RECURSIVE TIME- AND ORDER- UPDATE ALGORITHMS FOR RADIAL BASIS FUNCTION NETWORKS

By

MENG HOCK FUN

Bachelor of Engineering
Oklahoma State University
Stillwater, Oklahoma
1993

Master of Engineering
Oklahoma State University
Stillwater, Oklahoma
1996

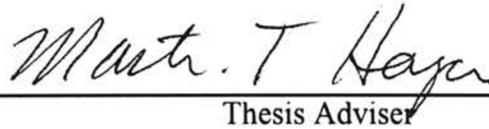
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
August, 2001

RECURSIVE TIME- AND ORDER- UPDATE

ALGORITHMS FOR RADIAL BASIS

FUNCTION NETWORKS

Thesis Approved:



Thesis Adviser

Dean of the Graduate College

ACKNOWLEDGMENTS

In this acknowledgement, I wish to express my heart-felt gratitude to my major adviser, Dr. M. T. Hagan. My achievement today would not be possible without his generous financial support, guidance, encouragement, advice, and friendship throughout my entire graduate studies at Oklahoma State University. I would like to thank my other committee members Dr. Carl Latino, Dr. Gary Yen and Dr. Eduardo Misawa for their helpful suggestions and assistance.

I would like to express my special appreciation to my wife, Amelia, for her strong encouragement at difficult times and loving support. In addition, I would like to thank my beloved mother, my sisters and my brother for their encouragement.

Finally, I would like to thank Halliburton Energy Services and Department of Electrical Engineering for their financial support in this research.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION 1
1.1 Objective 2
1.2 Contributions 3
1.3 Outline 5

CHAPTER 2
NETWORK ARCHITECTURES 7
2.1 Introduction 8
2.2 Basic Neural Network Architectures 9
2.2.1 Biological Neural Networks 9
2.2.1.1 Single Biological Neuron 10
2.2.1.2 Massive Interconnections and Parallel Structure 10
2.2.2 Artificial Neural Networks 11
2.2.3 Single Artificial Neuron 11
2.2.4 Transfer Function 12
2.2.5 Multiple-Input Neuron 13
2.2.6 A Layer of Neurons 14
2.2.7 Multilayer Network 16
2.2.8 Universal Approximation Capability 17
2.3 The Radial Basis Function Network 19
2.3.1 The Radial Basis Functions 19
2.3.2 The RBF Network Architecture 21
2.3.3 Universal Approximation Capability 23
2.4 Mathematical Preliminaries 24
2.4.1 RBF Network - Linear in Parameters 24
2.4.2 Time-Update Framework 27
2.4.3 Order-Update Framework 29
2.4.3.1 Order-Increase-Update 29
2.4.3.2 Order-Decrease-Update 31
2.4.4 Time and Order Update Framework 32
2.5 Special RBF Network for Subset Selection 35
2.6 Summary 37

CHAPTER 3
PROBLEM STATEMENT 38
3.1 Introduction 39
3.2 Objective 39
3.3 Illustrative Example 42
3.4 The Research Outline 45

CHAPTER 4
THE LEAST SQUARES METHOD 47
4.1 Introduction 48
4.2 Linear Model 48
4.3 Solving the Linear Model 50
4.3.1 The Linear Least Squares Problem 50
4.3.2 Characterization of Least Squares Solution 51
4.4 Orthogonal Projection 52
4.4.1 Subspace Projection 53
4.4.2 Matrix Projection and Geometrical Interpretation 54
4.5 Orthogonal Transformation 56
4.5.1 Motivation 56
4.5.2 Orthogonal Least Squares Method 57
4.6 Givens Rotations 58
4.6.1 Givens QR Methods 61
4.7 Summary 62

CHAPTER 5
TIME-UPDATE ALGORITHMS 63
5.1 Introduction 64
5.2 Matrix Inversion Lemma 66
5.3 Recursive Least Squares Algorithm 67
5.3.1 Time-Update for the Parameter 69
5.3.2 Time-Update for the Sum of Squares Errors 71
5.3.3 Implementation Considerations 74
5.4 QR Recursive Least Squares Method 76
5.4.1 Introduction 76
5.4.2 Preliminary Setup for QR-RLS Algorithm 77
5.4.3 Forming the QR-RLS Algorithm 82
5.4.4 Orthogonal Matrix Operation 85
5.4.5 Implementation Considerations 87
5.5 Results Summary 88

CHAPTER 6
ORDER-UPDATE AND SUBSET SELECTION 89
6.1 Introduction 90
6.2 Order-Update Algorithms 91
6.2.1 Block Matrix Inversion Lemma 92
6.2.2 Recursive Order-Update Algorithm for LS Method 93
6.2.2.1 Order-Update for the Parameter 94
6.2.2.2 Recursive Order-Update for the Sum of Squared Errors 96
6.2.2.3 Implementation Considerations 99
6.2.3 Recursive QR Order-Update Algorithm 101
6.2.3.1 QR Recursive Order-Update for Q, R, and Parameter .101

6.2.3.2 Implementation Considerations 104
6.3 Subset Selection 105
6.3.1 Background 105
6.3.2 Comparison of Subset Selection Methods 108
6.3.3 Forward Selection and Order-Update 108

CHAPTER 7
TIME- AND ORDER- UPDATE 111
7.1 Introduction 112
7.2 The Subset Selection Model 113
7.3 Recursive Time- and Order- Update 116
7.3.1 Recursive Least Squares with Automatic Weight Selection (RLS-AWS) 122
7.3.2 QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) 125
7.4 Fixing Centers 127
7.4.1 Centers Selects from Fixed Range/Grid 127
7.4.2 Centers Selected from Time Point 128
7.4.3 Centers Selection for the New Algorithms 128
7.5 Preliminary Results 129
7.5.1 Compare QR-RLS-AWS and RLS-AWS 130
7.5.2 Accuracy Test 131
7.5.3 Batch and Recursive Test 132
7.6 Summary 133

CHAPTER 8
RLS-AWS ALGORITHM IMPROVEMENT 134
8.1 Introduction 135
8.2 Alleviate the Storage Requirement 136
8.2.1 Time-Update Correlation Matrix 136
8.2.2 Improvement of the Forward Selection Method 139
8.2.3 Restructuring Time-Update Correlation Matrix 141
8.3 Order-Decrease-Update Algorithms 142
8.3.1 Block Matrix Inversion Lemma for Matrix Downdating 144
8.3.2 Recursive Order-Decrease-Update Algorithm for the LS Method 147
8.3.2.1 Recursive Order-Decrease-Update for the Parameter .. 149
8.3.2.2 Recursive Order-Decrease-Update for the Sum of Squared Errors 150
8.4 Recursive Backward Elimination 154
8.5 Recursive Efroymson Algorithm 155
8.5.1 Batch Efroymson Algorithm 155
8.5.2 Recursive Efroymson Algorithm 157
8.5.2.1 Stopping Rule 157

8.5.2.2 Restructuring the Time-Update Correlation Matrix 158
8.5.3 RLS-AWS Algorithm: Efroymson Method 161
8.5.3.1 Subset Selection with Efroymson Method 162
8.6 Implementation Consideration 167
8.6.1 Exponential Windowing 167
8.6.2 Reduce Computational Time 169
8.7 Summary 170

CHAPTER 9
QR-RLS-AWS ALGORITHM IMPROVEMENT 171
9.1 Introduction 172
9.2 Square Root Error Reduction Term 172
9.3 The New QR-RLS Structure 176
9.4 Recursive QR-Order-Update Algorithms 179
9.4.1 Recursive QR-Order-Increase-Update 180
9.4.2 Recursive QR-Order-Decrease-Update 182
9.5 Recursive Subset Selection Algorithms 186
9.5.1 Recursive QR Forward Selection Method 186
9.5.2 Recursive QR Backward Elimination Method 188
9.5.3 Recursive QR Efroymson Algorithm 190
9.6 Implementation Considerations 194
9.6.1 Exponential Windowing 195
9.6.2 Reduce Computational Time 195
9.7 Summary 196

CHAPTER 10
NUMERICAL TESTING AND APPLICATIONS 197
10.1 Introduction 198
10.2 Numerical Stability of the Algorithms 199
10.3 Applications 202
10.3.1 Chaotic Time Series 202
10.3.1.1 Recursive Forward Selection Method and Recursive Efroymson Method 203
10.3.1.2 Effects of the Stopping Rules 206
10.3.1.3 Summary 208
10.3.2 1-D Function Approximation 209
10.3.3 2-D Function Approximation 212
10.3.3.1 Comparison of Batch Forward Selection Method and Recursive Efroymson Method 213
10.3.4 Magnetic Levitation System 216
10.3.4.1 On-line Adaptation Results 217
10.3.4.2 Comparison to Multilayer Feedforward Network 221
10.4 Summary 222

CHAPTER 11
SUMMARY AND CONCLUSIONS 223
11.1 Research Summary 224
11.2 Conclusions...226

REFERENCES
REFERENCES 227

LIST OF TABLES

Table Page

10 -1 Numerical Test Results 199

10 - 2 Solution of Improved QR-RLS-AWS Algorithm After X-times Order-Update ..200

10 - 3 Sum of Squared Errors Comparison between the Recursive Efroymson Method and the Batch Forward Selection Method 214

LIST OF FIGURES

Figure Page

2 - 1 Biological Neurons 10

2 - 2 Biological Neuron to Artificial Neuron 11

2 - 3 Simplified Representation of an Artificial Neuron 12

2 - 4 Three Typical Transfer Functions 13

2 - 5 Multiple-Input Neuron 14

2 - 6 Multiple-Input Neuron - Simplified Representation 14

2 - 7 A Layer of Neurons 15

2 - 8 Two-Layer Network 16

2 - 9 Two-Layer Feedforward Network - Simplified Representation 17

2 - 10 Radial Basis Function Node 19

2 - 11 The Radial Basis Function Network 21

2 - 12 Special RBF Network for Subset Selection 35

3 - 1 Desired Target and the RBF Hidden Layer Outputs 43

3 - 2 Orthogonal Least Squares Centers Selection 44

4 - 1 Geometrical Interpretation of the Orthogonal Projection 55

4 - 2 Givens QR Annihilation on a 4x3 Matrix 61

5 - 1 Givens Rotations Applied to the Pre-Array in the QR-RLS Algorithm 86

5 - 2 QR-RLS Algorithm for Next Iteration 86

Figure Page

6 - 1 Ideal Subset Selection 105

7 - 1 The Subset Selection Model 113

7 - 2 The Time- and Order- Update Algorithm Flow Chart 116

7 - 3 The QR-RLS-AWS and RLS-AWS algorithms Result 130

7 - 4 RLS-AWS Algorithm Blow Out 131

7 - 5 QR-RLS-AWS Algorithm No Blow Out 132

7 - 6 Batch OLS algorithm 133

8 - 1 Flow Chart for RLS-AWS with Efroymson Method 163

8 - 2 System Trajectory Travels over the RBF Nodes Planted in 2-D Spaces 170

9 - 1 The Obtainable Subsets Given a Four-Nodes Linear Model 174

9 - 2 Flow Chart for QR-RLS-AWS with Efroymson Method 189

10 - 1 Input-Output of Logistic Map and the Potential Nodes of the RBF Network 203

10 - 2 Comparison of Efroymson Method and Forward Selection Method 205

10 - 3 Effects of F-to-enter and F-to-delete 207

10 - 4 The Target Function and the Input Patterns 209

10 - 5 1-D Function Approximation Results for Different Smoothing Factor 211

10 - 6 A Comparison of the Errors 212

10 - 7 Surface Function 213

10 - 8 Errors Comparison for a 82 Nodes RBF Network Constructed by Recursive Efroymson Algorithm and Batch Forward Selection Method 214

10 - 9 Magnetic Levitation System 216

10 - 10 System Identification Scheme 217

Figure Page

10 - 11 Input Sequence and Output of the Plant 218

10 - 12 Result of On-Line Adaptation After 80 Time Point 219

10 - 13 Results of On-Line Adaptation After 10000 Time Point 220

10 - 14 Number of Selected Nodes and Tracking Error 220

10 - 15 Error Comparison: The Trained RBF Network and Multilayer Network 222

Chapter 1

Introduction

1.1 Objective 2
1.2 Contributions 3
1.3 Outline 5

In this chapter, we first discuss the objectives of this research. Then, we summarize the contributions of this research. We distinguish between what is new and what was developed previously. Finally, we will outline the contents of each chapter.

