen:

$$u[k] = \frac{T}{R}r[k] - \frac{S}{R}y[k]. \tag{4.4.10}$$

Combining equations (4.4.8) and (4.4.9) results in the following closed loop expressions:

---

[15] The forward shift operator q has the property $q^n f[k] = f[k+n]$ and should not be confused with the output signal dimension of a MIMO plant. The letter q is reused here to conform to the commonly used notation in the literature. As this section deals only with SISO plants the definition of q is clear from the context.

[16] Although q is an operator and the variable z used in the z-Transform is a complex variable, they may be interchanged if care is taken to consider the effects of initial conditions on the equations involved. This fact may be used sporadically in the remainder of this dissertation.

$$y[k] = \frac{BT}{AR + BS} r[k]$$
$$u[k] = \frac{AT}{AR + BS} r[k] . \tag{4.4.11}$$

The closed loop characteristic polynomial is thus represented by the Diophantine equation:

$$AR + BS = A_c . \tag{4.4.12}$$

Given A_c as a design variable, and A and B as system data satisfying the plant description above, then equation (4.4.12) will always provide multiple solutions for polynomials *R* and *S*. The existence of multiple solutions (as opposed to a single solution) can be seen by inspection if one notes that given an arbitrary polynomial Q and a solution R^0 and S^0 then a new solution to (4.4.12) may be constructed as follows:

$$\begin{aligned} R &= R^0 + QB \\ S &= S^0 - QA . \end{aligned} \tag{4.4.13}$$

Selecting the solution that results in the lowest degree for polynomials *R* and *S* is called the minimum degree solution. Such a solution can be obtained using Euclid's algorithm or by forming the Sylvester matrix and solving the resulting set of linear equations. (See §11.3 of Åström and Wittenmark (1995) or §5.3 of Åström and Wittenmark (1997).)

To solve for the polynomial *T* we introduce a reference signal $y_m[k]$ which is related to the demand signal $r[k]$ by the following dynamic reference model:

$$A_m(q) y_m[k] = B_m(q) r[k] . \tag{4.4.14}$$

If $y[k]$ tracks $y_m[k]$, then substituting (4.4.14) into (4.4.11) gives the following condition:

$$\frac{BT}{AR + BS} = \frac{BT}{A_c} = \frac{B_m}{A_m} \tag{4.4.15}$$

which may be re-arranged to give:

$$T = \frac{A_c B_m}{B A_m} . \tag{4.4.16}$$

The condition (4.4.16) implies a number of factor cancellations involving *BT* and A_c . The polynomial *B* may be factored into two components:

$$B = B^+ B^- . \tag{4.4.17}$$

The first of these components, B^+, is chosen to be monic with well damped stable zeros which are cancelled by the controller. The second, B^-, represents the lightly damped and / or unstable zeros which cannot be cancelled by the controller and must therefore be a factor of B_m, yielding:

$$B_m = B^- B'_m . \tag{4.4.18}$$

Condition (4.4.15) implies that A_m must be a factor of A_c and as B^+ is cancelled it also must be a factor. Therefore A_c may be broken down as follows:

$$A_c = A_o A_m B^+ . \tag{4.4.19}$$

Substituting (4.4.17) through (4.4.19) into (4.4.16) gives

$$T = A_o B'_m . \tag{4.4.20}$$

Finally, factoring $R = R'B^+$ and substituting (4.4.17) and (4.4.19) into (4.4.12) reduces the Diophantine equation to:

$$AR' + B^- S = A_o A_m . \tag{4.4.21}$$

It is instructive to combine (4.4.17) and (4.4.18) to form a partially factored expression for (4.4.19):

$$A_c = \frac{A_o A_m B_m B^+ B^-}{B_m B^-} .$$

A solution is thus obtained if the design variable A_c is provided in the form of a factoring of B and three polynomials A_o, A_m and B_m . Furthermore, substituting (4.4.21) into (4.4.20) reveals:

$$\frac{T}{R} = \frac{A_o B'_m}{R} = \frac{(AR' + B^- S)B'_m}{A_m R} = \frac{AB_m}{BA_m} + \frac{SB_m}{RA_m}$$

which may be substituted into (4.4.10) and combined with (4.4.14) to give:

$$\begin{aligned} u[k] &= \left(\frac{AB_m}{BA_m} + \frac{SB_m}{RA_m} \right) r[k] - \frac{S}{R} y[k] \\ &= \frac{AB_m}{BA_m} r[k] + \frac{SB_m}{RA_m} r[k] - \frac{S}{R} y[k] = \frac{AB_m}{BA_m} r[k] + \frac{S}{R} y_m[k] - \frac{S}{R} y[k] \\ &= \frac{AB_m}{BA_m} r[k] + \frac{S}{R} \big(y_m[k] - y[k] \big) . \end{aligned} \tag{4.4.22}$$

The first term of this equation can be interpreted as a feedforward controller which attempts to cancel the plant dynamics and replace it with the reference model dynamics. The second term acts as a negative feedback controller which endeavours to make the plant output follow the reference model output. Equation (4.4.22) may not be realised as shown due to the inverse plant dynamics A / B . However, it highlights the partial cancellation through the appropriate choice of A_m and B_m .

Given plant data A and B together with design specification data A_o, A_m and B_m , the design algorithm may be summarized as follows:

1. Factor B using equation (4.4.17) such that $B = B^+ B^-$ where B^+ is monic.
2. Solve equation (4.4.21) for the minimum degree solution, determining R' and S using the factor B^- and the supplied specification for A_o and A_m .
3. Determine B'_m from (4.4.18) using the supplied specification for B_m .
4. Obtain $R = R'B^+$, and from (4.4.20), $T = A_o B'_m$ using the previously determined data.
5. Finally, calculate the control signal $u[k]$ using equation (4.4.10).

The design algorithm above provides no indication, other than equation (4.4.18), of how to choose the design polynomials. Let us therefore consider how the choices of A_o, A_m and B_m may be constrained.

4.4.2.1 CONSTRAINTS ON THE DESIGN SPECIFICATION POLYNOMIALS

In order for the controller (4.4.9) to be causal we see that:

$$\begin{aligned} \deg S &\leq \deg R \\ \deg T &\leq \deg R \,. \end{aligned} \tag{4.4.23}$$

Inspection of (4.4.9) reveals that, if the controller is not to introduce any delay, the equality condition in (4.4.23) must be satisfied. That is:

$$\deg R = \deg S = \deg T \,. \tag{4.4.24}$$

Furthermore, dividing both sides of (4.4.12) by A and noting that $\deg A > \deg B$ yields:

$$\deg R = \deg A_c - \deg A \,. \tag{4.4.25}$$

Combining the conditions in (4.4.24) together with equation (4.4.13) implies that, for the minimum degree solution, $\deg S = \deg R \leq \deg B < \deg A$. That is, there is always a solution where:

$$\deg S \leq \deg A - 1 \,. \tag{4.4.26}$$

Substituting (4.4.26) and (4.4.25) into (4.4.24) gives $\deg A_c \geq 2\deg A - 1$. Therefore, choosing the controller with the lowest possible order yields:

$$\deg A_c = 2\deg A - 1 \,. \tag{4.4.27}$$

Taking the polynomial degrees on both sides of equation (4.4.16) yields:

$$\deg T = \deg A_c + \deg B_m - \deg B - \deg A_m \,. \tag{4.4.28}$$

Now substituting (4.4.28) and (4.4.25) into the second condition in (4.4.23) and rearranging gives:

$$\deg A_m - \deg B_m \geq \deg A - \deg B = n_k \,. \tag{4.4.29}$$

This logically implies that the time delay associated with the reference model must be greater than or equal to the time delay of the process being controlled. Furthermore, a controller which has minimum degree requires that the equality condition in (4.4.29) must be satisfied. That is:

$$\deg A_m - \deg B_m = \deg A - \deg B = n_k \,.$$

This may be combined with the knowledge of the attempted cancellation shown in equation (4.4.22) to conclude that A_m and B_m should have the same degree as A and B respectively giving the following conditions:

$$\begin{aligned} \deg A_m &= \deg A \\ \deg B_m &= \deg B \,. \end{aligned} \tag{4.4.30}$$

Now taking the degree of the factors in equation (4.4.19) gives:

$$\deg A_o = \deg A_c - \deg A_m - \deg B^+ \,.$$

Substituting (4.4.27) and into the equation above, and using the first condition in (4.4.30), yields the following condition for A_o :

$$\deg A_o = \deg A - \deg B^+ - 1 \,. \tag{4.4.31}$$

Equations (4.4.30) and (4.4.31) together with (4.4.18) summarize the constraints that must be satisfied by the design specification polynomials to produces a minimum degree controller.

4.4.2.2 FACTORIZATION OF THE PROCESS ZERO POLYNOMIAL

We now turn our attention to the factorization of B in equations (4.4.17) and (4.4.18). There are two special cases associated with this factorization:

1. *All process zeros are cancelled* - If all the plant zeros are well damped and inside the unit circle (i.e. stable) it is possible to cancel them by the choosing $B^+ = B/b_0$ and $B^- = b_0$ in step one of the design algorithm. Condition (4.4.30) requires that the pole excess of the reference model and the process are the same. Furthermore, it is generally required that the static gain is unity. This implies that $B_m = A_m(1)q^{n_k}$ and (4.4.18) gives $B'_m = A_m(1)q^{n_k}/b_0$ and from (4.4.20) $T = A_o A_m(1)q^{n_k}/b_0$. The Diophantine equation (4.4.21) now becomes $AR' + b_0 S = A_o A_m$ which is easily solved by determining $A_0 A_m / A$ and observing that R' is the quotient and $b_0 S$ is the remainder of the solution. Finally, condition (4.4.31) implies A_o should be chosen such that $\deg A_o = \deg A - \deg B - 1 = n_k - 1$.

2. *No process zeros are cancelled* – Following the same sequence of steps presented in case one above we see that if none of the plant zeros are cancelled then $B^+ = 1$, $B^- = B$ and a static gain requirement with condition (4.4.30) implies $B_m = A_m(1)B/B(1)$, giving $T = A_0 A_m(1)/B(1)$. Lastly the Diophantine equation (4.4.21) is simply $AR + BS = A_c = A_o A_m$ where now $\deg A_o = \deg A - 1$. By solving the pole placement problem using state feedback it can be shown (§4.5, Åström and Wittenmark, 1997) that, if no process zeros are cancelled, the closed loop characteristic polynomial A_c may be factored into two polynomials corresponding to the state feedback controller and the state observer. These polynomials correspond to A_m and A_o above and are called the controller and observer polynomials respectively. Arbitrary roots may be assigned to A_m if the system is reachable and to A_o if the system is observable. This implies that one should select the roots (eigenvalues) of A_o with the same considerations in mind as when selecting observer poles.

4.5 ADAPTATION MECHANISMS

An adaptive controller may (amongst many alternatives) be defined as (Åström and Wittenmark, 1995) "…a controller with adjustable parameters and a mechanism for adjusting the parameters." This section deals with the latter part of this definition i.e. the mechanism for adjusting the parameters.

4.5.1 THE ADJUSTED PARAMETERS

The first question to be answered is: "Given the presented control law formulations, what are the parameters to be adjusted?" To answer this consider the previous section where two basic control law formulations were given

– the series inverse controller, or SIC law, expressed by equation (4.4.4) or (4.4.7) and the RST law shown in equation (4.4.10).

In the SIC law the parameters are implicit in the definition of the $\tilde{\boldsymbol{G}}(.)$ function. This function is non-linear and therefore, in this work, assumed to be represented by some form of neural network. The adjusted parameters are thus the network weights defining the network mapping. The activation rule evaluation of the inverse controller network is therefore equivalent to evaluating the control law. We note that if the process dynamics are time invariant then the inverse mapping that the control law is required to represent will, given the correct conditions, also converge to a time invariant solution. That is, if the plant dynamics are time invariant, the SIC law controller parameters will be constant even though the plant may be non-linear because the functional mapping of the control law is itself non-linear.

For the RST law the controller parameters that define the law's functional mapping are the coefficients of the *R, S* and *T* polynomials. The MDPP design algorithm showed how these values could be calculated given the *A* and *B* process polynomials and a design specification. The coefficients of the process polynomials may therefore be considered the adapted parameters and the algebraic calculation of the MDPP algorithm is analogous to the inverse controller network activation rule calculation in the SIC approach. A key difference is that the functional mapping of the RST law is linear and, for a non-linear plant, the process polynomials are valid only in a local region about the current operating condition. This implies that the MDPP algorithm must be continually evaluated as the estimated process polynomials change. This change corresponds to the plants state trajectory moving through the state space, resulting in new operating conditions.

We see that in the SIC approach the control law parameters are directly manipulated by the adaptation mechanism, whereas in the RST law the control law parameters are indirectly adjusted by first evaluating the MDPP algorithm. However, the MDPP algorithm uses the adaptation mechanism's estimation of process parameters. The former approach is typically referred to as direct adaptation or implicit self tuning, while the latter is called indirect adaptation or explicit self tuning. Alternatively, one may view indirect adaptation

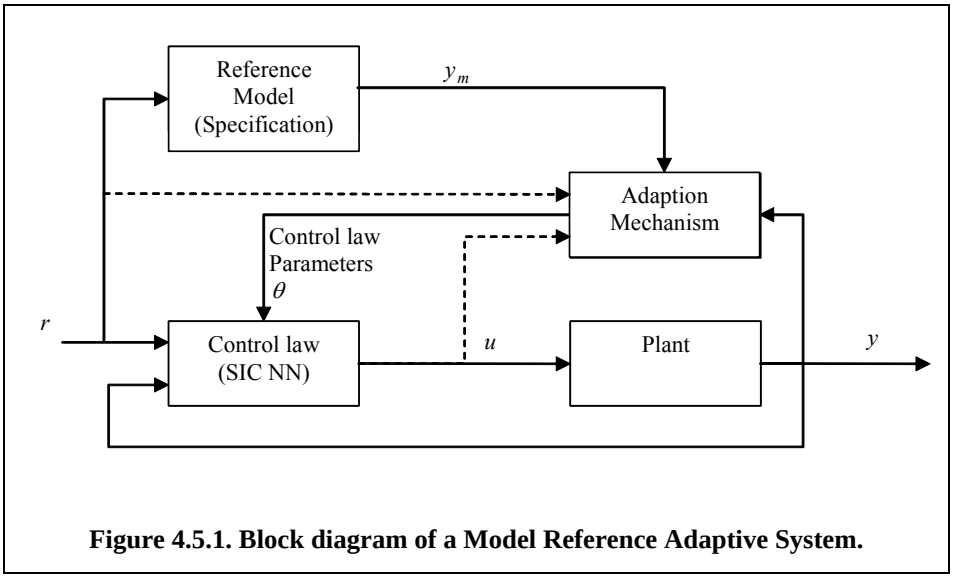


Figure 4.5.1. Block diagram of a Model Reference Adaptive System.

mechanisms as making explicit use of the estimated process parameters whereas the direct adaptation method does not.

4.5.2 INCORPORATING THE DESIGN SPECIFICATION

Implicit in the use of adaptive control is the desire to achieve some controller objective or performance. This is stated in the form of a control design specification. Therefore, the second key question to be asked is: "How is the design specification incorporated into the adaptation mechanism?"