1.1 Objective

This research addresses a problem commonly associated with the radial basis function (RBF) network. This problem is called the curse of dimensionality; the number of RBF nodes increases exponentially with the number of inputs. Due to this problem, RBF networks can only be used in models with low dimensional inputs. Many excellent methods have been proposed that have successfully reduced the number of nodes used in RBF networks. However, most of these methods are not suitable for online implementation. Online construction of small RBF networks is especially desirable for adaptive control, adaptive filtering and system identification of nonlinear systems.

Hence, this research focuses on developing online learning schemes that can construct small and parsimonious RBF networks. We have shown in this research that this goal can be achieved by modifying the off-line least squares learning method (LS) and the off-line orthogonal least squares (OLS) learning method for on-line operation combined with the subset selection techniques. These modifications have resulted in the development of a time- and order- update framework. Using this framework, two new recursive time- and order- update algorithms, the Recursive Least Squares with Automatic Weight Selection (RLS-AWS) algorithm and the QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS), have been developed. The theoretical framework and the synthesis of these on-line learning schemes are documented from Chapter 4 to Chapter 9.

1.2 Contributions

In the following, we summarize the new contributions from this research. This summary distinguishes what is new and what was developed previously.

- The main contribution of this research is the time- and order- update framework. This framework is adopted by combining three schemes: time-update, order-update, and subset selection. Using this framework, two new recursive time- and order-update algorithms, the RLS-AWS and the QR-RLS-AWS, are mathematically derived.
- Initially, both algorithms utilize the recursive forward selection method as their subset selection mechanism. Later, we have improved the algorithm's subset selection solution by developing the recursive Efroymson method for the RLS-AWS algorithm and the QR-RLS-AWS algorithm. These techniques also have not been documented anywhere.
- When numerical ill conditioning is not an issue, both algorithms yield the same solution. However, when numerical ill conditioning becomes a problem, we have shown that the QR-RLS-AWS algorithm yields a much more accurate solution than the RLS-AWS algorithm.
- We have applied these algorithms to the RBF network. Results have shown that these algorithms can effectively construct a small RBF network while operating in real-time.

- The time-update scheme involves the RLS algorithm and the QR-RLS algorithm, which are readily available from adaptive filtering theory (Haykin 1996, Sayed & Kailath 1992).
- Meanwhile, the least squares order-update scheme is mathematically derived based on block matrix inversion lemma. These derivations are tailored to order-increase-update and/or order-decrease-update the parameters in the RLS algorithm and it has not been documented anywhere.
- In addition, the orthogonal least squares order-update scheme based on the QR Givens rotations is mathematically derived. These derivations are tailored to order-increase-update and/or order-decrease-update the parameters in the QR-RLS algorithm and it has not been documented anywhere.

Although our primary interest has been in the application of these algorithms to the RBF network, the algorithms are general purpose recursive subset selection algorithms. They can be used for any linear-in-parameters model for which recursive subset selection is needed.

1.3 Outline

This thesis contains eleven chapters. Starting from the basic neural network building block, the artificial neural network architectures are introduced in Chapter 2. Then, a class of neural network architecture, the radial basis function (RBF) network, is discussed in detail along with frequently used mathematical notation.

Chapter 3 defines the scope and the objective of this research. We first discuss the importance of the on-line learning scheme using the RBF network. Then, we develop online learning schemes, based upon the off-line least squares (LS) and off-line orthogonal least squares (OLS) learning method, that can efficiently construct small RBF networks. These off-line LS and OLS learning methods are made on-line by employing the time-update and the order-update algorithms of the least squares and orthogonal least squares methods.

To understand these new algorithms, Chapter 4 reviews the least squares and the orthogonal least squares methods. It also discusses the necessary tools for solving the time-update and the order-update.

Then, the concept of time-update is introduced in Chapter 5. In this chapter, the necessary tools developed in Chapter 4 are used in developing the recursive least squares (RLS) algorithm and the numerically more accurate QR recursive least squares (QR-RLS) algorithm.

In Chapter 6, we first develop the order-update algorithms. Then, the concept of subset selection is introduced. Among these subset selection methods, we discuss the orthogonal least squares learning method (also called the forward selection method) in

detail. Later, we discuss how we can use the order-update algorithms in the forward selection method.

In Chapter 7, the time-update and forward subset selection method, which utilizes the order-update algorithm, are combined to develop the Recursive Least Squares with Automatic Weights Selection (RLS-AWS) and QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) algorithms. Detailed implementations of these new algorithms for the RBF networks are discussed. Several simple simulated results are shown to illustrate the performance and the node saving ability of the new algorithms.

In Chapter 8, we focus on improving the RLS-AWS algorithm. These improvements include alleviating the storage requirement, improving the algorithm's subset selection solution by developing the recursive Efroymson algorithm, and reducing the computation.

In Chapter 9, we focus on improving the QR-RLS-AWS algorithm. A new discovery has led us to rework the QR-RLS-AWS method in Chapter 7. Detailed discussion of this new scheme is derived and discussed. Then, we show how we can improve the algorithm's subset selection solution by developing the recursive QR-Efroymson algorithm.

In Chapter 10, we will discuss the numerical stability of the algorithms developed in Chapter 7, 8 and 9. Then, we will test these algorithms in two function approximation problems and two system identification problems.

Finally, Chapter 11 summarizes our research and lists some of the key results of this research.

Chapter
2

Network Architectures

2.1 Introduction 8
2.2 Basic Neural Network Architectures 9
2.2.1 Biological Neural Networks 9
2.2.2 Artificial Neural Networks 11
2.2.3 Single Artificial Neuron 11
2.2.4 Transfer Function 12
2.2.5 Multiple-Input Neuron 13
2.2.6 A Layer of Neurons 14
2.2.7 Multilayer Network 16
2.2.8 Universal Approximation Capability 17
2.3 The Radial Basis Function Network 19
2.3.1 The Radial Basis Functions 19
2.3.2 The RBF Network Architecture 21
2.3.3 Universal Approximation Capability 22
2.4 Mathematical Preliminaries 24
2.4.1 RBF Network - Linear in Parameters 24
2.4.2 Time-Update Framework 27
2.4.3 Order-Update Framework 29
2.4.4 Time and Order Update Framework 32
2.5 Special RBF Network for Subset Selection 35
2.6 Summary 37

This chapter introduces the relevant neural network architectures and frequently used mathematical notation. Readers are encouraged to pay special attention to section 2.4, and section 2.5 as it defines the framework of a class of neural network architecture, which will be used in the entire thesis.

2.1 Introduction

In section 2.2, we show how an artificial neuron evolves from a biological neuron to a single-layer of neurons then to multiple-layers of neurons (feedforward network). We will also define the mathematical notation and the symbolic representation of the feedforward network. Then, a short section is given to show the universal approximation capability of the feedforward network. Section 2.3 introduces the Radial Basis Function (RBF) network and ties it into the feedforward network architecture. This section also shows that the RBF network possesses the same universal approximation capability as the feedforward network. In section 2.4, we begin to discuss a class of RBF network, which we will use in later chapters. Special attention is given to the mathematical notation that describes how the RBF network architecture changes for time-updates and/or order-updates. Lastly, section 2.5 will discuss the architecture of the special RBF network for subset selection, which will be the key neural network architecture of this entire thesis.

2.2 Basic Neural Network Architectures

We will begin with a biological neuron and show how it is engineered to become an artificial neuron and a multiple-input neuron. Then, we provide a brief introduction to the neuron's transfer functions. After this brief introduction, we show how several artificial neurons are engineered to become a layer of neurons, and later to a multiple-layer network. At each engineering stage, we show how the artificial neural network mimics the biological neural network counterparts. Simultaneously, mathematical notation and representations are introduced. Lastly, we discuss the function approximation capabilities of a multilayer perceptron.

2.2.1 Biological Neural Networks

The work on artificial neural networks is inspired by the studies of how the human brain processes information; more importantly, the information processing nerve cell called the neuron. The struggle of understanding how the brain operates owes much to the pioneering work of Ramon & Cajal (1911), who first introduces the idea of neurons.

2.2.1.1 Single Biological Neuron

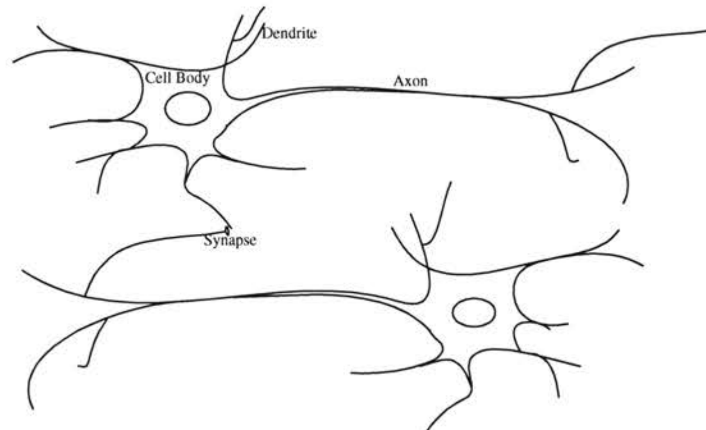


Figure 2 - 1 Biological Neurons

It has been understood that each neuron consists of four parts; dendrites, synapses, cell body and axons. The junction point between the dendrites and the cell body are the synapses. When a neuron is at work, it receives input signals, which are the electrical signals, from the axons of adjacent neurons to the dendrites. Then, these inputs are modulated by the complex chemical process in the synapses, which carry it into the cell body. The cell body sums and thresholds all the modulated incoming electrical signals and passes them on to the axon. The axon, a single long fiber, then carries the outgoing electrical signal from the cell body to the other neurons.

2.2.1.2 Massive Interconnections and Parallel Structure

There are billions of biological neurons in the brain and each neuron has massive interconnections with adjacent neurons. Although biological neurons are several orders of magnitude slower than silicon logic gates, the brain makes up for this relatively slow operating rate by having a massive parallel structure and massive interconnections between

neurons. Because of this massive parallel structure, all neurons can operate at the same time, which enables the brain to perform many tasks faster than any conventional computer.

2.2.2 Artificial Neural Networks

Since the brain is capable of such massive information processing, engineers and mathematicians mimic the brain by developing the artificial neural network. They start by imitating one biological neuron with a single dendrite, cell body and axon.

2.2.3 Single Artificial Neuron

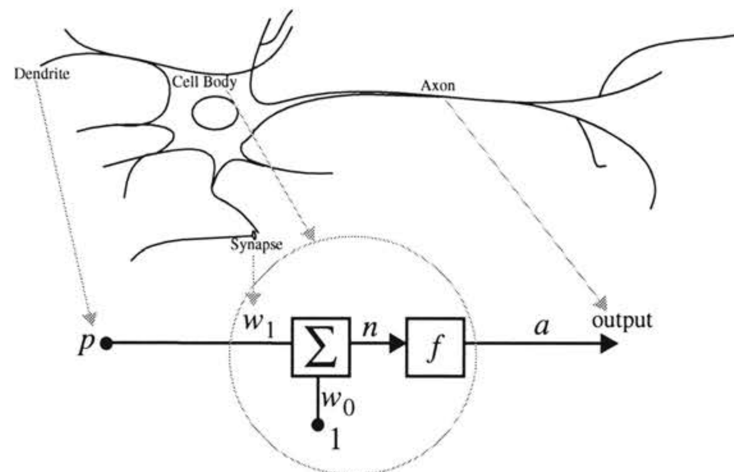


Figure 2 - 2 Biological Neuron to Artificial Neuron

Figure 2 - 2 shows a single-input/single-output artificial neuron in symbolic representation that mimics a biological neuron with one dendrite and one axon. The input, p, is multiplied by a scalar artificial weight, w_1, to form $w_1 p$, which imitates the electrical signal modulated by the synaptic weight. Then, the weighted input, $w_1 p$, is sent to a summer $\sum$ to sum an externally applied bias w_0, which imitates the modulated electrical

signal carried by the dendrite. The summer $\sum$ and the transfer function f closely resemble the cell body, which has the effect of summing and thresholding the modulated electrical signal. After the weighted input, $w_1 p$, and bias, w_0, are processed by the summer and the activation function, it is sent to the output a, which represents the electrical signal carried by the axon.