In the SIC law an independent dynamic reference model was used to generate a reference signal which the plant must track. This reference model therefore represents the desired performance and provides a convenient means by which a specification may be injected into the adaptation mechanism. The approach results in what is called a Model Reference Adaptive System (MRAS). A block diagram depicting this approach is shown in Figure 4.5.1. Note, as the name implies, the SIC law is represented by a neural network placed in series with the plant under control. The dotted line associated with the control and demand signal paths into the adaptation mechanism imply that the method used to actually change the parameters may or may not use these signals.

During the design of the RST law, using the MDPP algorithm, the design specification was included in the algorithm by the selection of A_o, A_m, B_m and the factoring of B , the polynomial representing the process zeros. The adaptation mechanism's function is to provide the A and B polynomials. Generally this approach is called an indirect Self Tuning Regulator (STR). Another key configuration of the STR method is obtained if the control law design equations are incorporated into the adaptation mechanism, which now provides the *R, S* and *T* control law polynomials directly. The resulting system is referred to as a direct Self Tuning Regulator, but we do not address this further. When using neural networks to estimate non-linear plant parameters, this approach represents a specific non-linear adaptive control implementation, known as instantaneous linearization

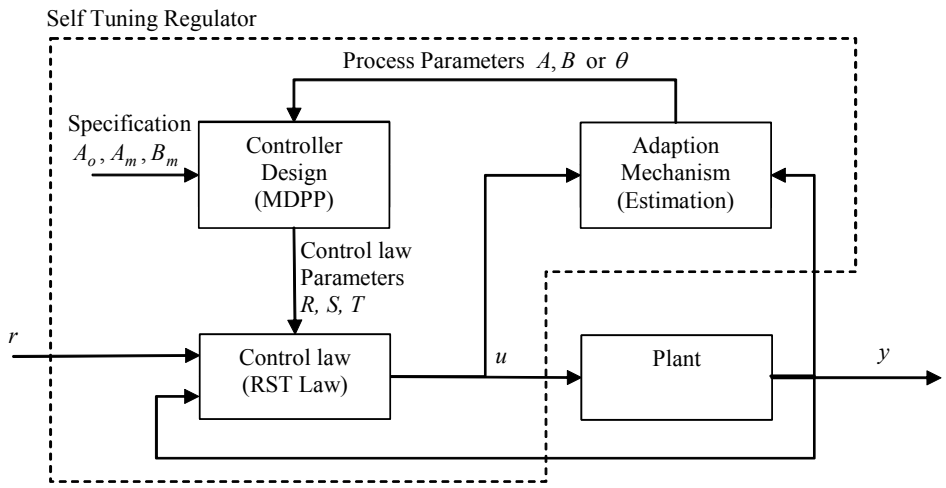


Figure 4.5.2. Block diagram of an indirect Self Tuning Regulator.

(Sørensen, 1994).

A block diagram of the indirect STR approach appears in Figure 4.5.2. An important observation is that the controller design treats the process parameters as if they are true, an assumption which is referred to as the certainty equivalence principle. Unfortunately this assumption only holds if the estimation process is persistently excited – a requirement that may be difficult to meet in a practical implementation.

Another shortcoming of the method is that if the linearized process parameters change rapidly, i.e. within the same time scale as the feedback signals, the resulting controller will not provide "optimal" control action. This occurs because the controller parameters are evaluated using a linearized plant model which, at the next sample update, is no longer a good approximation of the underlying process. This limits the applicability of the instantaneous linearization approach. The problem may be addressed by using generalised model predictive control where optimal future control actions are calculated at each sample time by minimising a cost function which weights the control actions and the predicted future outputs over some known time horizon.

4.5.3 CHANGING THE PARAMETERS

The final question to be resolved is: "How does the adaptation mechanism adjust the parameters?" Answering this question is really the essence of adaptive control systems design and analysis. Clearly, the overall system structure plays a key role in the mechanisation of the adaptation law. It is interesting to note however, that the MRAS approach described above is equivalent to a direct STR based on a MDPP design where all process zeros are cancelled. The reader is referred to §5.9 of Åström and Wittenmark (1995) for more details. Due to the adaptation mechanisms structural reliance, only the approaches used in this work will be discussed.

Mathematically the adaptation mechanism must evolve the adapted parameters $\boldsymbol{\theta}$ in time, that is:

$$\dot{\boldsymbol{\theta}} = \Omega(t)\,. \tag{4.5.1}$$

The structure of Ω thus defines the adaptation law. If we define a quadratic loss function that provides a cost associated with the error e between the desired output trajectory and the actual output trajectory, then driving the parameter vector in such a way as to minimize this cost would provide the required result. Therefore let:

$$J(\boldsymbol{\theta}) = \frac{1}{2}(y - y_m)^2 = \frac{1}{2}e^2 \tag{4.5.2}$$

then using a gradient descent approach to minimize (4.5.2) results in:

$$\begin{aligned} \dot{\boldsymbol{\theta}} = \Omega(t) &= -\gamma \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= -\gamma e \frac{\partial e}{\partial \boldsymbol{\theta}} \\ &= -\gamma e \frac{\partial y}{\partial u} \frac{\partial u(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \end{aligned} \tag{4.5.3}$$

or equivalently for a discrete time system:

$$\boldsymbol{\theta}[k] = \boldsymbol{\theta}[k-1] - \gamma' e[k] \frac{\partial y[k]}{\partial u[k,\boldsymbol{\theta}]} \frac{\partial u[k,\boldsymbol{\theta}]}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}[k-1]}. \tag{4.5.4}$$

Equation (4.5.3) is known as the MIT rule. The variable γ is called the adaptation gain, while the derivative $\partial e / \partial \boldsymbol{\theta}$ is known as the sensitivity derivative which, if the parameter changes are slow, may be evaluated assuming $\boldsymbol{\theta}$ is constant. Clearly, if the loss function (4.5.2) is changed, different adaptation laws will result. There are also enhancements to the basic law such as normalization, which removes the dependency of the gradient step size on the command signal. A slightly modified form of the law may also be derived using a Lyapunov function thus providing conditions under which the adaptation law is guaranteed to be stable. Åström and Wittenmark (1995) address all these details. However, in this work, only the basic idea behind the formulation above is followed.

When using a discrete time MRAS approach, the design specification is provided in the form of a reference model which generates a reference signal $\boldsymbol{y_m}[k]$. The objective is for the difference between $\boldsymbol{y_m}[k]$ and the closed loop plant output signal $\boldsymbol{y}(k,\boldsymbol{\theta})$, a function of the controller parameters, to be minimized. Therefore, letting $\varepsilon(k,\boldsymbol{\theta}) = \boldsymbol{y_m}[k] - \boldsymbol{y}(k,\boldsymbol{\theta})$, the adapted parameters should be adjusted in such a way that some loss (objective) function:

$$J(t,\boldsymbol{\theta}) = \sum_{k=1}^{t} \ell\big(\varepsilon(k,\boldsymbol{\theta})\big) \tag{4.5.5}$$

is minimized. This is identical to the problem addressed in section 3.3.1 and following the same logic presented there results in an equivalent optimisation problem formulation. The key difference is that the signal associated with the observed (reference) system output is now $\boldsymbol{y_m}[k]$. Also the network predicted system output, previously denoted by $\hat{\boldsymbol{y}}[k]$, is now the control signal $\boldsymbol{u}[k,\boldsymbol{\theta}]$, and is replaced by $\boldsymbol{y}(k,\boldsymbol{\theta})$, the series combination of the SIC law implementation and the plant under control. The adaptation mechanism problem may therefore be stated as:

$$\theta^*(t) = \arg\min_{\theta}\left(\frac{1}{2}\sum_{k=1}^{t}\beta(t,k)\big(\boldsymbol{y}_m[k] - \boldsymbol{y}(k,\boldsymbol{\theta})\big)^T \Lambda^{-1}\big(\boldsymbol{y}_m[k] - \boldsymbol{y}(k,\boldsymbol{\theta})\big)\right). \tag{4.5.6}$$

Having established this result the more sophisticated methods (as opposed to the simple gradient descent approach used in the MIT rule shown above) derived to solve the network learning rule may be brought to bear. There is however one caveat. All the algorithms discussed in section 3.3 go about solving equation (3.3.5) or equivalently (4.5.6) by differentiating the loss function (4.5.5) with respect to the parameter vector $\boldsymbol{\theta}$. This implies that in the derivation of the methods in section 3.3, what was previously the differentiation $\partial\hat{\boldsymbol{y}}[k]/\partial\boldsymbol{\theta}$ is now replaced by $\partial\boldsymbol{y}(k,\boldsymbol{\theta})/\partial\boldsymbol{\theta}$. We saw in section 3.3.3 that for all the network types $\partial\hat{\boldsymbol{y}}[k]/\partial\boldsymbol{\theta}$ is simply the regressor variable $\varphi^T[k]$. Therefore, using the chain rule, the differentiation of $\partial\boldsymbol{y}(k,\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ gives:

$$\frac{\partial\boldsymbol{y}(k,\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = \frac{\partial\boldsymbol{y}(k,\boldsymbol{\theta})}{\partial\boldsymbol{u}[k,\boldsymbol{\theta}]}\frac{\partial\boldsymbol{u}[k,\boldsymbol{\theta}]}{\partial\boldsymbol{\theta}},$$

but

$$\frac{\partial\boldsymbol{u}[k,\boldsymbol{\theta}]}{\partial\boldsymbol{\theta}} = \frac{\partial\hat{\boldsymbol{y}}[k]}{\partial\boldsymbol{\theta}} = \varphi^T[k].$$

Summarising, if we replace:

$$\begin{aligned}\hat{\boldsymbol{y}}[k]_{old} &\leftarrow \boldsymbol{y}(k,\boldsymbol{\theta})_{new}\\ \boldsymbol{y}[k]_{old} &\leftarrow \boldsymbol{y}_m[k]_{new}\\ \varphi[k]_{old} &\leftarrow \widetilde{\varphi}[k] = \varphi[k]_{old}\left(\frac{\partial \boldsymbol{y}(k,\boldsymbol{\theta})_{new}}{\partial \boldsymbol{u}[k,\boldsymbol{\theta}]_{new}}\right)^T\end{aligned} \tag{4.5.7}$$

then[17] the previously developed algorithms can be used. Unfortunately the final substitution in (4.5.7) is problematic for multi-output systems where an RBF or LMN network is used to form the SIC law. The new $\widetilde{\varphi}[k]$ becomes a matrix as opposed to a vector meaning that the efficiencies described in section 3.3.9.2 cannot be fully exploited. If Λ^{-1} is an identity matrix, a more efficient implementation can be achieved by directly differentiating (3.3.4) with respect to $\boldsymbol{\theta}$ and expressing the result as:

$$\frac{\partial J(t,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{t} \beta(t,k)\varphi[k]_{old}\left(\boldsymbol{y}[k]_{old} - \hat{\boldsymbol{y}}[k]_{old}\right).$$

Now performing the substitutions in (4.5.7) gives:

$$\frac{\partial J(t,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{t} \beta(t,k)\varphi[k]_{old}\left(\frac{\partial \boldsymbol{y}(k,\boldsymbol{\theta})_{new}}{\partial \boldsymbol{u}[k,\boldsymbol{\theta}]_{new}}\right)^T\left(\boldsymbol{y}_m[k]_{new} - \boldsymbol{y}(k,\boldsymbol{\theta})_{new}\right).$$

Therefore, substituting:

$$\left(\boldsymbol{y}[k]_{old} - \hat{\boldsymbol{y}}[k]_{old}\right) \leftarrow \left(\frac{\partial \boldsymbol{y}(k,\boldsymbol{\theta})_{new}}{\partial \boldsymbol{u}[k,\boldsymbol{\theta}]_{new}}\right)^T\left(\boldsymbol{y}_m[k]_{new} - \boldsymbol{y}(k,\boldsymbol{\theta})_{new}\right) \tag{4.5.8}$$

will result in the same update as performing the substitutions in (4.5.7). Now only the parameter update equations need be modified in the previously developed algorithms. The form of these equations is then the same as the discrete time version of the MIT rule shown in (4.5.4).

4.5.4 Using the Neural Network Jacobian Information

Until now, the reason for using neural network Jacobian information in neurocontrol has not been clearly evident. The focus has been mainly on where networks should be placed in the control system and how the training algorithms are affected. So why do we need to calculate the network Jacobian information?

Examination of equation (4.5.8) reveals that, for SIC law implementations some estimate of the plant input sensitivity, $\partial \boldsymbol{y}(k,\boldsymbol{\theta})/\partial \boldsymbol{u}[k,\boldsymbol{\theta}]$, is required. This can be approached in two ways. The first is to assume that the plant input sensitivity is estimated by some predefined known function, for example $\partial \boldsymbol{y}(k,\boldsymbol{\theta})/\partial \boldsymbol{u}[k,\boldsymbol{\theta}] = 1$ resulting in what is called direct sensitivity adaptation. The second approach is to model the plant behaviour using an ANN. If this is done online, the method will be referred to in the sequel as series inverse control using indirect sensitivity adaptation.

Recovering the plant input sensitivity information from a neural network model of a plant and constructing a neural network adaptation mechanism for a self tuning regulator system may be accomplished in precisely the

[17] The "*old*" subscript implies the symbols used in the algorithms of section 3.3 while the "*new*" subscript implies the symbols used in this section.

same way. Recall now the plant description expressed in equation (4.4.8). Shifting the time origin in this equation allows us to write:

$$y[k] = -a_1 y[k-1] - a_2 y[k-2] - \cdots - a_{n_a} y[k-n_a] + b_0 u[k-n_k] + \cdots + b_{n_b} u[k-n_k-n_b] \tag{4.5.9}$$

where the coefficients $a_1 \cdots a_{n_a}$ and $b_0 \cdots b_{n_b}$ are the time varying coefficients of the forward shift operator polynomials A and B. Stating (4.5.9) in matrix notation gives:

$$y[k] = \boldsymbol{y}(k,\boldsymbol{\theta}) = \boldsymbol{\theta}\ \boldsymbol{u}[k,\boldsymbol{\theta}] \tag{4.5.10}$$

where $\boldsymbol{u}[k,\boldsymbol{\theta}] = \begin{bmatrix} u[k-n_k] & u[k-n_k-1] & \cdots & u[k-n_k-n_b] \end{bmatrix}^T$ and $\boldsymbol{\theta} = [-a_1, -a_2, \cdots, -a_{n_a}, b_0, b_1, \cdots b_{n_b}]$. Clearly, $\partial \boldsymbol{y}(k,\boldsymbol{\theta}) / \partial \boldsymbol{u}[k,\boldsymbol{\theta}]$ is equivalent to differentiating (4.5.10) at any point in time and results in the vector $\boldsymbol{\theta}$ containing the (sign corrected) polynomial coefficients.