Mathematically, an artificial neuron can be described by the following equation:

$$a = f(w_1 p + w_0) \tag{2 - 1}$$

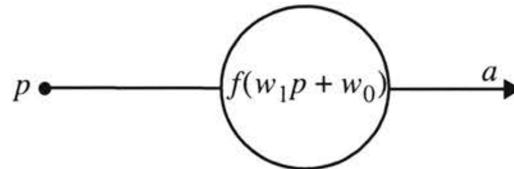


Figure 2 - 3 Simplified Representation of an Artificial Neuron

We will use a simplified symbolic representation, as shown in Figure 2 - 3, to denote an artificial neuron. This representation models how the dendrite and the axon interconnect to the cell body in a biological neuron. Specifically, the input p represents the dendrites, the output a represents the axon, and the transfer function $f(w_1 p + w_0)$ is represented by a node, which mimics the cell body.

2.2.4 Transfer Function

The transfer function, denoted by $f(\bullet)$ in Figure 2 - 2 and Figure 2 - 3, defines the output of a neuron. A particular transfer function is chosen by a designer to perform a particular task. Three of the most common transfer functions are hyperbolic tangent sigmoidal, hard limiter and linear. Figure 2 - 4 shows these typical transfer functions:

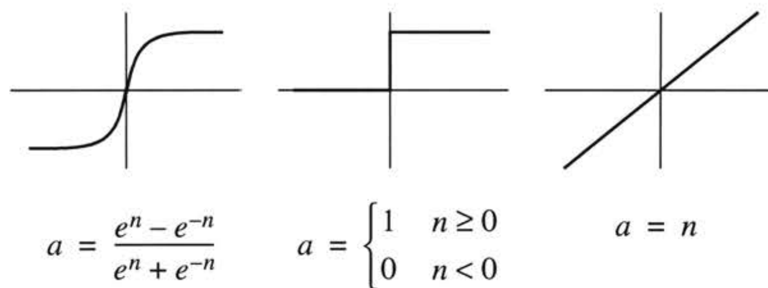


Figure 2 - 4 Three Typical Transfer Functions

The hyperbolic tangent transfer function is shown on the left of Figure 2 - 4. This transfer function is commonly used in multilayer perceptron networks, because it is a monotonically increasing function and it is differentiable. In the center, we have the hard limiting transfer function. Neurons that use this transfer function are commonly referred to as McCulloch-Pitts neurons. Since the output has the value of 1 or 0, it is commonly used for binary classification. Lastly, a linear transfer function is shown on the right. This transfer function is commonly used in the last layer of a feedforward network for function approximation applications.

The reader can refer to (Hagan *et al.* 1996, Haykin 1994) for a list of other transfer functions. Note that from now on, we will use "neuron" for artificial neuron and "neural network" for artificial neural network.

2.2.5 Multiple-Input Neuron

To mimic the multiple dendrite connections in a biological neuron, a neuron with multiple inputs is illustrated in Figure 2 - 5. As shown, the neuron has r inputs $p_1\ p_2\ \dots\ p_r$ weighted by r weights $w_{1,1}\ w_{1,2}\ \dots\ w_{1,r}$:

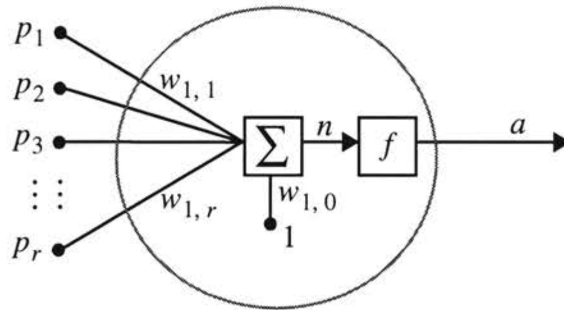


Figure 2 - 5 Multiple-Input Neuron

Mathematically, if we assume that the bias term is weighted by a constant input of 1, $p_0 = 1$, that is $\mathbf{p} = \begin{bmatrix} p_0 \; p_1 \; p_2 \; \dots \; p_r \end{bmatrix}^T$ and ${}_1\mathbf{w} = \begin{bmatrix} w_{1,0} \; w_{1,1} \; w_{1,2} \; \dots \; w_{1,r} \end{bmatrix}$, then the output is

$$a = f({}_1\mathbf{wp}). \tag{2 - 2}$$

Using Eq. (2 - 2), we can draw the simplified symbolic representation of a multiple-input neuron as in Figure 2 - 6.

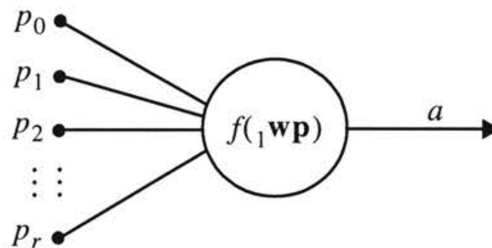


Figure 2 - 6 Multiple-Input Neuron - Simplified Representation

2.2.6 A Layer of Neurons

It is apparent that a biological neural network derives its computing power through its massive parallel structure and massive interconnections. To mimic the massive parallel biological structure, we can connected several multiple-input neurons in parallel. This

forms a layer of neurons, which operate in parallel. Figure 2 - 7 shows a single-layer of s neurons. As shown, the inputs are interconnected to each neuron forming the parallel structure.

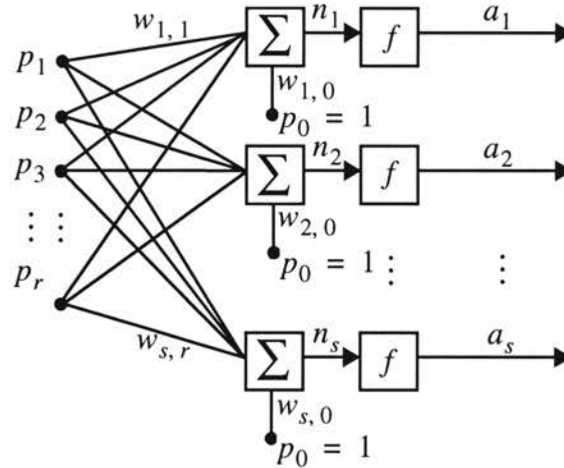


Figure 2 - 7 A Layer of Neurons

We can express the output of a layer of neurons using vectors and matrices:

$$\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p}) \tag{2 - 3}$$

The weights and biases are lumped into one weight matrix $\mathbf{W}$, as in

$$\mathbf{W} = \begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} & \cdots & w_{1,r} \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots & w_{2,r} \\ \vdots & \vdots & \vdots & & \vdots \\ w_{s,0} & w_{s,1} & w_{s,2} & \cdots & w_{s,r} \end{bmatrix} \tag{2 - 4}$$

where the row indices indicate the number of neurons, s, and the column indices indicate the number of inputs, r. Meanwhile, the input $\mathbf{p}$ is expressed in vector form as

$$\mathbf{p} = \begin{bmatrix} p_0 & p_1 & p_2 & \dots & p_r \end{bmatrix}^T . \tag{2 - 5}$$

Multiplied together, $\mathbf{Wp}$ forms the net input vector $\mathbf{n}$. The transfer function $\mathbf{f}(\bullet)$ then processes the net input vector element by element to form the output vector $\mathbf{a}$.

2.2.7 Multilayer Network

Neural networks achieve the massive interconnections of the biological neurons by cascading several layers of neurons. Typically, a network with multiple-layers of neurons is called a multilayer feedforward network. Each layer has its own weights, biases, net input and output. To distinguish between layers, a superscript is used to identify the layer number. For example, $\mathbf{W}^1$ is the weight and bias matrix for the first layer. Figure 2 - 8 shows two layers of neurons (two-layer feedforward network) with r inputs, s^1 neurons in the first layer and s^2 neurons in the second layer. If a network has more than 1 layer, we refer to the layer in between the input and output layer as the hidden layer.

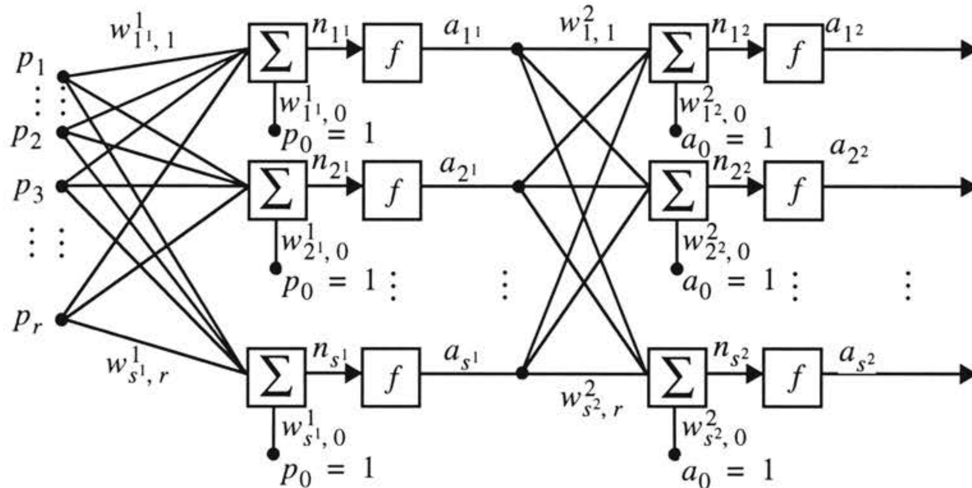


Figure 2 - 8 Two-Layer Network

A mathematical equation that describes the total output of the two-layer feedforward network is given by

$$\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p})) \quad . \qquad (2 - 6)$$

The simplified symbolic representation can also be used in modeling a multilayer feedforward network. Figure 2 - 9 shows a simplified representation of Figure 2 - 8. This representation has the advantage of showing how each node is linked to the inputs and the outputs. This will make it easier to demonstrate the relationship between the multilayer network and the radial basis network, which will be presented later. Note that p_0, and a_0 are the biases.

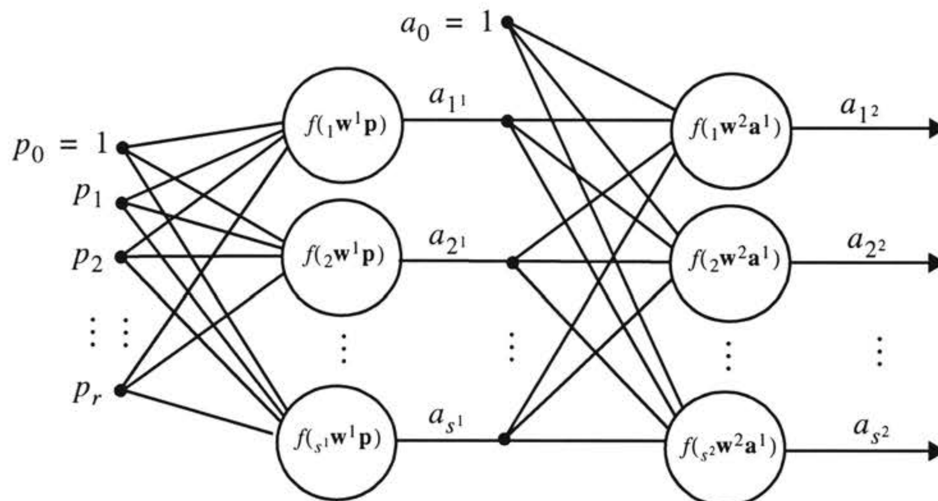


Figure 2 - 9 Two-Layer Feedforward Network - Simplified Representation

2.2.8 Universal Approximation Capability

One of the key features of the neural network that attracts many researchers is its universal approximation capability. According to Hornik's (1989 & 1991) universal approximation theorem, a multilayer feedforward neural network, with one or more hidden layers of squashing nonlinear transfer functions, is capable of approximating any real-valued continuous function arbitrarily well over a compact interval provided that sufficient

hidden neurons are available. Independently, Funahashi (1989) and Cybenko (1989) arrived at the same neural network universal approximation capability using functional analysis.