But, (4.5.10) is identically the FNN problem described in equation (2.4.4) with network output $\hat{\boldsymbol{y}}[k] = \boldsymbol{y}(k,\boldsymbol{\theta})$ and input $\boldsymbol{x}[k] = \boldsymbol{u}[k,\boldsymbol{\theta}]$, where plant and identification network inputs are now the same variable. Accordingly, an ANN which *correctly* describes the plant represents equation (4.5.9) and calculating the Jacobian of said network at any point in time is equivalent to differentiating (4.5.9). That is $\partial \hat{\boldsymbol{y}}[k] / \partial \boldsymbol{x}[k] = \partial \boldsymbol{y}(k,\boldsymbol{\theta}) / \partial \boldsymbol{u}[k,\boldsymbol{\theta}]$. Therefore, using the network Jacobian algorithms described in section 3.4, the required parameters for the STR control law synthesis algorithm (in this case MDPP) or the plant input sensitivity information for indirect sensitivity adaptation using the SIC law, can be calculated. Without the network Jacobian algorithms these control techniques could not be used. In the sequel, to avoid confusion with the actual plant input or state variables, the vector designated by symbols $\boldsymbol{x}[k]$ and $\boldsymbol{u}[k,\boldsymbol{\theta}]$ will be referred to as the information vector, designated by the symbol $\boldsymbol{\psi}[k]$.

4.6 CONCLUSIONS

This chapter introduced the field of neurocontrol. A five step design methodology was presented and the first three steps, namely plant description, control law formulation and adaptation mechanisms where elaborated upon. The omitted steps, stability and parameter convergence, were considered beyond the scope of this work, however, we reiterate that these steps are crucial to the successful implementation of real-word controllers.

The section on plant description derived the conditions necessary to express a continuously differentiable discrete time non-linear state space plant description in an I/O form. This is essentially the inverse of the observer problem discussed in many control texts. (See for example §4.4 of Åström and Wittenmark, 1997.) The derivation used the inverse function theorem to show that an I/O description amenable to mapping by a general neural network formulation was possible for some local region about a chosen operating point. The required condition was that the determinant of the matrix formed by the differential of the output with respect to the plant states, evaluated at the selected operating point, was non zero.

Having established the conditions for I/O plant description the next section addressed the two different control law formulations used in the simulations presented in the next chapter; Series Inverse Control (SIC) and Minimum Degree Pole Placement Design (MDPP).

The first main result indicated that, for a MIMO system with the same number of outputs as inputs, the SIC paradigm could be used for some local region about an operating point if the determinant of the Jacobian plant mapping evaluated at the operating point was non zero. It was also shown how reference model dynamics could be included in this control law formulation. It is important to recognize however, that the existence of these results makes no statement about the suitability of the law for practical implementation. In particular, control of non-minimum phase plants would require potentially unstable cancellation of the plant zeros by the series inverse control law dynamics. Furthermore, system disturbances of the actual plant output are only indirectly accounted for through the dynamic relationship between the demand and desired plant output, potentially making disturbance rejection problematic. These and other problems are not directly addressed in this dissertation.

Next the MDPP design algorithm was derived using shift operator polynomial techniques. It was shown that this approach is well suited to the I/O representation used in the neural network formulation. The algorithm is purely algebraic and can be implemented with minimal computational effort by solving the Diophantine equation. The resulting controller is a two degree of freedom controller which may, or may not, be configured to cancel the process zeros depending on the stability implications resulting from such a cancellation. Although not explicitly discussed, the issue of disturbances can also be addressed by using stochastic methods to derive the RST polynomials. One such method which was implemented in the software blockset was the minimum variance algorithm discussed in §4.2 of Åström and Wittenmark (1995). Regrettably, the polynomial formulations shown are valid only for SISO plants. Extension to MIMO systems is possible by treating the time delayed inputs and outputs as a state vector. State space system matrices can then be formed in an observer canonical form and MIMO design techniques can be brought to bear in the formulation of a control law. Unfortunately, this formulation of the system state can be inefficient leading to growing matrix dimensions, online matrix inversions, and potential numerical problems or poorly conditioned systems.

In the lastly major section the adaptation mechanisms used for dynamically updating the control laws were described. This section answered three key questions. Firstly, the parameters to be adjusted were defined for each of the control law formulations. The differing control laws and their associated parameters lead to the important concepts of direct and indirect adaptation, also known as implicit and explicit self tuning respectively.

Secondly it was shown how the related, but conceptually distinct, classes of MRAS and STR systems resulted when using differing methods to incorporate the design specification into the adaptation mechanism. The certainty equivalence principle was highlighted and the idea of instantaneous linearization of nonlinear plants using neural networks was introduced.

Finally a mathematical formulation of the adaptation mechanism problem was given. It was shown that, with the appropriate substitutions, the algorithms of section 3.3 could be used to solve the adaptation mechanism problem for all of the control law formulations presented. A key observation of this section was that the Jacobian of a FNN which had correctly identified the plant described by (4.5.9) could be used to acquire the parameters required by both SIC indirect sensitivity adaptation and instantaneous linearization using a STR approach. The correct convergence to the plant parameters of equation (4.5.9) is of paramount importance. If this is not

achieved the resulting control law is invalid, or at best suboptimal. Therefore, ensuring that the correct parameters are identified is imperative for the successful application of the technique. Another drawback to the approach is that knowledge about the order and time delay associated with the plant is assumed when formulating an STR design specification. A similar problem exists when trying to identify the plant input sensitivity for use in an MRAS system.

CHAPTER 5

SIMULATIONS

5.1 INTRODUCTION

In this chapter, we will present a number of simulations of various systems to illustrate the concepts and issues described in previous chapters. All of the simulations were generated in Matlab/Simulink using a class library and block set that was specially created for this purpose.

The use of such 'high level' simulation programs is sometimes criticised as being slow and sub-optimal when compared to customised programs written using compiler based languages such as *C* or *C++*. This is frequently a valid criticism, however, custom programs of the complexity required to simulate neurocontrol systems are extremely time consuming to create. Even with a carefully executed systems software design approach the resulting programs are often not transportable or reusable in other continuing research efforts. For Matlab m-code files and Simulink S-function, experience has shown that provided:

- the code is highly vectorized;
- all function calls made within m-code or S-function files are to low level internal Matlab functions;
- careful attention is paid to pre-allocating memory for all frequently used variables;

then the speed of the resulting program is comparable to compiled code. Memory requirements are usually larger for 'high level' programs. This, however, is generally not a problem, or one that is easily overcome, except when the code is to be used in an embedded system where cost and/or space constraints are large factors in the engineering design. The cost/space constraint may also be considerably reduced by the implementation of the networks in an "on-line" fashion as opposed to the batch mode processing most frequently used.

In spite of these observations, it is important to note that implementing neural networks on a serial processing system is computationally intense due to the parallel nature of the networks. For this reason, considerable computing power is required no matter what type of programming language is used.

Due to the nature of the algorithms developed in the previous chapter it is possible to satisfy all of the requirements listed above, thus allowing the development of highly efficient m-code S-function files. The result is a set of tools that are easy to use, highly flexible, and transportable between different computing systems.

5.2 INVERSE NEURAL CONTROL

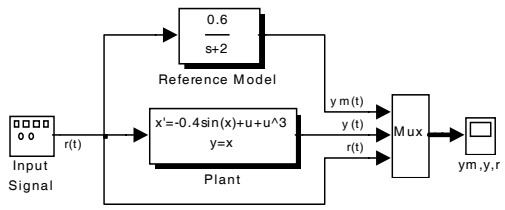
5.2.1 SYSTEM DESCRIPTION

Experiment 1 is an exact recreation of a controller presented in Van Breeman (1997) with one major difference; the neural network controller is a local model network as opposed to a RBF network. The experiment explores the control of a first order non-linear continuous time SISO system using various inverse neural network controller configurations. The system under consideration is given by:

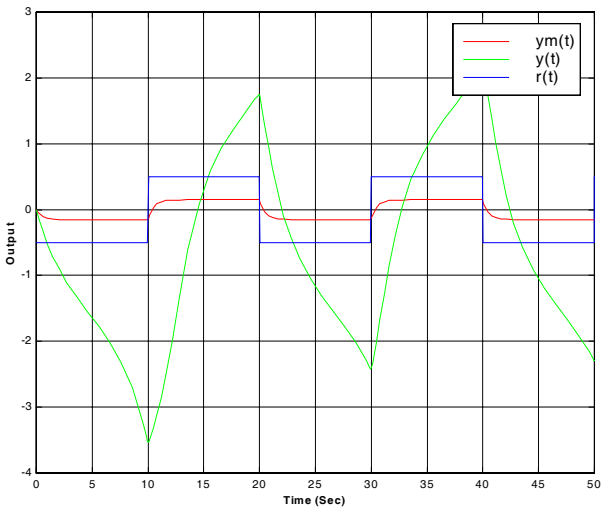
$$\begin{aligned} \dot{x} &= -0.4\sin(x) + u + u^3 \\ y &= x\,. \end{aligned} \tag{5.2.1}$$

The desired response of the closed loop system is described by the linear first order reference model equation:

$$\begin{aligned} \dot{x}_m &= -2x_m + 0.6r \\ y_m &= x_m\,. \end{aligned} \tag{5.2.2}$$



(a) Simulink Model



(b) Open Loop Response

Figure 5.2.1. Non-Linear First Order Plant - Open Loop System Model and its Response.

Depicted in Figure 5.2.1 is the Simulink model (ctlexp1a) corresponding to equations (5.2.1) and (5.2.2) together with the open loop response curves to a square wave input. Clearly, a controller is required to achieve model following. Let us assume that we would like to achieve the following tracking requirements:

- Absolute differential error of less than 5%.
- Relative error tolerance of less than 1%.
- Absolute error tolerance of less than 1%.

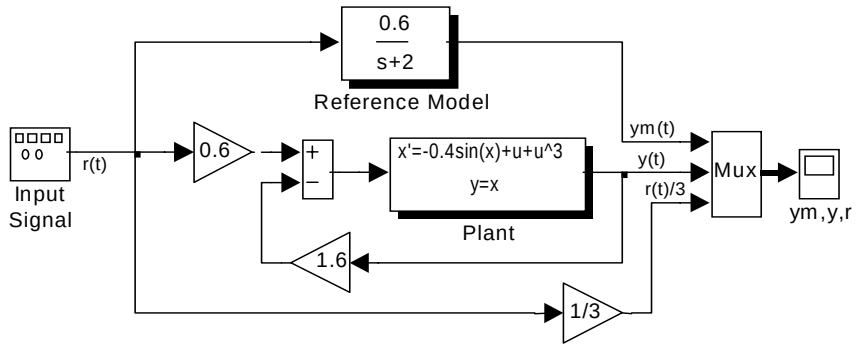
5.2.2 LINEAR STATE FEEDBACK CONTROL

In order to establish a baseline for controller results we begin by applying a linear state feedback controller to the problem. If the plant were unknown, it would obviously not be possible to perform this step. However, when plant equations are known an attempt should always be made at meeting the requirements with simpler and well understood linear control design methods.

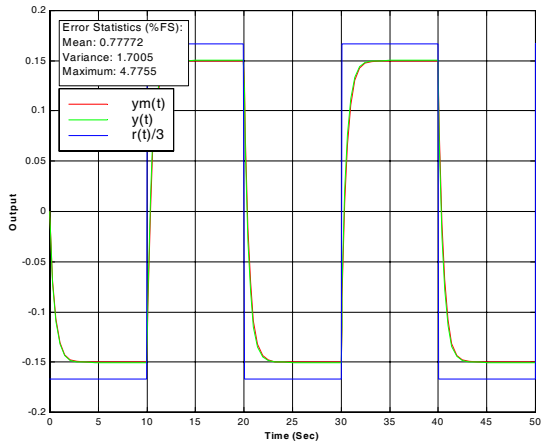
The design is performed about the operating point $(\bar{x},\bar{u}) = (0,0)$ using a pole placement feedback law:

$$\delta\dot{x} = -0.4\cos(\bar{x})\delta x + (1 + 3\bar{u}^2)\delta u = -0.4\delta x + \delta u$$
$$\therefore \delta u = -1.6\delta x + 0.6\delta r \,.$$

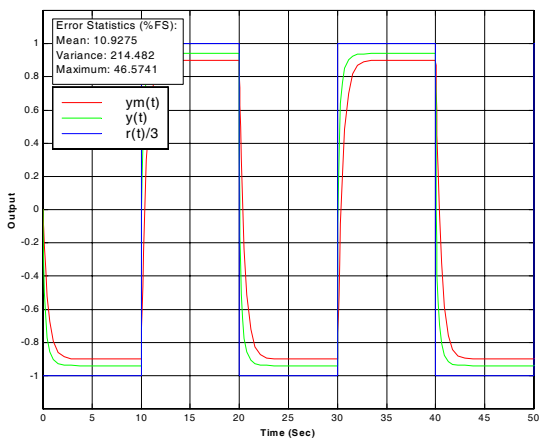
The resulting model (ctlexp1b) and the closed loop system response are shown in Figure 5.2.2 below.



(a) State Feedback Controller System Model



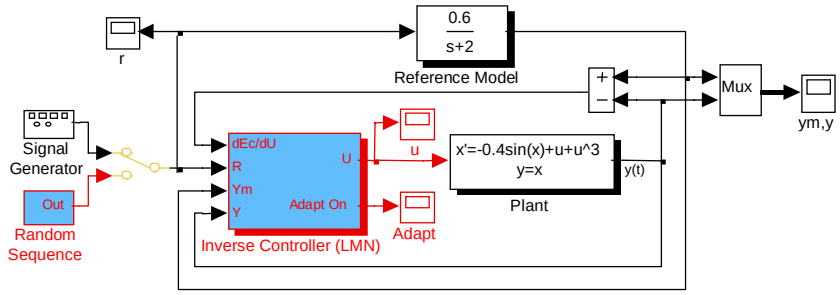
(b) Response at r=0.5



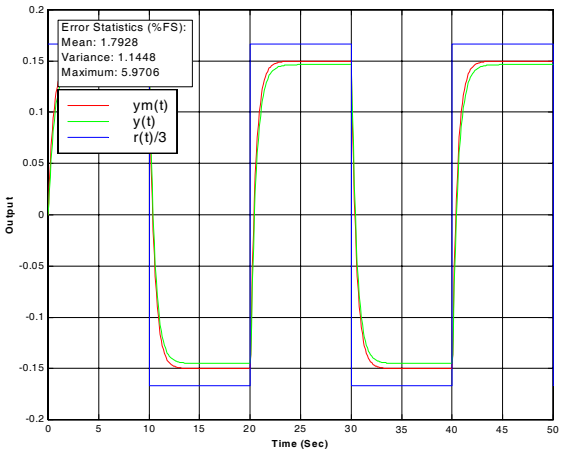
(c) Response at r=3.0

Figure 5.2.2. State Feedback System Model and its Closed Loop Responses.

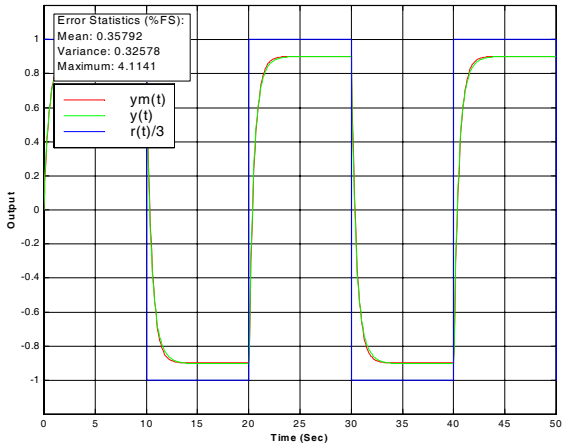
In plot (b) we see that, for small perturbations from the operating point, the linear controller performs the task well but, as expected, plot (c) shows the tracking error increases substantially when the system is perturbed by larger amounts.