The term "squashing function" refers to a class of transfer functions, which includes tangent sigmoidal, hyperbolic tangent and more. Later, Leshno *et al.* (1993) showed that a locally bound piecewise continuous transfer function also has the universal approximation capability. An example of such a function is the linear saturation function. Thus, neural networks such as the multilayer perceptrons, which use tangent sigmoidal transfer functions in the hidden layers and linear transfer functions in the output layer, are universal approximators that can approximate any continuous function to an arbitrary degree of accuracy.

2.3 The Radial Basis Function Network

The radial basis function network was introduced by Powell (1987a, 1987b) for multidimensional interpolation. Then, it was exploited by Broomhead & Lowe (1988) in the context of neural network design. In the following, we introduce the fundamental building block of the RBF network - the radial basis functions.

2.3.1 The Radial Basis Functions

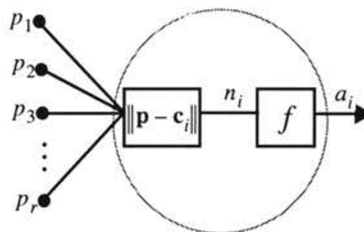


Figure 2 - 10 i^{th} Radial Basis Function Node

Rather than using monotonically increasing transfer functions such as the tangent sigmoidal, the RBF network utilizes radial basis functions as the transfer functions. Figure 2 - 10 shows the mathematical operation of a radial basis function node. As shown, the Euclidean distance between the input vector $\mathbf{p}$, and the center vector $\mathbf{c}_i$ of the ith radial basis function is first computed to form the net input n_i of ith radial basis function. Then, the net input is fed into the nonlinear radial basis function $f(\bullet)$. Typically, this radial basis function is also called the local receptive field, since it only activates if the distance is close to the centers. The following are two radial basis functions often used in practice (Broomhead & Lowe 1988; Poggio & Girosi 1990a).

Thin plate spline function:

$$f(n) = \left(\frac{n}{\sigma}\right)^2 \ln\left(\frac{n}{\sigma}\right), \text{ for } \sigma > 0 \text{ and } n \geq 0 \qquad (2 - 7)$$

Gaussian function:

$$f(n) = \exp\left(-\frac{n^2}{2\sigma^2}\right), \text{ for } \sigma > 0 \qquad (2 - 8)$$

Theoretical investigations and practical results have shown that the type of radial basis functions are not crucial to the performance of the RBF networks. Hence, this research will confine our discussion to the use of the Gaussian function only. In the Gaussian function, σ is the standard deviation, and it determines the width of the Gaussian function. It is sometimes referred to as the smoothing factor.

2.3.2 The RBF Network Architecture

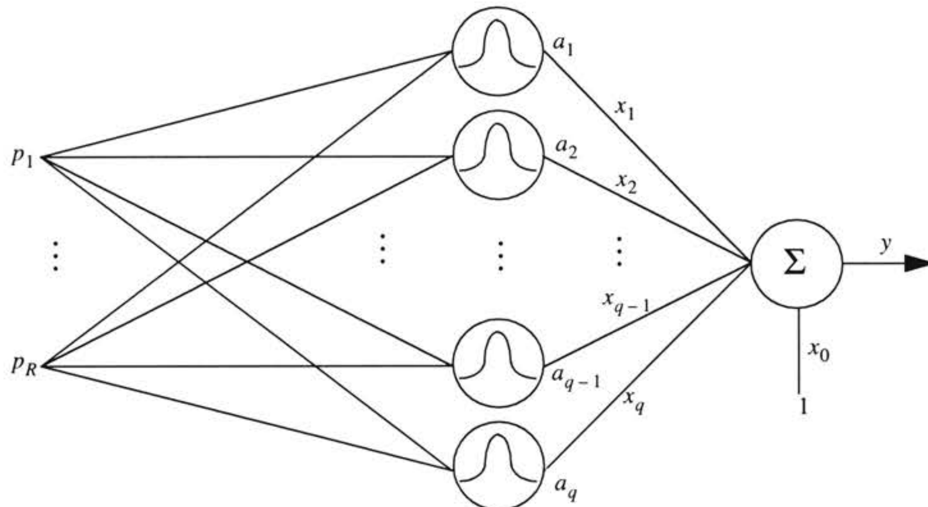


Figure 2 - 11 The Radial Basis Function Network

In terms of the network architecture, the RBF network is a two-layer feedforward network. In fact, the two-layer network architecture as shown in Figure 2 - 9 is the multi-input multi-output RBF network architecture. However, since multi-output RBF networks can always be separated into several single-output RBF networks, we will only consider the multi-input single-output RBF network. Figure 2 - 11 shows such a Radial Basis Function (RBF) network architecture. The main difference between a RBF network and a two-layer perceptron is its hidden layer transfer function. The RBF network uses the radial basis functions discussed in section 2.3.1. Also, the hidden layer has no bias. As in the feedforward network, the RBF network uses linear transfer functions in the output layer.

The following pair of mathematical equations represents a RBF network with r inputs, q hidden nodes and a scalar output:

$$a_0 = 1 \, , \; a_i = f(\|\mathbf{p} - \mathbf{c}_i\|) \, , \text{ and } y = \sum_{i=0}^{q} x_i a_i \qquad (2 - 9)$$

where

$\mathbf{p} = \begin{bmatrix} p_1 \; p_2 \; \ldots \; p_r \end{bmatrix}^T$ is the input vector,

$\mathbf{c}_i = \begin{bmatrix} c_{1,i} \; c_{2,i} \; \ldots \; c_{r,i} \end{bmatrix}^T$ is the ith RBF center vector,

$\| \bullet \|$ denotes the Euclidean norm,

$a_0 = 1$ is the second layer bias input,

a_i is the ith Gaussian radial basis function output except a_0,

x_i is the ith weight, and x_0 is the bias,

y is the output of the RBF network, and

$f(\bullet)$ is the Gaussian radial basis function.

2.3.3 Universal Approximation Capability

Like the multilayer perceptron, the universal approximation theorem is also available for the RBF network. Park & Sandberg (1991) show that by using a fixed smoothing factor in all the radially symmetric kernel functions, a RBF network with such kernel functions in the hidden layer is broad enough for universal approximation. Furthermore, Poggio & Girosi (1990b) show that a regularization RBF network has the best approximation property in addition to the universal approximation ability. This means that given an unknown nonlinear function y_f, there always exists a choice of coefficients that approximates y_f better than all other possible choices.

These existing theorems show that the RBF networks are universal approximators and can approximate any continuous function with as much accuracy as the multilayer network.

2.4 Mathematical Preliminaries

In this section, we lay down the fundamental mathematical notation for the RBF network that we will be using in later chapters. The intention of this section is to allow readers to become familiar with the mathematical notations and symbols of the RBF network architectures that we will be frequently using.

2.4.1 RBF Network - Linear in Parameters

Assume that we have a set of k input data $\{d(j), \mathbf{p}(j)\}_{j=0}^{k}$, where $d(j)$ is the desired response and $\mathbf{p}(j)$ is the network input. Then the RBF network output is

$$a_i(j) = f(\|\mathbf{p}(j) - \mathbf{c}_i\|)\text{, and } y(j) = \sum_{i=0}^{q} x_i a_i(j)\text{, } 1 \le j \le k. \tag{2 - 10}$$

For this research, we fix all the RBF centers $\mathbf{c}_i$ and the standard deviation σ in the hidden layer. Then there will be no unknown parameters in the hidden layer. Thus, the hidden layer performs a fixed nonlinear transformation with no adjustable parameters. The hidden layer output is

$$a_i(j) = f(\|\mathbf{p}(j) - \mathbf{c}_i\|)\text{, for } j = 0 \ldots k \text{ and } i = 0 \ldots q. \tag{2 - 11}$$

If we write out every element of $a_i(j)$, we form the following matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} a_0(0) & a_1(0) & \dots & a_q(0) \\ a_0(1) & a_1(1) & \dots & a_q(1) \\ \vdots & \vdots & & \vdots \\ a_0(k) & a_1(k) & \dots & a_q(k) \end{bmatrix}. \qquad (2 - 12)$$

where row j is the response of the first layer to input vector $p(j)$. In the future, we will refer to k as the current time step, since one new data vector will come into the network at each time step. We will refer to q as the model order, which refers to the number of neurons in the first layer. The $\mathbf{A}$ matrix is $k \times q$.

The error $e(j)$ between the desired response $d(j)$ and the RBF network output $y(j)$ is given as follows:

$$e(j) = d(j) - \sum_{i=0}^{q} x_i a_i(j), \text{ for } 0 \le j \le k. \qquad (2 - 13)$$

By defining

$$\mathbf{e} = \begin{bmatrix} e(0) & e(1) & \dots & e(k) \end{bmatrix}^T, \qquad (2 - 14)$$

$$\mathbf{d} = \begin{bmatrix} d(0) & d(1) & \dots & d(k) \end{bmatrix}^T, \text{ and} \qquad (2 - 15)$$

$$\mathbf{x} = \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_q \end{bmatrix}^T, \qquad (2 - 16)$$

we can form the error vector as

$$\mathbf{e} = \mathbf{d} - \mathbf{A}\mathbf{x}. \qquad (2 - 17)$$

In statistics, the above model is called the linear model, since the output of the RBF network is a linear combination of the weights $\mathbf{x}$, and the Gaussian node outputs $\mathbf{A}$. Hence, methods

used for solving the linear model can be applied to this linear-in-parameter RBF network. For the rest of this document, the terms "linear model" and "RBF network" refer to the same type of network. Note that Eq. (2 - 17) will be used in Chapter 4 for analysis of the batch least squares method and the batch orthogonal least square method.

The main thrust in this research is the calculation of the optimal linear parameters, x_j, in the RBF network. The optimal parameters are those that minimize the sum of squared errors $\mathbf{e}^T\mathbf{e}$.

In the remainder of this chapter we will develop notation that we will use in future chapters. This notation will be critical to the understanding of the four major problems addressed in this research: time-update, order-update, combined time- and order- update, and subset selection. The time-update (described in Chapter 5) is the process of updating the optimal linear parameters when a new data vector is received (k increased by 1). The order-update consists of two parts: the *order-increase-update* and the *order-decrease-update*. Initially, the order-update (described in Chapter 6), which we called order-increase-update later in Chapter 8 and Chapter 9, is developed for the recalculation of the optimal linear parameters when a new neuron is added to layer 1 (q is increased by 1). Later in Chapter 8 and Chapter 9, we introduce the order-decrease-update to work in conjunction to the order-increase-update. Note that the order-decrease-update is the recalculation of the optimal linear parameters when an existing neuron is deleted from layer 1 (q is decreased by 1). The combined time- and order- update is the process in which we perform both a time and an order update during the same time step. Keep in mind that the order-update part in

the combined time- and order- update can be order-increase-update only (discussed in Chapter 7) or the combined order-increase-update and order-decrease-update (discussed in Chapter 8 and Chapter9). The subset selection occurs before an order update. It is the process of selecting significant nodes or deselecting insignificant nodes in an RBF network. (Subset selection is described in Chapter 6.)

Our notation will be slightly different for each of the four problems discussed above. The objective will be to minimize the amount of redundant notation required for a specific problem.