(a) Inverse Neural Controller System Model



(b) Response at r=0.5



(c) Response at r=3.0

Figure 5.2.3. Inverse Neural Controller System Model and its Closed Loop Responses.

5.2.3 SERIES INVERSE CONTROL USING DIRECT ADAPTATION

An attempt is made to rectify the tracking error displayed in section 5.2.2 by implementing a series inverse local model neural network controller using direct adaptation. First, it must be verified that the inverse function exists about the operating point. Under closed loop control, the plant output must equal the reference model output, that is, using the notation of section 4.4.1, $H(x,u) = S(x_m, r)$, therefore:

$$\tilde{G}(x,r) = u = H^{-1}(x, S(x_m, r)).$$

The existence of the inverse of the plant mapping $H(x,u)$ can be verified using condition (4.3.10):

$$\det\left[\frac{\partial H(x,u)}{\partial u}\right]_{(\bar{x},\bar{u})} = 1 + 3\bar{u}^2 \neq 0.$$

As the reference model mapping $S(x_m, r)$ is smooth and the inverse of $H(x,u)$ exists, and is smooth, for all values of u over the region of interest, we know that an inverse controller solution exists and that such solution may be approximated by a LMN.

To train a controller the Simulink model (ctlexp1d) shown in Figure 5.2.3 was run for a 2000-second period. The input was a uniformly distributed random input sequence between +3.5 and -3.5, which had a 3%

probability of a value change at each sample. The inverse controller used to perform the task was setup in the manner shown in Table 5.2.1 below:

Inverse Controller Setup Parameters			
General Network Parameters			
Network Type:	Local Model Network		
Initial Weight Range (Min / Max):	0 / 0		
Validity Activation Threshold (%):	2		
Sample Time (Sec):	0.1		
Network Inputs (In information vector order)			
Name	**Range (Min/Max)**	**Number of regions**	**Overlap (%)**
r[k]	-4 / 4	8	30
y[k]	-1.5 / 1.5	2	60
u[k-1][18]	-1 / 1	1	100
Error Tolerances (%)			
Relative Differential	**Absolute Differential**	**Relative Value**	**Absolute Value**
1	0.5	0.1	0.1
Adaptation Method Parameters			
Adaptation Algorithm:	Exponential Forgetting Factor		
Initial Covariance Diagonal:	0.1		
Forgetting Factor:	0.998		

Table 5.2.1. Inverse Controller Setup Parameters.

The specified input signal was used so that the controller could be exposed to all operating regions of the plant in question. Once the training process was complete the final weights were saved, the input was switched to a square wave and the adaptation mechanism was switched off. The resulting closed loop response of the system is shown in Figure 5.2.3.

The results for *both* input signals now meet the required specification[19]. It is important to note that there was no adaptation of the controller network between different input signal amplitudes. This would indicate that the controller has learnt a true non-linear mapping to perform the control actions. In this simple example, it is possible to compare this mapping to that used by the linear state feedback controller. The comparison is shown in Figure 5.2.4. In the figure, the red mesh indicates the linear mapping while the solid coloured surface is the inverse controller mapping. The black points are data values used to generate the inverse controller surface.

[18] The past control value is included in the information vector purely because of a software limitation stemming from the desire to deal with the more general case where it is required. As it is assigned only one region, which is 100% activated at all times, it plays no meaningful role in the non-linear calculations of the validity functions. It does however add a redundant weight to the interpolated local models, which has to be trained to a minimal value.

[19] The reader may be curious as to why the error statistics shown in the plot indicate larger values than those called out by the specification. This can be explained by the earlier discussion on differential error. The error statistics shown are calculated by subtracting the actual value of the plant output from the desired value generated by the reference model and then scaling the results to represent a percentage of the output signal's range (full scale). The large errors generated when the signal is experiencing rapid changes in value are included in these calculations; however, the error due to the differential of the signal being different is within the required tolerance. This can be verified by running the simulation with the adaptation error tolerances set to the values in the specification and observing that the adaptation mechanism is not triggered.

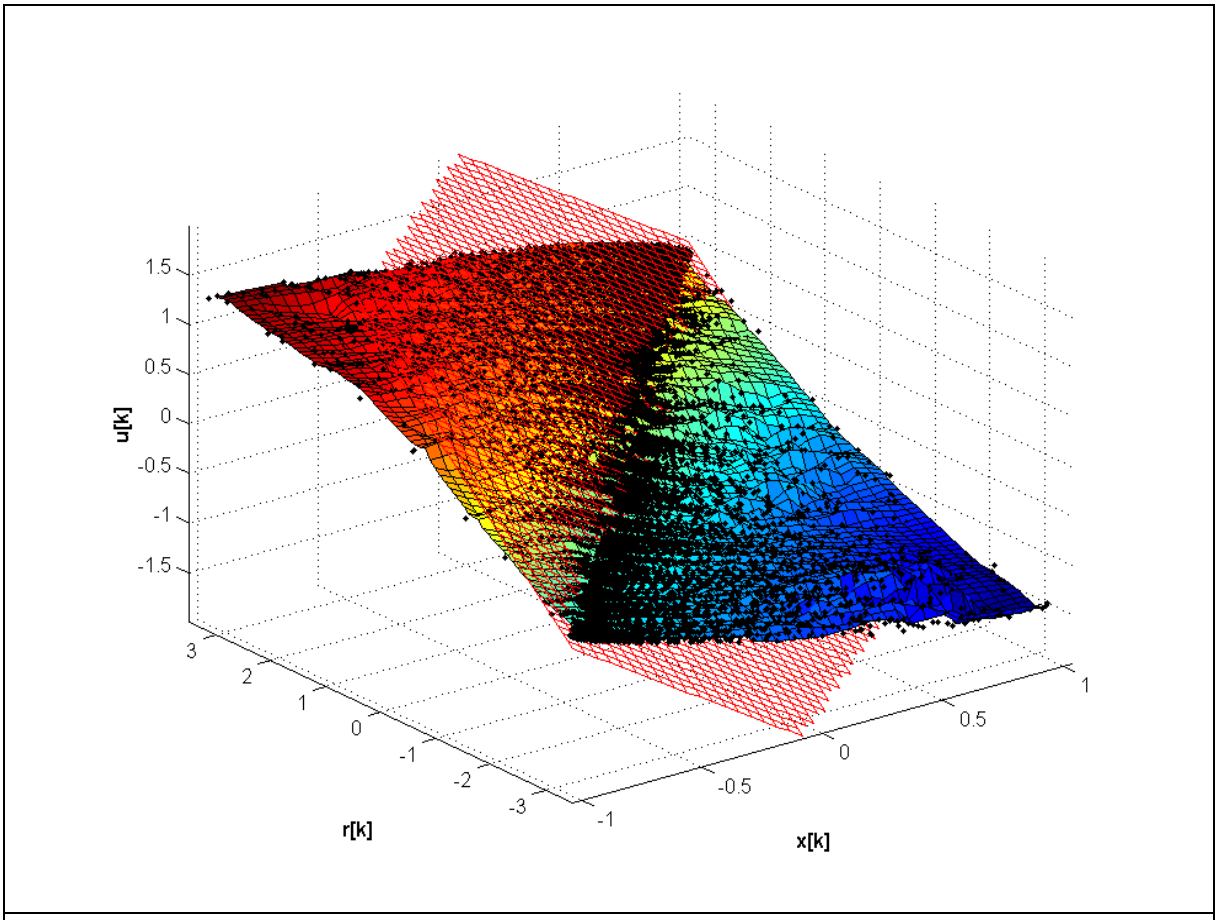
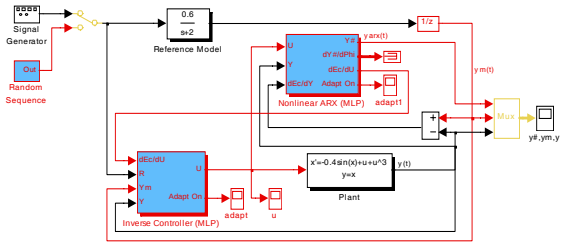


Figure 5.2.4. Comparison of Inverse and Linear State Feedback Controller Mappings.

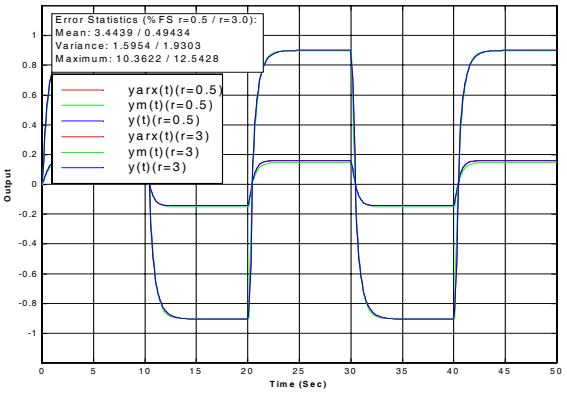
This plot illustrates two salient features. Firstly, it clearly shows how the two controllers closely approximate one another at the linear controller's designed operating point. Secondly, and more importantly, it demonstrates the need for persistently exciting signals when trying to identify a non-linear system. (In this case, the inverse of the system.) We notice that in those areas where there are many data points the mapping surface is reasonably smooth. However, where fewer data points exist, the surface becomes more irregular and does not always seem to follow the general curve trend. This is unfortunately a problem associated with all parametric techniques attempting to identify non-linear systems.

5.2.4 SERIES INVERSE CONTROL USING INDIRECT SENSITIVITY ADAPTATION

The last simulation (ctlexp2a) conducted with the system described in equation (5.2.1) demonstrates the use of indirect sensitivity adaptation to train an MLP inverse controller. Although MLP networks are not the main topic of this dissertation, the experiment indicates how, with the algorithms previously described, different types of networks may be substituted for one another while maintaining the same controller structure. This allows the system designer to use the network with the optimum characteristics for the problem at hand without concern for overall structural issues. The Simulink block set developed allows this substitution to occur with minimal effort. The model shown in Figure 5.2.5 could have used LMN networks, permitting a linear analysis of the final trained system. (The next section will demonstrate the use of LMN networks for Jacobian identification.)



(a) Indirect Sensitivity Adaptation System Model



(b) Response at r=0.5 and r=3.0

Figure 5.2.5. Indirect MLP Inverse Controller and its Closed Loop Responses.

The complete system takes the form shown in Figure 5.2.5. Using the indirect sensitivity adaptation method the plant is identified by a non-linear ARX model constructed using a MLP network model. This model is used to provide an estimate of the plant input sensitivity. That is:

$$\begin{aligned}\frac{dE_c}{du} &= \frac{dE_c}{dy}\frac{dy}{du} \\ &= (y_m - y)\frac{dy}{du} \\ &\approx (y_m - y)\frac{d\hat{y}}{d\boldsymbol{\psi}}\frac{d\boldsymbol{\psi}}{du}\end{aligned} \qquad (5.2.3)$$

where dy/du is the plant input sensitivity, and $d\hat{y}/d\boldsymbol{\psi}$ is the network Jacobian of the underlying MLP used in the non-linear ARX model. The known term $d\boldsymbol{\psi}/du$ is used to extract the current input from the information vector and, for this example, is simply unity. The product of $d\hat{y}/d\boldsymbol{\psi}$ and $d\boldsymbol{\psi}/du$ is therefore the estimate of the plant sensitivity using the MLP Jacobian information. Again, the network Jacobian algorithm described in section 3.4.1 is required to determine this value. Clearly, if we wished to use a LMN network instead of a MLP network to perform this function the algorithm derived in section 3.4.3 would have to be used. The reader is referred to sections 4.4.1 and 4.5 for detailed discussions of the control technique.

The training and simulation were performed in exactly the same manner as the previous section[20], providing the results presented in plot (b) of the figure. Although the identification step is not necessary with this particular plant, the plot clearly illustrates that the technique provides satisfactory results.

5.3 CONTROL USING INSTANTANEOUS LINEARIZATION

5.3.1 CONTROLLING A NON-LINEAR MASS-SPRING-DAMPER SYSTEM

In this section, we demonstrate how the ability to efficiently calculate the LMN Jacobian allows us to directly substitute for MLP networks in a non-linear control technique known as instantaneous linearization.

[20] Exact setup details for each of the networks can be viewed in the file ctlexp2a.mdl.

The plant under consideration may be described by the following general differential equation:

$$a\ddot{y} + b\dot{y} + cy + dy^{\gamma} = u \tag{5.3.1}$$

where a, b, c, d and γ are constants that define the behaviour. Physically this corresponds to a mass-spring-damper system where the spring stiffness is related to its extension by the term $(c + dy^{\gamma-1})$. The input to the system $u(t)$ is a compression or extension force and the output $y(t)$ is the degree of spring extension measured from an initial operating point. To illustrate various points made in previous chapters, we transform equation (5.3.1) into a discrete time state space and discrete time I/O model respectively.

Let:

$$x_1(t) = y(t)$$
$$x_2(t) = \dot{y}(t)$$

then:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -(b/a)x_2 - a^{-1}(c + dx_1^{\gamma-1})x_1 + u \end{bmatrix} \tag{5.3.2}$$
$$y = x_1 .$$

Equation (5.3.2) represents the continuous time state space model of the system. To convert it to a discrete time system we recall that:

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \int_t^{t+\Delta t} \boldsymbol{x}(t)dt .$$

Approximating with a Taylor series expansion:

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \dot{\boldsymbol{x}}(t)\Delta t + \ddot{\boldsymbol{x}}(t)\frac{\Delta t^2}{2!} + \cdots + \boldsymbol{x}^{(n-1)}(t)\frac{\Delta t^{(n-1)}}{(n-1)!} + \cdots .$$

If Δt, the sampling interval, is chosen small enough we can truncate the series after the second term. Substituting the time value by a time index *k* and assuming a unit sample time[21], equation (5.3.2) may be approximated by the following discrete time non-linear state space model:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \end{bmatrix} = \begin{bmatrix} x_1[k] + x_2[k]\Delta t \\ (1 - \Delta t b/a)x_2[k] - a^{-1}\Delta t(c + dx_1^{\gamma-1}[k])x_1[k] + \Delta t\, u[k] \end{bmatrix} \tag{5.3.3}$$
$$y[k] = x_1[k] .$$

Now we test for I/O model existence by ensuring that:

$$\det \left[\begin{matrix} \dfrac{\partial y[k]}{\partial x_1[k]} & \dfrac{\partial y[k]}{\partial x_2[k]} \\ \dfrac{\partial y[k+1]}{\partial x_1[k]} & \dfrac{\partial y[k+1]}{\partial x_2[k]} \end{matrix} \right]_{(\bar{x},\bar{u})} \neq 0 .$$

[21] As the model is also dependent on the sample interval the Δt notation will be retained in the terms having a direct dependence on this value.

Performing the required differentiation gives:

$$\det\begin{bmatrix}1 & 0\\ 1 & \Delta t\end{bmatrix}\Bigg|_{(\bar{x},\bar{u})} = \Delta t \neq 0 .$$

The I/O model therefore exists for all positive real values of Δt and is independent of the operating point. Therefore, none of the assumptions made above are violated. We obtain the actual I/O model by a combination of time index shifts and repeated substitutions from equation (5.3.3) thus:

$$y[k+2] = x_1[k+1] + x_2[k+1]\Delta t$$

where:

$$x_2[k+1]\Delta t = \Delta t\left[(1-\Delta t b/a)x_2[k] - a^{-1}\Delta t(c + dx_1^{\gamma-1}[k])x_1[k] + \Delta t\, u[k]\right] .$$

Substituting for x_1 gives:

$$y[k+2] = y[k+1] + (1-\Delta t b/a)x_2[k]\Delta t - a^{-1}\Delta t^2(c + dy^{\gamma-1}[k])y[k] + \Delta t^2\, u[k]$$

but:

$$x_2[k]\Delta t = y[k+1] - y[k] .$$

Therefore:

$$y[k+2] = y[k+1] + (1-\Delta t b/a)(y[k+1] - y[k]) - a^{-1}\Delta t^2(c + dy^{\gamma-1}[k])y[k] + \Delta t^2\, u[k] .$$

After some basic algebraic manipulation and a time index shift, we get the final I/O model:

$$\begin{aligned} y[k] &= (2 - \Delta t b/a)y[k-1] \\ &\quad + a^{-1}(\Delta t b - a - \Delta t^2 c - \Delta t^2 d y^{\gamma-1}[k-2])y[k-2] \\ &\quad + \Delta t^2\, u[k-2] . \end{aligned} \tag{5.3.4}$$

It is also convenient for us to obtain the derivatives of the system output with respect to the information vector, that is $dy[k]/d\psi[k]$:

$$\begin{aligned} \frac{\partial y[k]}{\partial y[k-1]} &= (2 - \Delta t b/a) \\ \frac{\partial y[k]}{\partial y[k-2]} &= a^{-1}(\Delta t b - a - \Delta t^2 c - \Delta t^2(\gamma-1)d y^{\gamma-1}[k-2]) \\ \frac{\partial y[k]}{\partial u[k-1]} &= 0 \\ \frac{\partial y[k]}{\partial u[k-2]} &= \Delta t^2 . \end{aligned} \tag{5.3.5}$$

The following parameters were selected for the model in equation (5.3.4):

- Sample time $\Delta t = 0.2$ seconds (i.e. Five samples per second.)
- Mass constant a=1.0 kg.
- Friction Constant b=1.0 kg/s.
- Spring constants c, d and γ equal to 1, 20 and 3 respectively.

Equations (5.3.4) and (5.3.5) may therefore be reduced to:

$$y[k] = 1.8y[k-1] - (0.84 + 0.8y^2[k-2])y[k-2] + 0.04\,u[k-2] \tag{5.3.6}$$

and:

$$\begin{aligned}
\frac{\partial y[k]}{\partial y[k-1]} &= 1.8 \\
\frac{\partial y[k]}{\partial y[k-2]} &= -0.84 - 1.6y^2[k-2] \\
\frac{\partial y[k]}{\partial u[k-1]} &= 0 \\
\frac{\partial y[k]}{\partial u[k-2]} &= 0.04\,.
\end{aligned} \tag{5.3.7}$$

Figure 5.3.1 shows the open loop response of the system to zero mean square wave with a 20 second period and an amplitude of 0.6N. Clearly, the system is lightly damped, apparently stable for the given input signal, and with non-linearity that is evidenced by the signal having different responses for positive and negative inputs.

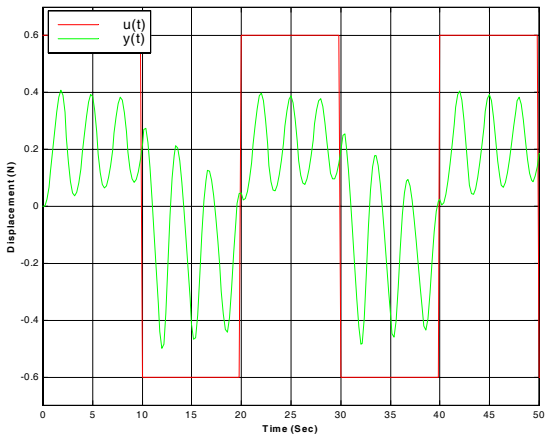


Figure 5.3.1. Open Loop Response of a Non-Linear Mass-Spring-Damper.

The first step in applying instantaneous linearization control is to obtain an identification model of the plant. To do this we set up an experiment that simulates the plant in an open loop manner, connect a non-linear ARX model in parallel with it, and train the LMN network in the ARX model. The Simulink model (ctrlexp3a) and parameters used are shown in Figure 5.3.2 and Table 5.3.1 respectively. The network structure chosen is a result of the information contained in equation (5.3.4). From this equation we know the number of time lags necessary for each information vector element and that the plant non-linearity is dependent on $y[k\text{-}2]$ only. It is therefore not necessary to have multiple regions for any of the other inputs to the information vector. The random signal input is used in an attempt to persistently excite the plant.

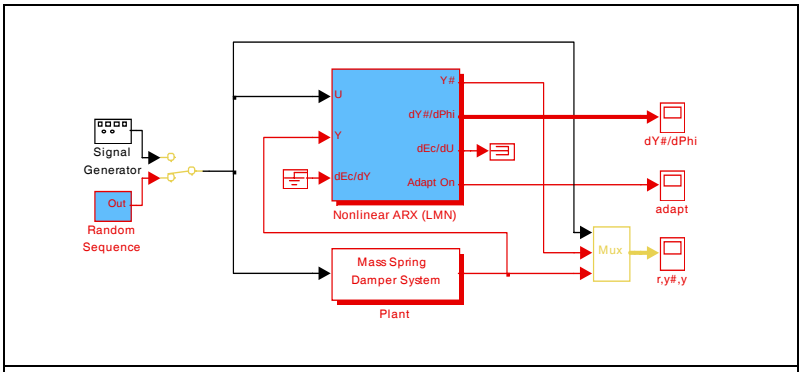


Figure 5.3.2. Identification System Model for Non-Linear Mass-Spring-Damper.

System Identification Setup Parameters			
General Network Parameters			
Network Type:	Local Model Network		
Initial Weight Range (Min / Max):	0 / 0		
Validity Activation Threshold (%):	2		
Sample Time (Sec):	0.2		
Network Inputs (In information vector order)			
Name	**Range (Min/Max)**	**Number of regions**	**Overlap (%)**
y[k-1]	-1 / 1	1	100
y[k-2]	-1 / 1	5	40
u[k-1]	-5 / 5	1	100
u[k-2]	-5 / 5	1	100
Error Tolerances (%)			
Relative Differential	**Absolute Differential**	**Relative Value**	**Absolute Value**
10	0.05	0.01	0.01
Adaptation Method Parameters			
Adaptation Algorithm:	Exponential Forgetting Factor		
Initial Covariance Diagonal:	10		
Forgetting Factor:	0.995		
Random Signal Parameters			
Minimum / Maximum	-0.45 / 0.45		
Probability of value change (%)	15		
Distribution	Uniform		
Sample Period	0.2		

Table 5.3.1. Non-Linear Mass-Spring-Damper Identification Setup Parameters.

The simulation length was set for a period of 2000 seconds, the adaptation was enabled, and the model allowed to process. Upon completion the weights were saved, the adaptation was disabled, and the simulation was run with the new weights and a different input to verify the accuracy of the identified model. Generally, this would

be done by comparing the outputs of the network model to the plant output. However, in this case we know, from equation (5.3.7), the analytic expressions for the coefficients that we wish to identify.

The identified coefficients, namely the differentials of the output with respect to the information vector, can be extracted from the identified network model (see section 4.5.4) by determining the network Jacobian using the algorithm derived in section 3.4.3. This was done with the system shown in Figure 5.3.2, using the same input square wave as in the open loop response test shown in Figure 5.3.1. The results are plotted in Figure 5.3.3.

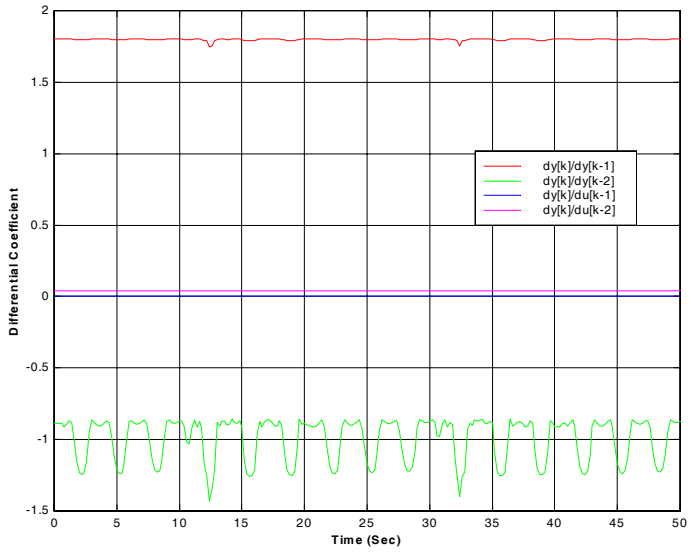


Figure 5.3.3. Polynomial Coefficients of the Non-Linear Mass-Spring-Damper.

If we compare the values in the plot to those resulting from the analytic solution of equation (5.3.7), we see that the identification model has performed an excellent job of estimating the correct values. Recall that these values are none other than the coefficients of the polynomials describing the discrete time transfer function of the plant at each instant in time:

$$y[k] \approx \hat{y}[k] = M(\boldsymbol{\psi}[k]) = \frac{\boldsymbol{B}(z)}{\boldsymbol{A}(z)} u[k - n_k]$$

$$\boldsymbol{A}(z) = 1 + a_1 z + \cdots + a_{n_a} z^{n_a}$$

$$\boldsymbol{B}(z) = b_0 + b_1 z + \cdots + b_{n_b} z^{n_b}$$

$$a_i = -\left.\frac{\partial \hat{y}[k])}{\partial y[k-i]}\right|_{k,\forall i \in [1 \cdots n_a]} \quad ; \quad b_i = \left.\frac{\partial \hat{y}[k])}{\partial u[k-n_k-i]}\right|_{k,\forall i \in [0 \cdots n_b]} .$$

By determining the roots of these polynomials for each instant of time it is possible, for a given input signal, to plot the pole zero migration resulting from the system non-linearity. If the input signal excites all the regions of interest then an understanding of the plant stability characteristics is possible. Such a plot was generated for the system under investigation using the previously trained identification model and square wave input. The results are shown in Figure 5.3.4. It now becomes evident that although the plant remained stable during the open loop simulations, there are certain regions in which instability may occur.

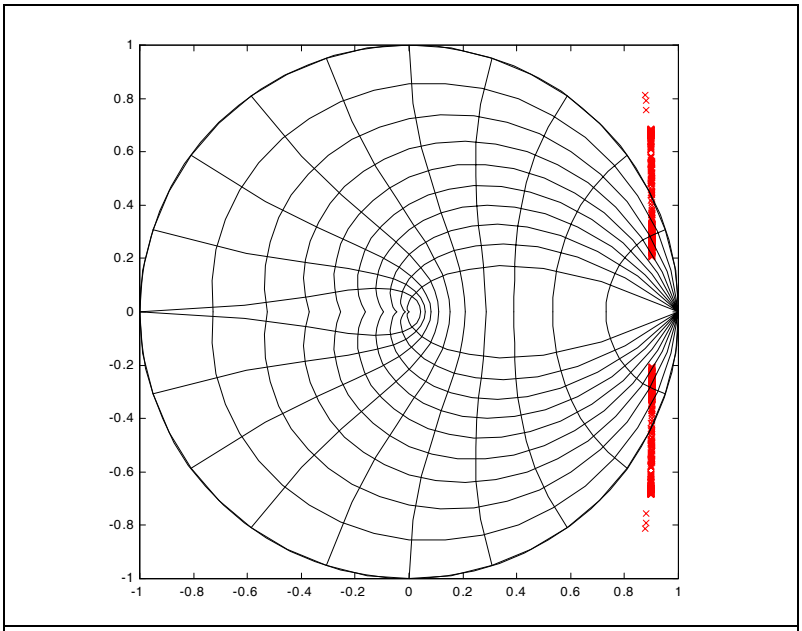


Figure 5.3.4. Pole Zero Plot for Mass-Spring-Damper System.

The final step is to use these polynomials in the design of a linear controller, which is executed at each sample. This was done by creating a Simulink block that implements the controller (sometimes referred to as an RST controller) shown in equation (5.3.8):

$$u[k] = \frac{1}{\boldsymbol{R}(z)}(\boldsymbol{T}(z)r[k] - \boldsymbol{S}(z)y[k])\,. \qquad (5.3.8)$$

The coefficients to the controller polynomials are obtained by solving for the Minimum Degree Pole Placement (MDPP) design discussed in section 4.4.2 at each sample. The resulting Simulink model (ctlexp3b) is shown in Figure 5.3.5.

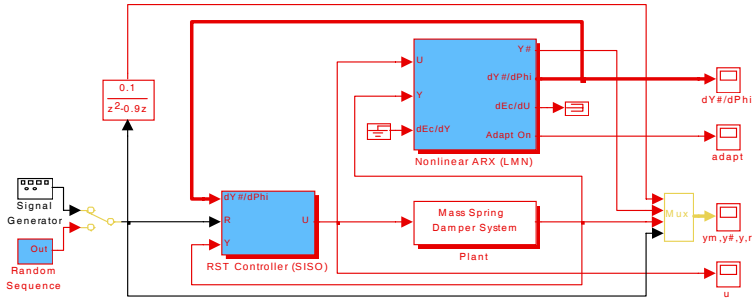


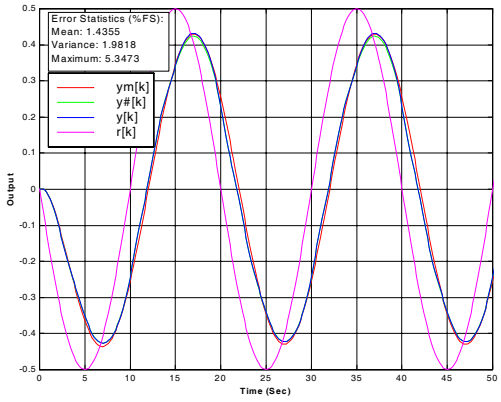
Figure 5.3.5. Instantaneous Linearization Control of Mass-Spring-Damper.

The MDPP design parameters were setup so that the controller solution would cause the closed loop system to behave as a first order system having the following transfer function:

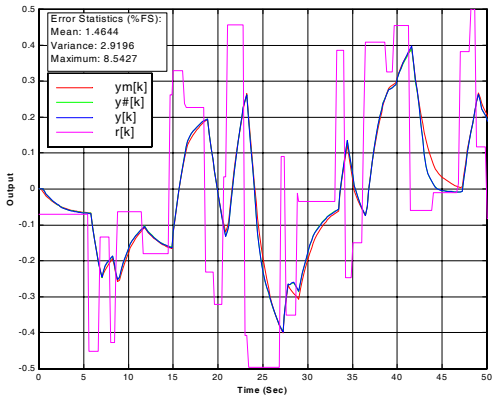
$$G(z) = \frac{0.1}{z - 0.9}$$

Note that in this controller system the required behaviour is obtained as a direct result of the controller design algorithm and is not supplied as a reference model. The inclusion of the transfer function block in the Simulink model is purely to provide data for graphical comparison of the results. The additional delay included in this block is to account for the unit delay in the plant / controller system.