2.4.2 Time-Update Framework

Let us assume that in addition to the current set of k time points, $\{d(j), \mathbf{p}(j)\}_{j=0}^{k}$, we receive new data $\{d(k+1), \mathbf{p}(k+1)\}$, and we would like to update the linear model in Eq. (2 - 17). Then, the RBF hidden layer output becomes

$$a_i(j) = f(\|\mathbf{p}(j) - \mathbf{c}_i\|), \text{ for } j = 0 \ldots k+1 \text{ and for } i = 0 \ldots q. \tag{2 - 18}$$

The matrix in Eq. (2 - 12) will have an extra row appended to it. Hence, the old hidden layer output matrix is denoted by $\mathbf{A}(k)$,

$$\mathbf{A}(k) = \begin{bmatrix} a_0(0) & a_1(0) & \ldots & a_q(0) \\ a_0(1) & a_1(1) & \ldots & a_q(1) \\ \vdots & \vdots & & \vdots \\ a_0(k) & a_1(k) & \ldots & a_q(k) \end{bmatrix}, \tag{2 - 19}$$

while the new hidden layer matrix is denoted by $\mathbf{A}(k+1)$,

$$\mathbf{A}(k+1) = \begin{bmatrix} a_0(0) & a_1(0) & \dots & a_q(0) \\ a_0(1) & a_1(1) & \dots & a_q(1) \\ \vdots & \vdots & & \vdots \\ a_0(k) & a_1(k) & \dots & a_q(k) \\ a_0(k+1) & a_1(k+1) & \dots & a_q(k+1) \end{bmatrix}. \qquad (2 - 20)$$

Furthermore, by applying the time-indexing notation to Eq. (2 - 17), we obtain the old error vector:

$$\mathbf{e}(k) = \mathbf{d}(k) - \mathbf{A}(k)\mathbf{x}(k) \qquad (2 - 21)$$

where

$$\mathbf{e}(k) = \left[e(0)\ e(1)\ \dots\ e(k)\right]^T, \qquad (2 - 22)$$

$$\mathbf{d}(k) = \left[d(0)\ d(1)\ \dots\ d(k)\right]^T, \qquad (2 - 23)$$

$$\mathbf{x}(k) = \left[x_0(k)\ x_1(k)\ \dots\ x_q(k)\right]^T. \qquad (2 - 24)$$

The updated error after the new data is incorporated will be

$$\mathbf{e}(k+1) = \mathbf{d}(k+1) - \mathbf{A}(k+1)\mathbf{x}(k+1) \qquad (2 - 25)$$

where

$$\mathbf{e}(k+1) = \left[\tilde{e}(0)\ \tilde{e}(1)\ \dots\ \tilde{e}(k)\ \tilde{e}(k+1)\right]^T, \qquad (2 - 26)$$

$$\mathbf{d}(k+1) = \left[d(0)\ d(1)\ \dots\ d(k)\ d(k+1)\right]^T, \qquad (2 - 27)$$

$$\mathbf{x}(k+1) = \left[x_0(k+1)\ x_1(k+1)\ \dots\ x_q(k+1)\right]^T. \qquad (2 - 28)$$

Note that $\mathbf{x}(k+1)$ minimizes $\mathbf{e}^T(k+1)\mathbf{e}(k+1)$, whereas $\mathbf{x}(k)$ minimizes $\mathbf{e}^T(k)\mathbf{e}(k)$. Using this time-indexing notation, we will derive the recursive time-update algorithms, which we will discuss in Chapter 5.

2.4.3 Order-Update Framework

In the order-update framework, we will consider the order-increase-update and the order-decrease-update. The order-increase-update is the recalculation of the optimal linear parameters when a new neuron is added to the RBF network and the order-decrease-update is the recalculation of the optimal linear parameters when an existing neuron is deleted from the RBF network.

2.4.3.1 Order-Increase-Update

Suppose we want to add a neuron (Gaussian node) to layer 1 of the RBF network. Then, the hidden layer output of the RBF network becomes

$$a_i(j) = f(\|\mathbf{p}(j) - \mathbf{c}_i\|), \text{ for } j = 0 \ldots k, \text{ and } i = 0 \ldots q+1. \qquad (2 - 29)$$

This also means that the new Gaussian node forms an extra column appended to Eq. (2 - 12). In terms of notations, will add a subscript q to a vector or matrix to denote the order-indexing. Hence, the hidden layer matrix at order q is denoted by $\mathbf{A}_q$,

$$\mathbf{A}_q = \begin{bmatrix} a_0(0) & a_1(0) & \ldots & a_q(0) \\ a_0(1) & a_1(1) & \ldots & a_q(1) \\ \vdots & \vdots & & \vdots \\ a_0(k) & a_1(k) & \ldots & a_q(k) \end{bmatrix}, \qquad (2 - 30)$$

and the hidden layer matrix at order $q+1$ is denoted by $\mathbf{A}_{q+1}$,

$$\mathbf{A}_{q+1} = \begin{bmatrix} a_0(0) & a_1(0) & \dots & a_q(0) & a_{q+1}(0) \\ a_0(1) & a_1(1) & \dots & a_q(1) & a_{q+1}(1) \\ \vdots & \vdots & & \vdots & \vdots \\ a_0(k) & a_1(k) & \dots & a_q(k) & a_{q+1}(k) \end{bmatrix}. \tag{2 - 31}$$

If we apply the order-indexing notation to the error vector of Eq. (2 - 17), we obtain:

$$\mathbf{e}_q = \mathbf{d} - \mathbf{A}_q \mathbf{x}_q \tag{2 - 32}$$

where

$$\mathbf{e}_q = \left[e(0)\ e(1)\ \dots\ e(k)\right]^T, \tag{2 - 33}$$

$$\mathbf{d} = \left[d(0)\ d(1)\ \dots\ d(k)\right]^T, \tag{2 - 34}$$

$$\mathbf{x}_q = \left[x_0\ x_1\ \dots\ x_q\right]^T. \tag{2 - 35}$$

Since the order-update cannot affect the desired response $\mathbf{d}$, there is no subscript q attached. Meanwhile, the updated error vector will satisfy the following

$$\mathbf{e}_{q+1} = \mathbf{d} - \mathbf{A}_{q+1} \mathbf{x}_{q+1} \tag{2 - 36}$$

where

$$\mathbf{e}_{q+1} = \left[\tilde{e}(0)\ \tilde{e}(1)\ \dots\ \tilde{e}(k)\right]^T, \tag{2 - 37}$$

$$\mathbf{d} = \left[d(0)\ d(1)\ \dots\ d(k)\right]^T, \tag{2 - 38}$$

$$\mathbf{x}_{q+1} = \left[\tilde{x}_0\ \tilde{x}_1\ \dots\ \tilde{x}_q\ x_{q+1}\right]^T. \tag{2 - 39}$$

Note that each element $\tilde{x}_i$ in the parameter vector $\mathbf{x}_{q+1}$ is updated, and it will not have the same value as x_i in the parameter vector $\mathbf{x}_q$. Also, because of the added Gaussian node, a new parameter x_{q+1} is created in the parameter vector $\mathbf{x}_{q+1}$. Note that each element $\tilde{e}(j)$ in $\mathbf{e}_{q+1}$ is the newly computed error, so it is different from $\mathbf{e}_q$. Using this order-indexing notation, we will derive the order-increase-update algorithms, which we will discuss in Chapter 6.

2.4.3.2 Order-Decrease-Update

Suppose we want to remove an existing neuron (assume that it is the νth neuron where ν is between 0 and q) from layer 1 of the RBF network. Then, the hidden layer output of the RBF network becomes

$$a_i(j) = f(\|\mathbf{p}(j) - \mathbf{c}_i\|), \text{ for } j = 0 \ldots k, \text{ and } i = 0 \ldots \nu - 1, \nu + 1 \ldots q. \qquad (2 - 40)$$

This also means that the new Gaussian node has the ν column removed. Because we have one less neuron, the order decreases by 1 to $q - 1$. Hence, the hidden layer matrix at order $q - 1$ is denoted by $\mathbf{A}_{q-1}$,

$$\mathbf{A}_{q-1} = \begin{bmatrix} a_0(0) & a_1(0) & \ldots & a_{\nu-1}(0) & a_{\nu+1}(0) & \ldots & a_q(0) \\ a_0(1) & a_1(1) & \ldots & a_{\nu-1}(1) & a_{\nu+1}(1) & \ldots & a_q(1) \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_0(k) & a_1(k) & \ldots & a_{\nu-1}(k) & a_{\nu+1}(k) & \ldots & a_q(k) \end{bmatrix}, \qquad (2 - 41)$$

If we apply the order-indexing notation to the error vector of Eq. (2 - 17), we obtain:

$$\mathbf{e}_{q-1} = \mathbf{d} - \mathbf{A}_{q-1}\mathbf{x}_{q-1} \qquad (2 - 42)$$

where

$$\mathbf{e}_{q-1} = \left[\tilde{e}(0)\ \tilde{e}(1)\ \dots\ \tilde{e}(k)\right]^T, \tag{2 - 43}$$

$$\mathbf{d} = \left[d(0)\ d(1)\ \dots\ d(k)\right]^T, \tag{2 - 44}$$

$$\mathbf{x}_{q-1} = \left[\tilde{x}_0\ \tilde{x}_1\ \dots\ \tilde{x}_{q-1}\right]^T. \tag{2 - 45}$$

Note that each element $\tilde{x}_i$ in the parameter vector $\mathbf{x}_{q-1}$ is updated, and it will not have the same value as x_i in the parameter vector $\mathbf{x}_q$. Also, because an existing Gaussian node is removed, there will be one less parameter in $\mathbf{x}_{q-1}$. Using this order-indexing notation method, we will derive the order-decrease-update algorithms, which we will discuss in Chapter 8 and Chapter 9.

2.4.4 Time and Order Update Framework

By combining the time-indexing and order-indexing notations described in section 2.4.2 and section 2.4.3, we can arbitrarily indicate the location of a vector or matrix according to time and according to order. For example, the hidden layer matrix at order index q and time index k is denoted by $\mathbf{A}_q(k)$,

$$\mathbf{A}_q(k) = \begin{bmatrix} a_0(0) & a_1(0) & \dots & a_q(0) \\ a_0(1) & a_1(1) & \dots & a_q(1) \\ & & & \\ a_0(k) & a_1(k) & \dots & a_q(k) \end{bmatrix} \quad {}_q\mathbf{a}^T(k) \tag{2 - 46}$$

$$\mathbf{a}_q(k)$$

We will express the last row vector and the last column vector of $\mathbf{A}_q(k)$ using the notation shown in Eq. (2 - 47), and Eq. (2 - 48).

$${}_q\mathbf{a}(k) = \left[a_0(k)\ a_1(k)\ \dots\ a_q(k)\right]^T, \tag{2 - 47}$$

$$\mathbf{a}_q(k) = \left[a_q(0)\ a_q(1)\ \dots\ a_q(k)\right]^T. \tag{2 - 48}$$

We can make up the matrix $\mathbf{A}_q(k)$ using the row vector of Eq. (2 - 47) or the column vector of Eq. (2 - 48):

$$\mathbf{A}_q(k) = \left[\mathbf{a}_0(k)\ \mathbf{a}_1(k)\ \dots\ \mathbf{a}_q(k)\right] = \left[{}_q\mathbf{a}(0)\ {}_q\mathbf{a}(1)\ \dots\ {}_q\mathbf{a}(k)\right]^T. \tag{2 - 49}$$

Again, if we apply the time and order-indexing notations to Eq. (2 - 17), we obtain:

$$\mathbf{e}_q(k) = \mathbf{d}(k) - \mathbf{A}_q(k)\mathbf{x}_q(k) \tag{2 - 50}$$

where

$$\mathbf{e}_q(k) = \left[e_q(0)\ e_q(1)\ \dots\ e_q(k)\right]^T, \tag{2 - 51}$$

$$\mathbf{d}(k) = \left[d(0)\ d(1)\ \dots\ d(k)\right]^T, \tag{2 - 52}$$

$$\mathbf{x}_q(k) = \left[x_0(k)\ x_1(k)\ \dots\ x_q(k)\right]^T. \qquad (2 - 53)$$

After an order-increase-update and a time-update, the new hidden layer matrix is:

$$\mathbf{A}_{q+1}(k+1) = \begin{bmatrix} a_0(0) & a_1(0) & \dots & a_q(0) & a_{q+1}(0) \\ a_0(1) & a_1(1) & \dots & a_q(1) & a_{q+1}(1) \\ \vdots & \vdots & & \vdots & \vdots \\ a_0(k) & a_1(k) & \dots & a_q(k) & a_{q+1}(k) \\ a_0(k+1) & a_1(k+1) & \dots & a_q(k+1) & a_{q+1}(k+1) \end{bmatrix}. \qquad (2 - 54)$$

Meanwhile, the updated error vector will satisfy the following

$$\mathbf{e}_{q+1}(k+1) = \mathbf{d}(k+1) - \mathbf{A}_{q+1}(k+1)\mathbf{x}_{q+1}(k+1) \qquad (2 - 55)$$

where

$$\mathbf{e}_{q+1}(k+1) = \left[e_{q+1}(0)\ e_{q+1}(1)\ \dots\ e_{q+1}(k)\ e_{q+1}(k+1)\right]^T, \qquad (2 - 56)$$

$$\mathbf{d}(k+1) = \left[d(0)\ d(1)\ \dots\ d(k)\ d(k+1)\right]^T, \qquad (2 - 57)$$

$$\mathbf{x}_{q+1}(k+1) = \left[x_0(k+1)\ x_1(k+1)\ \dots\ x_q(k+1)\ x_{q+1}(k+1)\right]^T. \qquad (2 - 58)$$

Note that the parameter vector $\mathbf{x}_{q+1}(k+1)$ is updated, and it will not be the same as the parameter vector $\mathbf{x}_q(k)$. Also, because of the added Gaussian node, a new parameter $x_{q+1}(k+1)$ is created in the parameter vector $\mathbf{x}_{q+1}(k+1)$. Similarly, $\mathbf{e}_{q+1}(k+1)$ is the newly computed error, so it is different than $\mathbf{e}_q(k)$.

2.5 Special RBF Network for Subset Selection

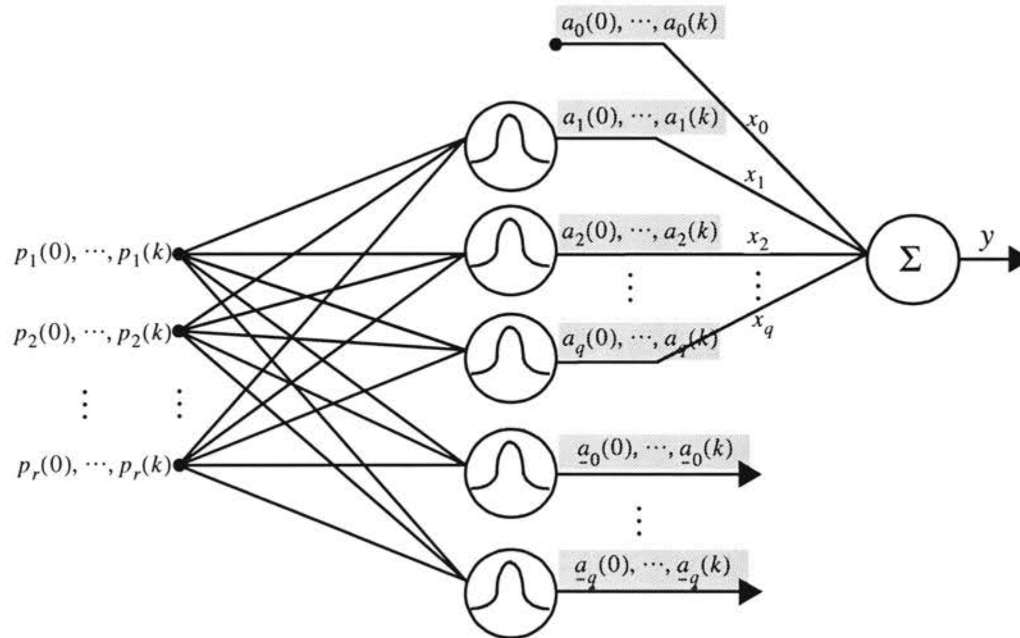


Figure 2 - 12 Special RBF Network for Subset Selection

In this section, we will define the special RBF network architecture that we will be using in Chapter 7. As shown in Figure 2 - 12, the architecture of this network consists of a RBF network and a subnet of Gaussian nodes with no output layer. We will assume that all the nonlinear parameters are fixed in both networks. The main idea here is to create a set of Gaussian nodes that are not used in the computation of the network output but can be made available if they are needed to improve the approximation ability of the network. Chapter 6 will explain how we choose which unused node to add to the computation of the network output.

The RBF network mathematical equation is described in Eq. (2 - 46) and Eq. (2 - 50). Meanwhile, the RBF network subnet without the output layer is described as

$$\underline{a}_i(j) = f(\|\mathbf{p}(j) - \mathbf{c}_i\|), \text{ for } j = 0 \dots k \text{ and for } i = 1 \dots \underline{q}. \tag{2 - 59}$$

We use an under bar to indicate the nodes associated with the subnet and a subscript to indicate the number of nodes in the subnet. If we write out every element, $\underline{a}_i(j)$, of the subnet this forms the following matrix:

$$\underline{\mathbf{A}}_q(k) = \begin{bmatrix} \underline{a}_1(0) & \underline{a}_2(0) & \dots & \underline{a}_q(0) \\ \underline{a}_1(1) & \underline{a}_2(1) & \dots & \underline{a}_q(1) \\ \vdots & \vdots & & \vdots \\ \underline{a}_1(k) & \underline{a}_2(k) & \dots & \underline{a}_q(k) \end{bmatrix} \quad {}_{\underline{q}}\mathbf{a}^T(k) \qquad \underline{\mathbf{a}}_q(k) \tag{2 - 60}$$

To denote the last row vector of $\underline{\mathbf{A}}_q(k)$, we introduce a left subscript. That is:

$${}_{\underline{q}}\mathbf{a}(k) = \left[\underline{a}_1(k) \; \underline{a}_2(k) \; \dots \; \underline{a}_q(k)\right]^T. \tag{2 - 61}$$

Meanwhile, the last column vector of $\underline{\mathbf{A}}_q(k)$ is denoted as

$$\underline{\mathbf{a}}_q(k) = \left[\underline{a}_q(0) \; \underline{a}_q(1) \; \dots \; \underline{a}_q(k)\right]^T. \tag{2 - 62}$$

$\underline{\mathbf{A}}_q(k)$ represents the unselected nodes and $\mathbf{A}_q(k)$ represents the selected nodes. Note that $\underline{\mathbf{A}}_q(k)$ and $\mathbf{A}_q(k)$ may have different numbers of columns.

2.6 Summary

Having presented the relevant neural network background and notation, we have built the framework for later chapters. For now, we will restrict our research to two-layer single-input/single-output RBF networks for function approximation. Although the work done in this research is based on the RBF network, it can be applied to all nonlinear models/ approximators that have a linear-in-parameters structure, such as the fuzzy basis function network, functional-linked network, Volterra series model and more.

Chapter 3

Problem Statement

3.1	**Introduction**	**39**
3.2	**Objective**	**39**
3.3	**Illustrative Example**	**42**
3.4	**The Research Outline**	**45**

RBF neural networks have always suffered from the curse of dimensionality; the number of RBF nodes increases exponentially with the number of inputs. This problem is especially acute when RBF networks are used in on-line control techniques such as stable adaptive control. Hence, this research explores on-line learning techniques that will construct small RBF networks. Many methods have been proposed to solve this problem, but most of these attempts are for off-line use. In fact, very few methods have been found that can efficiently construct small RBF network on-line. In this research, we design and implement on-line learning methods based upon the off-line least squares and orthogonal least squares learning methods. This chapter defines the problem addressed by this research.

3.1 Introduction

This research focuses on real-time adaptive control and identification of nonlinear systems using a class of radial basis function (RBF) networks. In real-time applications, such as stable adaptive control, a large number of RBF nodes are needed to guarantee a minimum network reconstruction error. This limits the use of this technique to systems with low input dimension. The RBF network suffers from the curse of dimensionality; the number of nodes needed increases exponentially with the number of inputs. We would like to construct small RBF networks in real-time, while guaranteeing the minimum network reconstruction error. To achieve these capabilities, we develop real-time algorithms based on the off-line least squares and orthogonal least squares methods.

3.2 Objective

The control and identification of a nonlinear system can often be viewed as a nonlinear function approximation problem. If this nonlinear function is continuous and differentiable over a compact subset of its domain, then according to universal approximation theorems (Park & Sandberg 1991, Poggio & Girosi 1990a, 1990b), there exists a linear combination of radial basis functions that can uniformly approximate this nonlinear function to any degree of accuracy, provided that enough basis functions are available. The RBF network can be mathematically represented as:

$$y(\mathbf{p}) = \sum_{i=0}^{q} x_i a_i(\mathbf{p}, \mathbf{c}_i) \qquad (3 - 1)$$

where a_i is the RBF node, $\mathbf{c}_i$ is the center, $\mathbf{p} = [p_1, p_2, \ldots, p_r]$ is a set of input signals, and x_i is the output weight.

However, because radial basis functions are local receptive fields, the number of basis functions employed can be very large. The resulting expansions will thus be capable of only approximating the nonlinear function $y(\mathbf{p})$ on a particular subset of the input space. Worst of all, the number of RBF nodes exponentially increases with the number of inputs. This phenomenon is referred to as the curse of dimensionality. (Haykin 1994)

The curse of dimensionality problem becomes particularly acute in stable adaptive control using the RBF network (Sanner 1993, Sanner & Soltine 1992, 1995, Tzirkel & Fallside 1992). In this technique, the RBF network (used inside the relevant region) is combined with a sliding mode controller (used outside of the relevant region) to achieve globally stable adaptive control. To guarantee the stability of the controller, the RBF network has to be constructed in such a way that it yields a minimum network reconstruction error. According to Sanner (1993), this criteria is guaranteed by constructing the RBF centers on an equally spaced mesh grid covering a relevant region. To achieve small reconstruction error, hundreds of RBF nodes may be needed to cover a relevant range in each dimension, and therefore several thousand RBF nodes may be needed to cover the relevant region of the input space. Consequently, this limits the control technique to low dimensional systems.

To alleviate this problem, several stable adaptive control techniques (Fabri & Kadirkamanathan 1996, Liu & Kadirkamanathan 1996) were introduced by employing a

growing RBF network combined with the sliding mode controller. These techniques also assume that the centers are equally spaced and cover a relevant region. However, it only activates the RBF nodes when the nonlinear system inputs are near their RBF centers. Hence, the RBF network grows larger and larger as the nonlinear system visits various regions of the state space. Accordingly, this technique may save many RBF nodes depending on whether or not the nonlinear system visits the corresponding centers. However, in the worst scenario, no saving of RBF nodes will occur when the nonlinear system visits all of the input space.

Again, these techniques are clearly unsatisfactory, as they do not reduce the growth of the RBF nodes, i.e. no saving of RBF nodes will occur when the nonlinear system visits all of the state space. Therefore, they suffer from curse of dimensionality as well.

For practical purposes, it is desired to construct small RBF networks on-line. Small RBF networks often provide better performance, because they generalize better. A search of existing literature has revealed very few ad-hoc techniques (Karayiannis & Mi, 1997) that can select small RBF networks, while operating in real-time and simultaneously guaranteeing a minimum reconstruction error. On the other hand, there exist many off-line (batch) techniques that can select small RBF networks and guarantee a minimum reconstruction error. Off-line techniques usually collect a finite set of input-output data and perform complex calculations to determine the number of nodes required. In general, these techniques can be classified into three categories:

1. network pruning techniques - such as the optimal brain damage method (Cun *et al.* 1990), and the optimal brain surgeon method (Hassibi & Stork 1992).