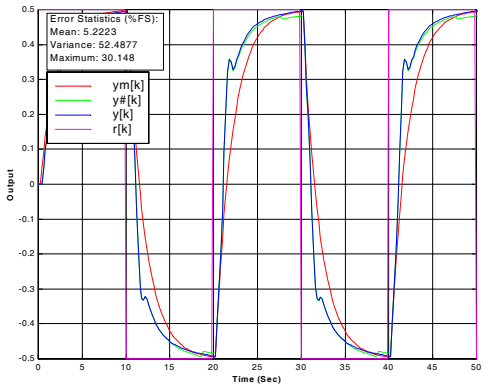
The weights obtained during the identification phase of the experiment were loaded into the identification model and, with the adaptation switched off, the simulation was allowed to execute. The results for various input signals are shown in Figure 5.3.6 below:



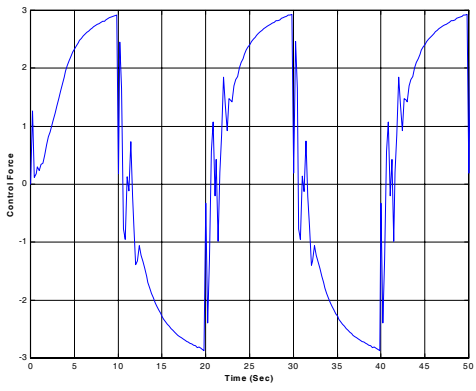
(a) Sine Wave A=0.5, f=1/20 Hz



(b) Uniform Random Signal, A=0.5, p=15%



(c) Square Wave A=0.5, f=1/20 Hz



(d) Input for the Square Wave Signal

Figure 5.3.6. Mass-Spring-Damper System -Closed Loop Responses.

The plots show that, for the sine wave input (a), the system has achieved a response very close to the desired results. For the random input signal (b), tracking error was acceptable, except for short periods of large and rapidly input changes. The cause of these errors becomes more evident when we examine the response to a square wave input (c), together with the control signal generated by such input (d), and highlights the Achilles Heel of the instantaneous linearization technique.

When rapid changes are demanded of the plant the system moves, within one or two samples, from one operating region to another. The control action generated at the original operating point, based on the linear model at that point, is no longer valid at the next operating point. That is, the bias term in the local linearization has become excessive. The plant therefore under or overshoots the desired point because the control gains are sub optimal for the new region of operation.

This problem has been, in part, the motivation behind researchers pursuing model predictive control, which attempts to optimise the future control action at each sample. Interestingly, this approach still requires an estimate of the plant Jacobian at each sample, and thus, the techniques developed in this dissertation have application to these methods as well.

5.3.2 STABILISING AN INVERTED PENDULUM

In the previous section, we saw how instantaneous linearization could be applied to control a BIBO stable plant. We now show how the technique may be applied to an initially unstable system. The system under consideration is commonly used in the control literature to compare different control techniques, namely the inverted pendulum problem. (IPP)

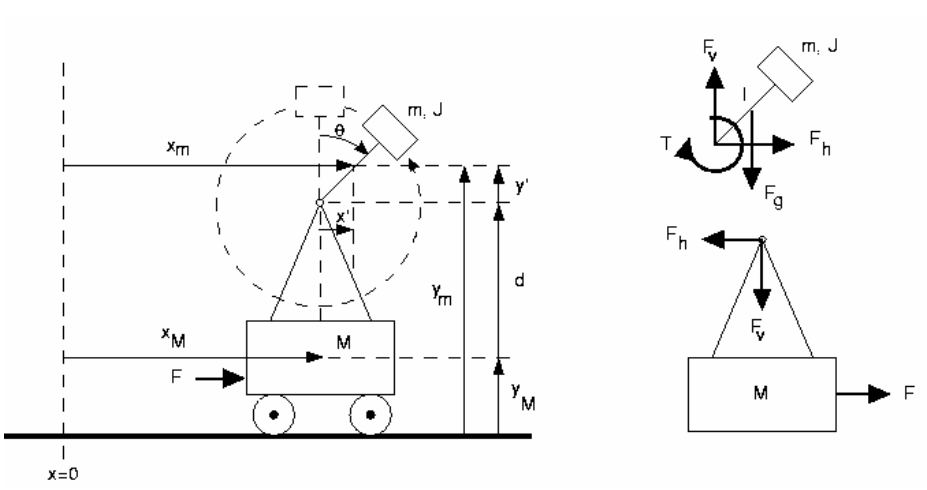


Figure 5.3.7. The Inverted Pendulum Problem.

Shown in Figure 5.3.7 is a diagrammatic representation of the inverted pendulum problem. The system consists of a rigid cart of mass *M* that moves on a flat frictionless surface in the lateral direction only. The location of the cart's centre of mass, measured from some initial point, is represented by the variables x_M and y_M in the horizontal and vertical planes respectively. Suspended *d* units above the cart's centre of mass is a pivot, on which a rigid pendulum arm rotates. Attached to the arm is a mass. The arm-mass combination has a total mass *m,* whose centre of mass is located at the co-ordinates x_m and y_m, which, when measured along the pendulum arm, is *l* units from the pivot. The input to the system is a force *F* acting horizontally on the cart while the controlled value is the angle θ, measured between the vertical and the pendulum arm. The objective is to control the pendulum mass, assuming no input constraints, such that the angle θ will track an arbitrary reference signal in the range $\pm\pi/2$ radians.

The pendulum and cart co-ordinates are related to each other by:

$$\begin{aligned} x_m &= x_M + l\sin(\theta) \\ y_m &= y_M + l\cos(\theta) + d\ . \end{aligned} \tag{5.3.9}$$

By differentiating equation (5.3.9) twice and substituting, we may express the horizontal and vertical forces acting on the pendulum as:

$$\begin{aligned} F_h &= m\ddot{x}_m = m\ddot{x}_M + ml\cos(\theta)\ddot{\theta} - ml\sin(\theta)\dot{\theta}^2 \\ F_v &= m\ddot{y}_m + F_g = -ml\cos(\theta)\dot{\theta}^2 + ml\sin(\theta)\ddot{\theta} + mg \end{aligned} \tag{5.3.10}$$

where F_g is the force due to gravity. Resolving the rotational dynamics of the arm-mass combination results in:

$$T = J\ddot{\theta} = ml^2\ddot{\theta} = -l\cos(\theta)F_h + l\sin(\theta)F_v \tag{5.3.11}$$

where T is the torque at the pivot and J is the moment of inertia of the arm-mass combination. Resolving the horizontal forces acting on the cart gives:

$$F - F_h = M\ddot{x}_M\ . \tag{5.3.12}$$

Substituting (5.3.10) into (5.3.11) and (5.3.12) and rearranging terms results in the following pair of coupled differential equations:

$$\begin{aligned} \ddot{x}_M &= \frac{1}{M+m}\left[F - ml\cos(\theta)\ddot{\theta} + ml\sin(\theta)\dot{\theta}^2\right] \\ \ddot{\theta} &= \frac{1}{2l}\left[-\cos(\theta)\ddot{x}_M + g\sin(\theta)\right]. \end{aligned}$$

These equations are decoupled by substituting the equation for $\ddot{\theta}$ into the equation for $\ddot{x}_M$, rearranging, and substituting the result back into the equation for $\ddot{\theta}$. Defining the state vector, $\boldsymbol{x} = [x_M \quad \dot{x}_M \quad \theta \quad \dot{\theta}]^T$ the final result may be written in state space form as:

$$\begin{bmatrix} \dot{x}_M \\ \ddot{x}_M \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x}_M \\ \dfrac{1}{\left(\frac{M}{m} + 1 - \frac{1}{2}\cos^2\theta\right)}\left(\dfrac{F}{m} - \dfrac{1}{2}g\sin\theta\cos\theta + l\sin(\theta)\dot{\theta}^2\right) \\ \dot{\theta} \\ \dfrac{1}{\left(\frac{2lM}{m} + 2l - l\cos^2\theta\right)}\left(-\dfrac{F}{m}\cos\theta + \left(\dfrac{M+m}{m}\right)g\sin\theta - l\sin(\theta)\cos(\theta)\dot{\theta}^2\right) \end{bmatrix} \tag{5.3.13}$$

$$\theta = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_M \\ \dot{x}_M \\ \theta \\ \dot{\theta} \end{bmatrix}.$$

In order for the instantaneous linearization technique to be applied, one must first identify the plant. This appears to be a dilemma, as we need to collect information from an initially unstable plant to design the controller. The solution is to first design a linear controller that stabilises the plant in some predetermined (possibly narrow) operating region. The identification network is placed in parallel with the stabilised closed

loop plant-controller system, which is then subjected to disturbance. Once the identification network has successfully identified this initial operating region, it can be used in a feedback control loop. In the feedback mode, the identification model is permitted to continue adapting as the operating region is gradually extended. In this way, the identification network slowly extends its "knowledge" of the plant behaviour while maintaining a controlled stable closed loop system.

To demonstrate this, a discrete time linear state feedback LQR controller was designed to stabilise equation (5.3.13). This was done by linearizing the state space model about an equilibrium point, and solving the Ricatti equation (using the "lqrd" command in the Matlab Control System Toolbox) to arrive at a set of regulator gains. The design may be summarised by the following set of values and matrices:

$$M = 0.5, \quad m = 0.2, \quad l = 0.3, \quad g = 9.8$$

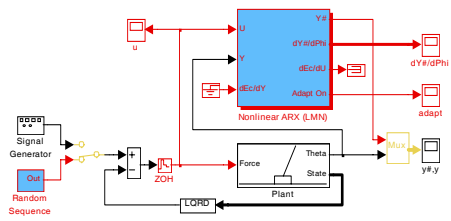
$$\bar{\boldsymbol{x}} = [0 \quad 0 \quad 0 \quad 0]^T, \quad \bar{u} = [0]$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1.6333 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 19.0556 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1.6667 \\ 0 \\ -2.7778 \end{bmatrix}$$

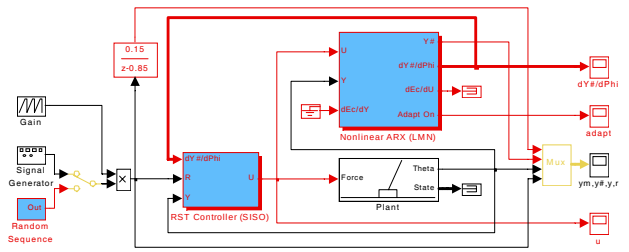
$$\boldsymbol{Q} = \begin{bmatrix} 20 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \quad \boldsymbol{R} = [1]$$

$$\boldsymbol{K} = [-3.6350 \quad -6.1624 \quad -40.2405 \quad -10.3404].$$

The final Simulink models used to perform the simulation for both the initial identification (ctlexp5a) and the "extension" step (ctlexp5b) are shown in Figure 5.3.8.



(a) Initial Indentification System Model



(b) "Extended" Identification and Control System Model

Figure 5.3.8. System Models used for Instantaneous Linerization Control of an Inverted Pendulum.

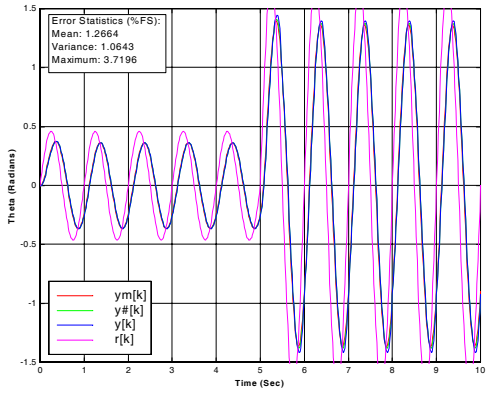
It is important to note the zero order hold block placed in front of the plant in Figure 5.3.8 (a). This block is necessary to prevent the identification network from identifying the closed loop system dynamics, and must have the same sampling period as the identification network. It is also the reason why the discrete time version of the LQR design algorithm was used in the regulator design.

To perform the initial identification step the network was setup as described in the table below:

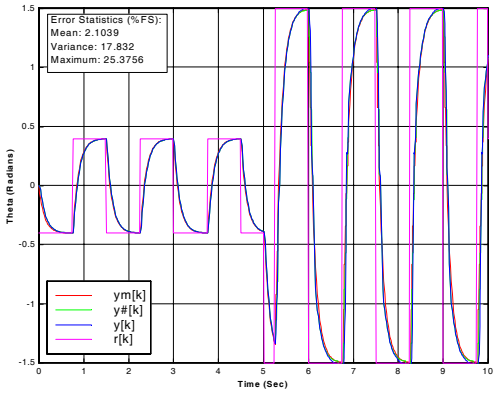
System Identification Setup Parameters			
General Network Parameters			
Network Type:	Local Model Network		
Initial Weight Range (Min / Max):	0 / 0		
Validity Activation Threshold (%):	2		
Sample Time (Sec):	0.02		
Network Inputs (In information vector order)			
Name	**Range (Min/Max)**	**Number of regions**	**Overlap (%)**
y[k-1]	-1 / 1	5	40
y[k-2]	-1 / 1	1	100
u[k-1]	-50 / 50	1	100
u[k-2]	-50 / 50	1	100
Error Tolerances (%)			
Relative Differential	**Absolute Differential**	**Relative Value**	**Absolute Value**
10	0.0005	0.01	0.01
Adaptation Method Parameters			
Adaptation Algorithm:	Exponential Forgetting Factor		
Initial Covariance Diagonal:	10		
Forgetting Factor:	0.995		
Random Signal Parameters			
Minimum / Maximum	-13 / 13		
Probability of value change (%)	15		
Distribution	Uniform		
Sample Period	0.02		

Table 5.3.2. Inverted Pendulum Identification Setup Parameters

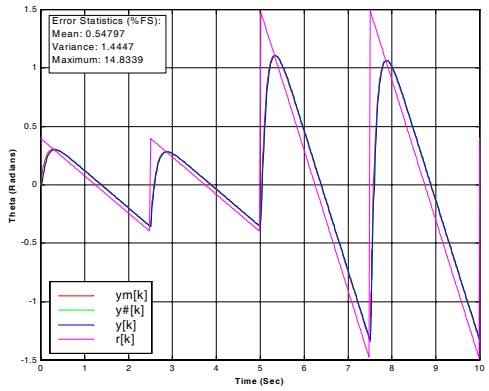
The simulation was run for 500 seconds using a random disturbance input signal (see table), with the final weights being saved. These weights were then transferred to the "extension" part of the experiment where the simulation was run for 700 seconds using a 0.5Hz square wave demand signal. The demand input, initially at 0.4 radians in amplitude, was gradually increased to its maximum amplitude of 1.4 radians over a 500 seconds period. For the final 200 seconds the demand amplitude remained constant at 1.4 radians. The network was allowed to adapt during this entire period. Once 700 seconds had elapsed the weights were saved and the process repeated using the weights obtained so far, but with the square wave being replaced by a uniformly distributed random demand signal. The maximum amplitude of the signal was permitted to follow the same amplitude schedule used for the square wave. The probability of value change at each sample was set to 15%. The final weights were saved and the adaptation was turned off. Finally, the system was presented with various demand signals without any adaptation taking place. The closed loop response of the final system to these signals is shown in Figure 5.3.9.