2. network growing techniques - such as the orthogonal least squares method (Chen *et al.* 1991), and the cascade correlation learning architectures (Fahlman & Lebiere 1990).

3. network parameter determination - such as the Bayesian regularization method (MacKay 1992, 1994).

The orthogonal least squares (OLS) method is a simple and efficient method. This method guarantees a level of network reconstruction error and produces a small RBF network. This procedure first assumes that each input data is a potential RBF center. A set of RBF hidden layer outputs is obtained by feeding the input data into the hidden layer using all of the potential RBF centers. Then, one by one, the potential RBF center that produces the largest reduction in network error is added to the network. This selection process continues until an adequate network reconstruction error has been reached. We will demonstrate this concept through the following example.

3.3 Illustrative Example

Assume that we have the following sine wave function

$$y_f(p) = \sin(p) \quad 0 \le p \le 2\pi \quad \text{sampling interval} = \pi/8 . \tag{3 - 2}$$

A set of potential RBF hidden layer outputs are created using the potential RBF centers selected from the input data and $\sigma = 1$. As shown in Figure 3 - 1, 17 RBF nodes (right figure) are available to approximate the sine wave function (+ mark in left figure). These

nodes are arranged from left to right as $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{17}\}$, with centers at $\{0, \pi/8, \ldots, 2\pi\}$.

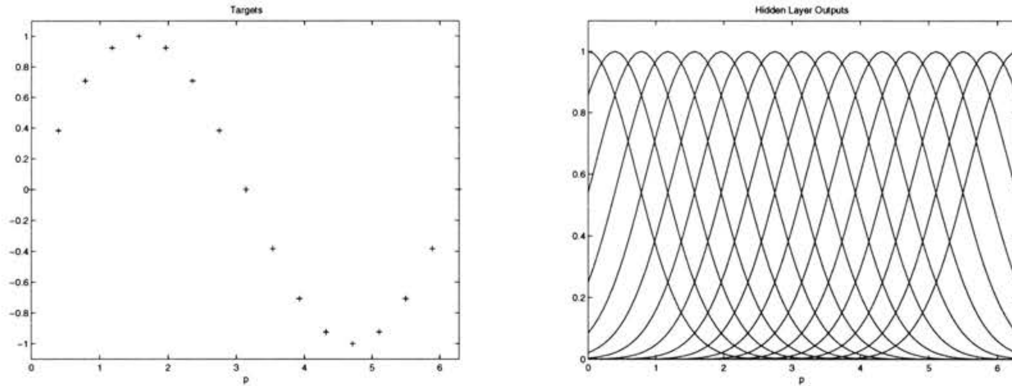


Figure 3 - 1 Desired Target and the RBF Hidden Layer Outputs

Intuitively, if we were to select the RBF nodes visually, we would select two RBF nodes, one at the peak and the other at the valley (corresponding to RBF $\mathbf{a}_5$ and $\mathbf{a}_{13}$ nodes) of the sine wave. The outputs of these two nodes would seem to best match the curvature of the sine wave.

The OLS method finds the optimal nodes one by one in several steps. It begins with no nodes. At each step, a node that produces the largest reduction in the network error is selected from the RBF hidden layer outputs and is added into the network. These steps are repeated until a target network reconstruction error is reached. This procedure is best explained by a graphical example as shown in Figure 3 - 2.

Figure 3 - 2(a) shows the first step of this algorithm. It adds node $\mathbf{a}_5$ into the network, since this node provides the largest reduction in network error. However, one node is not enough to capture the whole sine wave. Hence, in a second step, (Figure 3 - 2(b))

another node $\mathbf{a}_{13}$ is selected to aid the reconstruction of the sine wave. Together, both nodes approximate the complete sine wave, as shown in Figure 3 - 2(c). (By adding more nodes, we can further improve the approximation.)

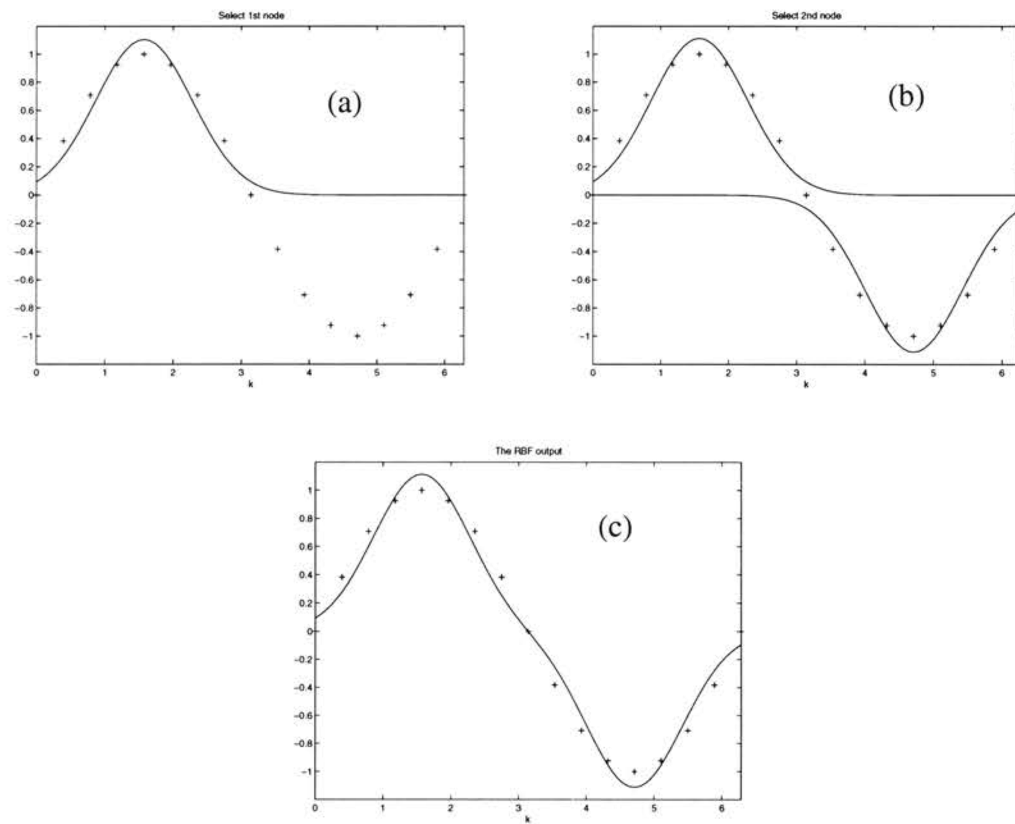


Figure 3 - 2 Orthogonal Least Squares Center Selection

3.4 The Research Outline

The OLS learning algorithm is a simple and efficient algorithm for selecting a small size RBF network. However, one drawback with this method is that the training is done in batch mode only. (Batch means that the entire training set must be available. A recursive algorithm can update the parameter estimates as each new data point is received.)

This research has found a way to produce on-line LS and OLS learning methods. This goal is achieved by considering three major issues: time-update, order-update, and subset selection. The time-update is the process of updating the RBF weights when a new time point data is received. Two time-update algorithms, the recursive least squares (RLS) algorithm and the QR recursive least squares (QR-RLS) algorithm, are discussed in Chapter 5 to address this issue.

The order-update is the recalculation of the optimal RBF weights when a new RBF node is added (order-increase-update) or deleted (order-decrease-update). Chapter 6 addresses this issue by deriving the recursive order-increase-update procedures for least squares and orthogonal least squares methods to recalculate the optimal weights. Meanwhile, the subset selection occurs before the order-update. It is a process of selecting the optimal node for the RBF network and deciding whether an order-update is necessary at a particular time point. This issue is also discussed in Chapter 6.

If we take all three processes together, we arrive at a time- and order- update framework, which can be used to select useful RBF nodes sub-optimally and recursively. Because the framework can be applied to the RLS method and the QR-RLS method, two

algorithms are developed. We call these algorithms Recursive Least Squares with Automatic Weight Selection (RLS-AWS) and QR Recursive Least Square with Automatic Weight Selection (QR-RLS-AWS).

In Chapter 8 and 9, we devote our efforts to improve the RLS-AWS and the QR-RLS-AWS algorithms. These improvements include alleviating the storage requirement, improving the algorithm's subset selection solution by developing the recursive Efroymson method, and reducing the computation efforts. Subsequently, we make these algorithms practical for real-time usage.

With these two algorithms, we hope to alleviate the problem of the curse of dimensionality by producing moderate RBF network sizes.

Chapter 4

The Least Squares Method

4.1 Introduction	48
4.2 Linear Model	48
4.3 Solving the Linear Model	50
4.3.1 The Linear Least Squares Problem	50
4.3.2 Characterization of Least Squares Solution	51
4.4 Orthogonal Projection	52
4.4.1 Subspace Projection	53
4.4.2 Matrix Projection and Geometrical Interpretation	54
4.5 Orthogonal Transformation	56
4.5.1 Motivation	56
4.5.2 Orthogonal Least Squares Method	57
4.6 Givens Rotations	58
4.6.1 Givens QR Methods	61
4.7 Summary	62

In chapter 2, we pointed out that the RBF network with fixed centers and fixed standard deviation could be viewed as a linear model. In this chapter, we will discuss the necessary tools for solving this "linear" RBF network: the least squares method and the orthogonal least squares method.

4.1 Introduction

This chapter is organized as follows. In section 4.1, we introduce the linear model. In section 4.2, we formulate the linear model solution as a linear least squares problem. Then, we characterize the least squares solution as a solution to the normal equations. The Orthogonality Theorem and Uniqueness Theorem are summarized as the backbone proofs for this least squares method. In section 4.3, we view the least squares problem geometrically. The concept of matrix projection through the subspace projection is introduced and a geometrical interpretation of the matrix projection is discussed. In section 4.4, we highlight the numerical problems, which occur when we use the normal equations to solve the least squares problem. A better least squares method based on an orthogonal transformation is introduced. We discuss several orthogonalization tools, but detailed attention is given to the Givens rotation. The Givens rotation will serve as the central process for the Givens QR algorithm; hence, it is discussed in detail in section 4.5. Readers are encouraged to pay extra attention to the Givens rotation operations as they are central to the QR recursive least squares algorithm in chapter 6, 7, 8 and 9.

4.2 Linear Model

Linear models exist in all scientific disciplines. The linear model might seem to be highly restricted, but many industrial processes can be described very accurately by these types of models. In fact, it is one of the most widely used models in industrial applications,

such as in control, signal processing, etc. In linear models, one assumes that the desired vector $\mathbf{d} \in \Re^m$ is related to the unknown parameter vector $\mathbf{x} \in \Re^n$ by a linear relation

$$\mathbf{Ax} = \mathbf{d}, \tag{4 - 1}$$

where $\mathbf{A} \in \Re^{m \times n}$ is a known data matrix. This equation has an exact solution when we can match the desired vector $\mathbf{d}$ exactly with a linear combination of the columns of $\mathbf{A}$. For example, if $m = 3,\ n = 1$,

$$\mathbf{A} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } \mathbf{d} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}, \tag{4 - 2}$$

then $\mathbf{x} = 2$ produces an exact solution. However, in many instances, $\mathbf{d}$ cannot be expressed in the form of $\mathbf{Ax}$. For example, if $m = 3,\ n = 1$,

$$\mathbf{A} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } \mathbf{d} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \tag{4 - 3}$$

no value of $\mathbf{x}$ can ever produce $\mathbf{d}$ exactly. This leads us to an extension of the linear model,

$$\mathbf{Ax} + \mathbf{e} = \mathbf{d} \tag{4 - 4}$$

where $\mathbf{e}$ is the error vector. The solution of Eq. (4 - 4) is to find a parameter $\mathbf{x}$ such that $\mathbf{Ax}$ is as close as possible to $\mathbf{d}$; in other words, find the "best" fit (albeit not perfect) between $\mathbf{Ax}$ and $\mathbf{d}$.