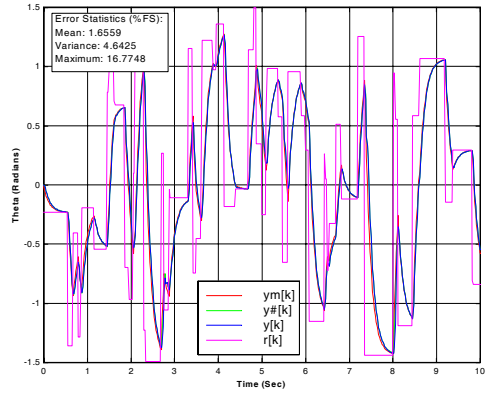
(a) Sine Wave, f=1Hz



(b) Square Wave, f=0.667Hz



(c) Sawtooth Wave, f=0.4Hz



(d) Uniform Random Signal, Max A=1.5 Rad, p=15%

Figure 5.3.9. Instantaneous Linearization Control of an Inverted Pendulum - Closed Loop Responses.

Clearly, the final controller performs exceptionally well demonstrating tracking to pendulum angles in excess of 85 degrees. Tracking is remarkably accurate in all but the most difficult of circumstances. Note that the control signal input must approach infinity as the pendulum approaches 90 degrees. Thus, as these large angle values are approached, numerical stability problems of the controller design algorithm result.

5.4 DISCUSSION AND CONCLUSIONS

We conclude this chapter by discussing some of the general issues that have not been directly addressed in the previous examples but are pertinent to the approach used. Most of the issues have a direct parallel in linear system identification and adaptive control. The reader can thus obtain a wealth of pertinent knowledge from more general texts (Astrom and Wittenmark, 1995, Ljung, 1991, 1999) in these areas.

5.4.1 A PRIORI PLANT INFORMATION

As with any system that requires system identification, the most challenging part of the problem is usually identifying the system involved. The problem is further exacerbated when one adds the additional burden of

identifying non-linearities in the system. Unfortunately, this step, in one form or another, is impossible to avoid. It simply is not feasible to control a system about which the controller has no knowledge. This knowledge may be implicit in the control design procedure or is attained on-line, as in an adaptive system. Usually, the most desirable form of this knowledge is a mathematical model based on the physics of the problem. This form generally permits a critical analysis of the plant's characteristics, leading to information about its stability and operating conditions and, ultimately, a controller design. However, in many practical systems, an accurate mathematical model cannot be derived because one's knowledge of the system is incomplete, or it becomes impractical to model all the processes taking place in the system. Fortunately, the information describing a system usually resides somewhere in the continuum between the extremes just mentioned. It is desirable to exploit this information to its maximum extent. This point is demonstrated by the use of equations (5.3.6) and (5.3.7), obtained because we had a through understanding of the plant physics, to determine the lag space and operating regions of the identification network used in section 5.3.1. The information was not, however, a prerequisite for the application of the technique.

The more analytically derived information we have the better. For example, we may typically know the general structure of the plant, represented by equation (5.3.4), but not know the constants for the particular system under consideration. This greatly eases the system design burden, because the lag space that must be represented by the information vector and the variables on which the non-linearities are dependent are immediately known. This information becomes crucial in determining the optimal identification model. Methods exist for estimating the required lag space (He and Asada, 1993) however, if the natures of the non-linearities are unknown, it is difficult to even estimate the optimum number and location of operating regions to be used.

Clearly, LMN networks enjoy a substantial advantage over MLP networks in this area. Because LMN networks are essentially interpolated linear models, the relationship between the network weights and the physical system is reasonably transparent. This permits the use of linear control techniques in analysing the network results. It also allows a designer to insert *a priori* knowledge about the plant in the network weights before training is started. In this way, an identification model may be very close to the overall solution at the outset.

Although not shown in this dissertation, this same technique could be used to provide the initial stabilising control for a system such as the IPP. In section 5.3.2 a LQR regulator was designed to provide initial stabilisation. An alternative approach may have been to obtain a linear I/O model of the system at the equilibrium point and insert this model into the identification network, thus dispensing with the initial LQR design altogether.

5.4.2 IDENTIFICATION IN THE PRESENCE OF DISTURBANCES

An important issue, beyond the scope of this dissertation, is the effect of disturbance noise on the identification process and network adaptation. The techniques shown here use only NARX models of the process. To account for disturbances it may be necessary to use models with more complexity, such as NARMAX models, in the identification step. These models, and their application to non-linear system identification, have been well studied in the literature and the reader is referred to references such as (Leontaritis and Billings, 1985) and

Sjoberg *et al*, (1994) for further information. Unfortunately, these more complicated forms all use feedback in the network structure, which may lead to stability problems of the network model.

5.4.3 THE NETWORK VERIFICATION STEP

In contrast to batch mode techniques, when the implemented adaptation activation method is used, the verification step used to test the identification model is not strictly necessary. This is because the adaptation is automatically switched off when the network meets the required specification and all subsequent input samples may be viewed as test vectors for the network. If the network attains a state where the adaptation method effectively remains deactivated then the required mapping has been learnt. When the network cannot attain the required accuracy or is over fitting the solution, the adaptation mechanism will be activated whenever an input sample falls within a region of the mapping where the network is providing an inaccurate estimate. Naturally, this behaviour is a trade-off because adapting the mapping in one area can lead to deterioration of network performance in another. If the network is incapable of providing the required mapping, instability of the adaptation process may result.

5.4.4 STABILITY

The stability of the system during adaptation is a function not only of the controller / model combination but also, as implied in the previous section, of the adaptation process itself. This problem, generally referred to as robust adaptive control, has been tackled, in the linear case, by a number of researchers with a substantial degree of success. Typically, a Lyapunov stability approach is used to derive the laws and conditions under which parameter adaptation will remain stable. The non-linear counterpart is a complex problem which, to the author's knowledge, is still an open issue. This observation obviously raises questions about the practicality of the approach used in section 5.3.2. Clearly, the approach can work, but the exact conditions under which stability can be maintained must be resolved before being used on a "real" plant were instability poses any serious risk.

The use of Local Model Networks makes the stability analysis of the non-adapting closed loop stem far more tractable, because the system may now be viewed as a number of interpolated linear systems. As demonstrated in section 5.3.1, the pole zero plot of the system can be generated as a by-product of the identification model. However, this plot must be used with caution, as its accuracy is entirely dependent on the accuracy of underlying identification model. System poles and zeros can exhibit extreme sensitivity to small variations in the polynomial coefficients resulting in an inaccurate pole zero plots. Furthermore, the plot cannot be obtained until the plant is identified, which, for an unstable system requires the controller. The only solution to this dilemma triggers the problem described in the previous paragraph.

Note that these issues are not a result of using neural networks per se, but are general problems associated with any non-linear control technique that is adapting because of incomplete plant information.

5.4.5 MISCELLANEOUS ISSUES

The examples shown in this chapter have used both discrete and continuous time models of the plant. The identification models and controllers are, because of their structure, all discrete time systems. This means care

must be taken to ensure that the sample rates are high enough to capture all appropriate plant dynamics. If the plant under consideration is not band limited, or the measurements contain unwanted high frequency components then anti-aliasing filters must be used.

Although only SISO plants have been shown in the control examples, the extension of the technique to MIMO systems is straightforward. Naturally MIMO identification is considerably more complex and, if possible, it is usually beneficial, from an identification viewpoint, to break the MIMO system down into a number of separate MISO systems. From a controller perspective this may be counter productive and a trade-off may be necessary.

The bulk of this work demonstrates the calculation of differentials using LMN networks. The computing technique is also applicable, in a simpler form, to RBF networks. The approach in MLP networks has already been well established. In the interests of brevity, the experiments for each network type have not been presented, but simulations using all the networks were performed. These simulations demonstrate, from a systems viewpoint, the ability to interchange various networks.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 SYNOPSIS

This work has covered a range of topics all aimed at demonstrating the online computation and use of feedforward artificial neural networks (ANN) Jacobian or system gradient information in neurocontrol applications. More specifically, the networks under consideration were the multi-layer perceptron (MLP), the radial basis function (RBF) network and local model network (LMN) with particular emphasis being placed on the latter.

Beginning in chapter two, an attempt was made to treat these networks as information processing systems whose analysis could be subdivided into three broad levels of consideration; computation abilities, algorithmic structure, and implementation details. Furthermore, all the networks were described under a general unifying neural network framework consisting of five major elements, namely; the operating environment, the network components, an activation update rule, the objective function and lastly the learning or adaptation rule.

The adaptive control operating environment of interest dictated that the key computational ability required was for the networks to describe a non-linear dynamic system using a time history NARX model. It was shown in chapter two, using various results from approximation theory, such as the Stone-Weierstrass theorem, Kolomogorov's representational theorem and Shannon's sampling theorem, that ANN's do indeed provide a viable solution to the non-linear function approximation problem, the basis required for the formation of an NARX model. Unfortunately, these results provided no insight into the actual design procedure required for an MLP network to perform an approximation within a given error tolerance. More insight could indeed be obtained for LMN and particularly for RBF networks, if some knowledge about the extent of the domain and smoothness of the approximated function was known. The main criticism of RBF networks was their inefficient use of memory and susceptibility to the curse of dimensionality. The LMN provided a reasonable compromise between the disadvantages of RBF and MLP networks. LMN's also provided an intuitively appealing and familiar approach to "pre-program" the network with *a-priori* information.

In chapter three, the core of this dissertation, the algorithmic and implementation levels of analysis were addressed. The chapter began by concentrating on the activation update rule element for each of the network

types. Simple Matlab code listings were provided that showed how efficient activation rule implementations could be accomplished. In particular, for RBF and LMN networks, it was shown how an axis orthogonal implementation could be used to dramatically reduce the computational expense for high dimension systems. Also demonstrated was how a LMN network could be constructed within the general neural network framework described in chapter two.

Next, significant space was dedicated to describing the objective function and adaptation or learning rule elements. These two elements are inextricably linked as the adaptation rule follows from the objective function that is optimised when learning is performed. It was shown, by judicious choice of variable construction, that the learning rule for each of the three network types could be described as a linear or pseudo-linear regression problem. This allows the adaptation of all three network types to be approached as an optimisation problem and to develop solutions using a common systems identification framework without regard to the intricacies associated with specific network formulations. With this ground work in place a number of adaptation rules or training algorithms were presented based primarily on recursive least squares (RLS) principles. Each of these algorithms attempted to address potential difficulties and problems encountered when performing online system identification. The final method presented, namely the recursive SVD algorithm, was developed by the author to circumvent many of the problems associated with other RLS methods. The section on the learning rule closed by discussing implementation specific issues such as network specific regressor calculation and the back propagation algorithm, the use of dead zones, global vs. local learning and structure optimisation.

The penultimate section of chapter three derived an expression for the network Jacobian for each network type. The expression for the LMN network forms the basis for satisfying the first research objective. That is, to derive an algorithm to efficiently calculate the system gradient of an LMN network. The second objective, to implement this algorithm in an online manner that can be used in non-linear adaptive neurocontrol, is also discussed in detail. This includes the presentation of Matlab code which makes maximum use of the efficiencies of the previously presented axis orthogonal implementation of the activation rule for both RBF and LMN networks.

In the fourth chapter, focus shifted to the control aspects of the neurocontrol problem. A five step design methodology, of which three were discussed in detail, was presented. The two omitted steps, stability and parameter convergence analysis were considered beyond the scope of the research topic. Important results from the remaining three steps, namely, plant description, control law formulation and adaptation mechanisms were derived. The conditions for which a valid I/O plant description of a continuously differentiable discrete time non-linear state space plant could be attained were derived. Two different control law formulations, series inverse control (SIC) and the RST controller using minimum degree pole placement design (MDPP), were discussed. The latter control law formulation was coupled with system identification using neural network to form a control system based on instantaneous linearization. Adaptation mechanisms for various combinations of indirect and direct adaptation in both model reference and self tuning regulator systems were also investigated.

Finally, in chapter five, the information and analyses of the preceding chapters was combined to perform a number of simulations based on a Simulink block library which was specifically created for this purpose. In this

chapter it was shown how the online LMN Jacobian extraction algorithm enhances the flexibility of the LMN permitting its use in both model free and model based neurocontrol applications, thus satisfying the third research objective. This was done using both SIC and instantaneous linearization paradigms controlling a number of different plants.

6.2 OBSERVATIONS AND CONCLUSIONS

During the course of this work there have been many insights and new problem perceptions that have come to light. The set of conclusions presented in this section only highlights what the author believes to be the most important. We begin with the research objectives:

- All three research objectives were met. Algorithms to efficiently calculate system gradient or network Jacobian matrices were devised not only for the LMN network, but also for the RBF and MLP networks as well.
- These algorithms were successfully implemented in an online fashion for all three network types. It was shown, for axis orthogonal RBF and LMN networks, that this could be achieved with minimal computational expense during the activation rule evaluation.
- The algorithms were demonstrated in a non-linear adaptive neurocontrol setting using both model free and model based control paradigms. The LMN network was successfully substituted for an MLP network with no changes to the system structure thus demonstrating the enhanced flexibility of the LMN network and how it could be used as a “black box” replacement for an MLP network. Obviously this was subject to the capability of each of the chosen network structures to perform the required mapping.

Although the research objectives were successfully met there are a number of difficulties associated with the general approach. In particular, the proposed time shift operator polynomial representation of the plant for non-linear adaptive control applications is problematic for all but the simplest systems:

- Knowledge about the plant delays and order must be implicitly included in the control system structure. This knowledge is typically not available. Furthermore, for MIMO systems, the delay and number of zeros associated with each output may differ complicating the plant description still further. The resulting system is inefficient and difficult to successfully identify. More compact and succinct plant descriptions, such as those obtained when using subspace identification methods, may provide a better framework for neurocontrol systems.
- The system root locations may be highly sensitive to operator polynomial coefficient values. This means that the parameter identification must be highly accurate in order for good control to be achieved. If this cannot be guaranteed then the controller design must be made robust to these uncertainties, potentially negating the performance improvement gained by using non-linear or adaptive techniques. This renders the entire approach moot.

Another area that is difficult to successfully implement is the online adaptation of the system:

- The recursive algorithms used to train the networks in an online environment can be difficult to setup and are prone to numerical problems. In this work significant effort has been spent to address this issue by investigating and developing algorithms that are stable and robust to numerical errors.
- A thorough understanding is necessary to recognise when adaptation problems may arise. An unwary user may easily be lulled into a sense of false security as the adaptation process may appear to work well under test or simulated conditions but real-world controllers must be robust to many environmental uncertainties. This obviously requires rigorous proofs to ensure convergence to the *correct* system parameters. Although the requirement for persistent excitation and the conditions under which this can be achieved are well known, it may be difficult to achieve in practice.
- Proofs to ensure stability of the system are also crucial. Although some stability results do exist they are applicable only to specific conditions and system constructs. To the author's knowledge all the current proofs involving neural networks in a control context are only applicable to direct adaptation schemes.

In spite of these difficulties, with care, successful implementations can be created. The use of network Jacobian information can be useful not only in control applications but also for applications such as virtual instrumentation, system monitoring and signal selection and fault detection to name but a few. However, the difficulties pointed out in many of the conclusions above highlight the fact that ANN's should be viewed as only another tool at a designer's disposal and not, as is sometimes done, a "silver bullet" to avoid tackling the real issues underlying a problem.

6.3 RECOMMENDATIONS FOR FUTURE WORK

As with most research, the evolution of this work has raised more questions than were answered. There were many topics discussed for which substantial bodies of work exist but were only superficially considered. In particular, the topics of approximation theory and neurocontrol come to mind. Neural networks have been applied and studied in many different fields resulting in a wide range of nomenclatures, perspectives and customised algorithms. This multi-disciplinary nature has resulted in a large volume and scope of research material. Varying degrees of mathematical sophistication and diversity are required to fully understand the potential pitfalls associated with various methods, making it difficult for new researchers or developers to *effectively* apply the techniques in a reasonable time frame. With these comments in mind, the following therefore represents a partial list of potential future work:

- In this work an attempt has been made to provide a common framework in which to discuss the various network structures. However, the majority of the literature tends to consider the various network paradigms as distinct. The author believes that this approach is causing potential synergies to be overlooked. For example it could be highly advantageous if a method could be devised to map RBF or LMN structures into an MLP network, much like the LMN may be considered an extension of the RBF network. This may, at first, seem absurd, but it is intriguing to note that the Gaussian activation functions used in RBF's can be

constructed from “back to back” sigmoid activation functions frequently used in MLP’s. The payoff is that such a mapping would potentially allow the RBF network construction results to be used for MLP networks.

- Another area of unification where potential synergies exist is in a more holistic approach to supervised and unsupervised networks. Although not explicitly discussed in this work, unsupervised networks have also been successfully used in neurocontrol work. They exhibit interesting properties particularly in terms of expressing a plant’s salient features in a compact form using techniques such as Principle Component Analysis (PCA) or Independent Component Analysis (ICA). These approaches enhance the ability to reject statistically irrelevant information such as system noise. This is precisely one of the shortcomings of the techniques used in this work. Interestingly, the recursive SVD algorithm developed in section 3.3.8 performs essentially the same function, albeit in a different context, as principle component analysis and provides a promising starting point for this research.
- The SVD algorithm mentioned above also tackles the problem of algorithmic stability associated with the recursive identification procedure. Although there have been other attempts at using the SVD in recursive neural network system identification, all these approaches tend to focus on either improving the covariance update equation or the parameter update equation separately. The distinctive formulation of the objective function in section 3.3.8 allows one to address both of these equations simultaneously and to the authors knowledge the resulting method is unique. The algorithm has some interesting properties, but more refinements are needed to reduce the computational expense and to obtain more rigorous results regarding its stability and convergence properties.
- Another important and ongoing research area in neural networks is the structure optimisation problem. Clearly this is closely related to defining the initial construction of a network in a logical and methodical manner. We have seen in chapter two that, given limited information about the desired mapping, RBF networks can be constructed, using spectral analyses, which are guaranteed to approximate the mapping within some given error bound. Such results are very limited for MLP’s and LMN design techniques might also be enhanced by using a spectral approach. If no knowledge about the desired mapping is known then the problem reverts to one of online structure optimisation. This is an extremely challenging area of research in which completely generalized results are perhaps not even possible.
- Most directly related to extensions of the work presented here would be the enhancement of the Jacobian calculations for non-axis orthogonal networks, which may not always provide the best structure for a particular problem. Also desirable would be a method for determining error bounds associated with the Jacobian information. More in-depth applications in neurocontrol could also be useful; particularly the use of Jacobian information in reducing the computation expense incurred in the optimisation search phase of model predictive control applications. Finally, in the control arena, stability results and parameter convergence of the neural network approach continues to be a challenging research problem.

The list of suggested further work presented above is by no means exhaustive. With the exception of the last two bullets, the common theme that the author feels is important going forward is that primary research efforts be

focused on a holistic view of neural network processing and the enabling mathematics. While niche algorithms and applications are important for solving engineering problems and may provide insights not previously recognised, research should primarily be aimed at enhancing the essential body of knowledge necessary for the advancement of the field.

REFERENCES

Albus, J.S. (1971), *A Theory of Cerebellar Function*, Mathematical Biosciences, Vol. 10, pp 25-61

Albus, J.S. (1981), *Brains Behaviour and Robotics,* Peterborough, NH: BYTE Books

Anderson, J. Pellionisz, A. Rosenfeld, E. [Editors], (1990), *Neuro-computing 2. Foundations of Research,* MIT press, Cambridge, Mass.

Anderson, J. and Rosenfeld, E. [Editors], (1988), *Neuro-computing Foundations of Research*, MIT press, Cambridge, Mass.

Arbib, M. A. (Editor), (1995), *The Handbook of Brain Theory and Neural Networks,* Bradford Books, MIT Press, Cambridge MA

Åström, K.J. Wittenmark, B. (1995), *Adaptive Control, 2nd Edition,* Addison –Wesley

Åström, K.J. Wittenmark, B. (1997), *Computer-Controlled Systems: Theory and Design, 3rd Edition,* Prentice-Hall

Bailer-Jones, C.A.L. MacKay, D.J.C. Withers, P.J. (1998), *A recurrent neural network for modelling dynamical systems,* Computational Neural Systems, Vol 9, pp 531-547

Barron, Andrew R. (1994), *Approximation and Estimation Bounds for Artificial Neural Networks,* Machine Learning, Vol 14, pp 115-133

Bierman, G.J. (1977), *Factorization Methods for Discrete Sequential Estimations,* Academic Press, New York NY

Bosman, S. (1996), *Locally weighted approximation - Yet another type of neural network*, MSc Thesis, Department of Computer Science, University of Amsterdam, The Netherlands, July 1996

van Breeman, A.J.N. (1997), *Neural Adaptive Control,* Master's Thesis - Report # BSC003N97, University of Twente, The Netherlands, January 28 1997

van Breeman, A.J.N. Veelenturf, L.P.J. (1996), *Neural Adaptive Feedback Linearization Control,* Journal A, Vol. 37, No. 3, pp 65-71, October 1996

Deutsch, S. (1983), *RCG Cable Analysis of a Dendritic Tree Based on Rall's Idealised Model*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, pp 1007-1010, September / October

Eberhart, R.C. Dobbins, R.W. (1990), *Early Neural Network Development History: The Age of Camelot*, IEEE Engineering in Medicine and Biology Magazine, Vol. 9, No. 3, pp 15 - 18

Foss, B.A. Johansen, T.A. (1993), *On local and fuzzy modelling*, Proceedings of the Third International Conference on Industrial Fuzzy Control and Intelligent Systems, IFIS '93, pp 80 - 87, Houston, TX, December

Golub, G.H., Van Loan, C.F. (1989), *Matrix Computations,* The Johns Hopkins University Press, Baltimore MD

He, X. and Asada. H. (1993), *A New Method for Identifying Orders of Input-Output Models for Nonlinear Dynamic Systems,* Proceedings of the American Control Conference, San Francisco California

Hunt, K.J. Sbarbaro, D. (1991), *Neural Networks for Nonlinear Internal Model Control*, IEE Proceedings Part D, Vol. 138, No. 5 pg 431-438, September 1991

Hunt, K.J. Haas, R. Murray-Smith, R. (1996), *Extending the functional equivalence of radial basis function networks and fuzzy inference systems*, IEEE Transaction on Neural Networks, Vol 7, No. 3 pp 776 - 781, May

Hunt, K.J. [Editor], (1993), *Polynomial Methods in Optimal Control and Filtering,* Volume 49 of IEE Control Engineering Series, Peter Peregrinus

Hunt, K.J. Sbarbaro, D. Zbikowski, R. Gawthrop, P.J. (1992), *Neural Networks for Control Systems - A Survey*, Automatica, Vol 28, No. 6 pp 1083-1112

Hyötyniemi, H. (1994), *Self-Organizing Artificial Neural Networks in Dynamic Systems Modeling and Control,* Helsinki University of Technology, Control Engineering Laboratory, Report 97, November

Hyötyniemi, H. (1996), *Regularization of Parameter Estimation,* The 13th IFAC World Congress, July 1-5, San Francisco, California.

Johansen, T.A. Foss, B.A. (1992), *A NARMAX model representation for adaptive control based on Local Models,* Modeling Identification and Control, Vol. 13, No. 1, pp25-39

Johansen, T.A. Foss, B.A. (1993), *Constructing NARMAX models using ARMAX models,* International Journal of Control, Vol. 58, pp1125-1153

Leontaritis, I.J. Billings, S.A. (1985), *Input-Output parametric model for non-linear systems. Part I: Deterministic Non-linear Systems*, International Journal of Control, Vol 41, pp303-328

Lewis, E.R. (1983), *The Elements of Single Neurons: A Review*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, September / October

Ljung, L. and Sonderstrom, T. (1983), *Theory and Practice of Recursive Identification,* MIT Press, Cambridge MA

Ljung, L. (1991), *System Identification Toolbox User's Guide*, The Mathworks Inc

Ljung, L. (1999), *System Identification - Theory for the User*, Prentice Hall

Motter, M.A. (1998), *Control of the NASA Langley 16 Foot Transonic Tunnel with the Self Organising Feature Map*, Ph.D Thesis, University of Florida.

Murray-Smith, R. and Johansen T.A.[Edited], (1997), *Multiple Model Approaches to Modelling and Control*, Taylor and Francis, London

Niznik, C.A. (1983), *The Neural Path Probabilistic Delay Model*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, pp 1014-1018, September / October

Parker, S.P. (Editor in Chief), (1997), *Dictionary of Mathematics,* McGraw-Hill, New York NY

Pearlmutter, B.A. (1989), *Learning State Space Trajectories in Recurrent Neural Networks,* Proceedings of International Joint Conference on Neural Networks, Washington DC Vol. 2, pp 365-372

Pineda, F. (1987), *Generalization of Back Propagation to Recurrent Neural Networks*. Physical Review Letters, Vol. 19, pp 2229-2232

Platt, J. (1991), *A resource allocating network for function interpolation,* Neural Computation, Vol 3, pp 213-225

Poggio, T. and Girosi, F. (1990), *Gaussian Networks for Approximation and learning*, Proceedings of the IEEE, Vol. 8, No. 9, pp 1481 - 1497, September

Psaltis, D. Sideris, A. Yamamure, A.A. (1988), *A Multilayered Neural Network Controller*, Control Systems Magazine, Vol. 8, No. 2, pp 17-21

Quinlan, P.T. (1998), *Structural change and development in real and artificial neural networks*, Neural Networks, Vol: 11, Issue: 4, pp. 577-784, June

Rumelhart, D.E. McClelland, J.L. and the PDP Research Group. (1986), *Parallel Distributed Processing, Vols. I and II*, MIT Press, Cambridge Mass.

Sanner, R.M. and Slotine, J.J.E. (1992), *Gaussian Networks for Direct Adaptive Control*, IEEE Trans. on Neural Networks, Vol. 3, No. 6, pp 837-863, November

Sbarbaro, D. and Johansen, T.A. (1997), *Multiple local Laguerre models for modelling nonlinear dynamic systems of the Wiener class*, IEE Proceedings- Control Theory and Applications, Vol. 144, No. 5, pp 375-380

Schwenker, F. Kestler, H.A. Palm, G. (2001), *Three learning phases for radial-basis-function networks,* Neural Networks, Vol 14, pp 439-458

Sjöberg, J. Hjalmlmerson, H. Ljung, L, (1994), *Neural Networks in System Identification,* Pre-prints of the 10th IFAC symposium on SYSID, Copenhagen, Denmark. Vol 2, 49-71

Soloway, D. Haley, P.J. (1997), *Neural Generalized Predictive Control: A Newton-Raphson Implementation*, NASA TM 110244, NASA Langley Research Center, February 1997.

Sørensen, O. (1994), *Neural Networks in Control Applications*, Ph.D Thesis, Aalborg University, Department of Control Engineering

Stevens, J.K. (1985), *Reverse Engineering the Brain*, Byte, pp 287-299, April 1985

Stepniewski, S.W. Jorgensen, C.C. (1998), *Toward a More Robust Pruning Procedure for MLP Networks*, NASA TM 1998-112225, Ames Research Center, Moffet Field, CA, April 1998

Sundararajan, N. Saratchandran, P. (2000), *Analysis of Minimal Radial Basis Function (RBF) Neural Network Algorithm for Real-Time Identification of Nonlinear Dynamic Systems,* IEE Proceedings – Control Theory Applications, Vol 147, No. 4 pp 476-484

Suykens, J.A.K. Bersini, H. (1996), *Neural Control Theory: an Overview,* Journal A, Vol. 37, No. 3, pp 4-10, October 1996

Suykens, J.A.K. Vandewalle, J.P.L. De Moor, B.L.R. (1996), *Artificial Neural Networks for Modelling and Control of Non-Linear Systems*, Kluwer Academic Publishers

Suzuki, S. (1998), *Constructive function approximation by three-layer artificial neural networks*, Neural Networks, Vol 11 pp1049-1058

Williams, R.J. and Zisper, D. (1988), *A Learning Algorithm for Continually Fully Recurrent Neural Networks,* Technical Report ICS Report 8805*,* UCSD, La Jolla, CA 920923, November

Yingwei, L. Sundararajan, N. Saratchandran, P. (1997), *A Sequential Learning Scheme for Functional Approximation Using Minimal Radial Basis Function Neural Networks,* Neural Computation, Vol 9, pp 461-478

Yingwei, L. Sundararajan, N. Saratchandran, P. (1998), *Performance Evaluation of a Sequential Minimal Radial Basis Function (RBF) Neural Network Learning Algorithm,* IEEE Transaction on Neural Networks, Vol 9, No. 2 pp 308-318

Zbikowski, R. Hunt, K.J. Dzielinski, A. Murray-Smith, R. Gawthrop, P.J. (1994), *A Review of Advances in Neural Adaptive Control Systems*, ESPRIT III Project 8039, Daimler Benz AG, University of Glasgow, Scotland, June 1994

Zbikowski, R.W. (1994), *Recurrent Neural Networks: Some Control Aspects,* Ph.D. Thesis, Department of Mechanical Engineering, Glasgow University, Glasgow Scotland, May

Zhang, Y. Dai, G. Zhang, H. and Li, Q. (1994), *A SVD-Based Extended Kalman Filter and Applications to Aircraft Flight State and Parameter Estimation,* IEEE Proceedings of the American Control Conference, June 1994, Baltimore, Maryland.

Zhang, Y. and Li, Q. (1999), *A Fast U-D Factorization-Based Learning Algorithm with Applications to Nonlinear System Modelling and Identification*, IEEE Transaction on Neural Networks, Vol 10, No. 4 pp 930 – 936