4.3 Solving the Linear Model

4.3.1 The Linear Least Squares Problem

How do we measure the distance between $\mathbf{Ax}$ and $\mathbf{d}$? There are many possible ways. One choice is motivated by statistical considerations and leads to a simple solution. It is the Euclidean vector norm (the two-norm). This leads to the minimization problem

$$\min_{\mathbf{x}} \|\mathbf{d} - \mathbf{Ax}\|_2, \quad \mathbf{A} \in \Re^{m \times n}, \quad \mathbf{d} \in \Re^m, \tag{4 - 5}$$

where $\| \bullet \|_2$ denotes the Euclidean vector norm. If $\mathbf{d} - \mathbf{Ax} = \mathbf{e}$, then $\|\mathbf{d} - \mathbf{Ax}\|_2 = \mathbf{e}^T\mathbf{e}$.

Of course, there are other norms we can use, such as the Holder vector p-norms $\| \bullet \|_p$, which are defined by

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad 1 \le p \le \infty. \tag{4 - 6}$$

For our research, however, we will only focus on the two-norm. In fact, the two-norm minimization problem is the linear least squares problem. In the following section, we will characterize the set of all solutions to the least squares problem.

4.3.2 Characterization of Least Squares Solution

The solution of the least squares problem has been widely discussed in many books. We will not attempt to cover this material in great detail, but will give a brief review on some of the characteristics of the least squares solution. For more detail, see (Bjorck 1996, Golub & VanLoan 1996, and Haykin 1996).

There are two unique least squares properties; the orthogonality condition and the uniqueness condition. We summarize these two properties in the following two theorems.

Theorem 4 - 1 Orthogonality Condition

Let us denote the set of all solutions to the least squares problem Eq. (4 - 5) by

$$S = \{\mathbf{x} \in \mathbf{R}^n \mid \|\mathbf{d} - \mathbf{A}\mathbf{x}\|_2 = \min\}. \tag{4 - 7}$$

It can be shown that $\mathbf{x} \in S$ if and only if the following orthogonality condition holds:

$$\mathbf{A}^T(\mathbf{d} - \mathbf{A}\mathbf{x}) = \mathbf{0}. \tag{4 - 8}$$

Proof. (See Bjorck 1996 pp.5).

Theorem 4 - 1 is known as the principle of orthogonality (Haykin 1996). If we expand Eq. (4 - 8), we obtain

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{d}, \tag{4 - 9}$$

which is called the normal equation. This implies that the solution of Eq. (4 - 5) must satisfy the normal equation. Assuming for now that the inverse matrix $(\mathbf{A}^T\mathbf{A})^{-1}$ exists, we may solve the linear least squares problem as

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{d}. \tag{4 - 10}$$

It is important to know when this solution is unique. This is covered by the following Uniqueness Theorem.

Theorem 4 - 2 Uniqueness

If $\mathbf{A} \in \Re^{m \times n}$ has full rank n, then there exists a unique least squares solution $\mathbf{x}$ and a residual $\mathbf{e} = \mathbf{d} - \mathbf{A}\mathbf{x}$, which is given by

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{d}, \quad \text{and} \quad \mathbf{e} = \mathbf{d} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{d}. \qquad (4 - 11)$$

Proof. (See Bjorck 1996 pp.7).

Theorem 4 - 2 shows that we may expect a unique solution to the least squares problem only when the data matrix $\mathbf{A}$ has linearly independent columns. In this case, the inverse matrix $(\mathbf{A}^T\mathbf{A})^{-1}$ is non-singular (therefore invertible) and the least squares solution is unique.

On the other hand, if $rank(\mathbf{A}) < n$, then an infinite number of solutions can be found for minimizing the sum of squared errors. We defer discussion of this issue to the later part of the chapter. In the meantime, we assume that data matrix $\mathbf{A}$ is of full column rank, so that the least squares estimate $\mathbf{x}$ has the unique value defined by Eq. (4 - 10).

4.4 Orthogonal Projection

Notice in Eq. (4 - 11) that the residual e contains a term $\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$. This term is called the projector. To understand the concept of projection and its relationship with the least squares solution, we must first understand the concept of projection onto a subspace.

4.4.1 Subspace Projection

Let S be a subspace of $\Re^m$ and $\mathbf{s} \in S$ be an element in S. Then, the orthogonal complement of S, denoted by $S_\perp$, is defined as the set of all m-dimensional vectors that are orthogonal to vectors in S. i.e. if $\mathbf{a}$ is an element in $S_\perp$, then

$$S_\perp = \{\mathbf{a} \in \Re^m | \mathbf{a}^T\mathbf{s} = 0 \quad \text{for all} \quad \mathbf{s} \in \Re^m\}. \tag{4 - 12}$$

Note that $S_\perp$ is also a subspace of $\Re^m$. Together, S and $S_\perp$ comprise all of $\Re^m$, and have no vectors in common except for the zero vector:

$$S \cup S_\perp = \Re^m \quad \text{and} \quad S \cap S_\perp = \{\mathbf{0}\}. \tag{4 - 13}$$

An important relationship between the S and $S_\perp$ is that any m-vector $\mathbf{d}$ can be represented as

$$\mathbf{d} = \mathbf{d}_S + \mathbf{d}_{S_\perp}, \text{ where } \mathbf{d}_S \in S \text{ and } \mathbf{d}_{S_\perp} \in S_\perp. \tag{4 - 14}$$

Let S be a subspace of $\Re^m$, then $\bar{\mathbf{P}}_S$, a unique $m \times m$ matrix, is an orthogonal projector onto the subspace S if it satisfies the following properties (Bjorck 1996):

1. Every vector in the subspace S can be written as a linear combination of the columns of $\bar{\mathbf{P}}_S$, i.e., the vector $\mathbf{d}_S$ lies in S if and only if $\mathbf{d}_S = \bar{\mathbf{P}}_S\mathbf{d}$ for some m-vector $\mathbf{d}$.
2. $\bar{\mathbf{P}}_S^T = \bar{\mathbf{P}}_S$ (Symmetric Property).
3. $\bar{\mathbf{P}}_S^2 = \bar{\mathbf{P}}_S$ (Idempotent Property).

It is important to note that these properties also apply to the orthogonal complement projector $\mathbf{I} - \bar{\mathbf{P}}_S$. If we apply the projector $\bar{\mathbf{P}}_S$ to any m-vector $\mathbf{d}$, it will produce $\mathbf{d}_S$ - the portion of $\mathbf{d}$ that lies in S.

$$\bar{\mathbf{P}}_S \mathbf{d} = \mathbf{d}_S \tag{4 - 15}$$

Similarly, if we apply the orthogonal complement projector $\mathbf{I} - \bar{\mathbf{P}}_S$ to any m-vector $\mathbf{d}$, it will produces $\mathbf{d}_{S_\perp}$ - the portion of $\mathbf{d}$ that lies in $S_\perp$.

$$(\mathbf{I} - \bar{\mathbf{P}}_S)\mathbf{d} = \mathbf{d}_{S_\perp} \tag{4 - 16}$$

4.4.2 Matrix Projection and Geometrical Interpretation

The notion of subspace projection is closely tied to the matrix projection. As in subspace projection, we can decompose a m-vector $\mathbf{d}$ into a sum of two quantities. In matrix projection, these two quantities fall into the range space of $\mathbf{A}$ and the null space of $\mathbf{A}^T$:

$$\mathbf{d} = \mathbf{d}_R + \mathbf{d}_N, \tag{4 - 17}$$

where $\mathbf{d}_R \in range(\mathbf{A})$ and $\mathbf{d}_N \in null(\mathbf{A}^T)$. The matrix projector, $\bar{\mathbf{P}}_A$, is a projector onto the range of $\mathbf{A}$ with properties similar to the subspace projector $\bar{\mathbf{P}}_S$. (Bjorck 1996)

1. $\bar{\mathbf{P}}_A \mathbf{d} = \mathbf{d}_R$ and $(\mathbf{I} - \bar{\mathbf{P}}_A)\mathbf{d} = \mathbf{d}_N$.

2. $\bar{\mathbf{P}}_A^T = \bar{\mathbf{P}}_A$ (Symmetric Property).

3. $\bar{\mathbf{P}}_A^2 = \bar{\mathbf{P}}_A$ (Idempotent Property).

Using Eq. (4 - 17) and Property (1), we can decompose a m-vector $\mathbf{d}$ into

$$\mathbf{d} = \bar{\mathbf{P}}_A\mathbf{d} + (\mathbf{I} - \bar{\mathbf{P}}_A)\mathbf{d}. \qquad (4 - 18)$$

$\bar{\mathbf{P}}_A$ projects the $\mathbf{d}$ vector onto the column space of the matrix $\mathbf{A} \in \Re^{m \times n}$ where

$$\bar{\mathbf{P}}_A = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T, \qquad (4 - 19)$$

and

$$\mathbf{I} - \mathbf{A}(\mathbf{A}^T\mathbf{A})\mathbf{A}^T = \mathbf{I} - \bar{\mathbf{P}}_A \qquad (4 - 20)$$

is the orthogonal complement projector (or simply orthogonal projector). When applying the matrix projector $\bar{\mathbf{P}}_A$ to the desired data vector $\mathbf{d}$, we get an estimated data vector $\hat{\mathbf{d}}$. Likewise, if we apply the orthogonal projector, $\mathbf{I} - \bar{\mathbf{P}}_A$, to the desired vector $\mathbf{d}$, we obtain the error vector $\mathbf{e} = \mathbf{d} - \hat{\mathbf{d}}$. This projection operator can be illustrated by the following diagram.

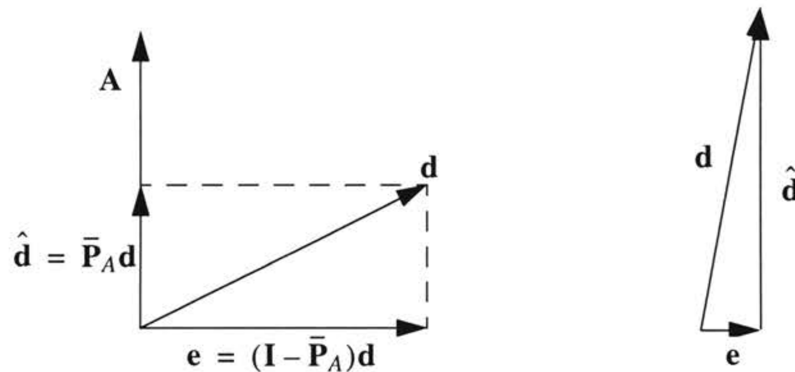


Figure 4 - 1 Geometrical Interpretation of the Orthogonal Projection

As shown, the objective is to minimize the length of the error vector $\mathbf{e}$, so that the desired response, $\mathbf{d}$, is as close as possible to the estimated data vector $\hat{\mathbf{d}}$.

4.5 Orthogonal Transformation

4.5.1 Motivation

Although the normal equation is the fastest way to solve the least squares problem, it suffers from a lack of accuracy. This problem is illustrated by the following example. Consider

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 10^{-8} \end{bmatrix},$$

then the associated sum of squared matrix is

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 2 + 10^{-16} \end{bmatrix}.$$

Now, if we have an infinite precision computer, we will obtain an exact $\mathbf{A}^T\mathbf{A}$ solution. However, if we use double precision arithmetic, $2 + 10^{-16}$ will be rounded to 2, and we will obtain $\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$, which is not invertible. Since $\mathbf{A}$ is the original data matrix, this ill-conditioning problem cannot be avoided by choosing another

parameterization. The accuracy of the computed normal equation solution may then depend on the square of the condition number of $\mathbf{A}$.

4.5.2 Orthogonal Least Squares Method

Due to this numerical difficulty, modern least squares methods have been developed based on orthogonal transformations. These are called the orthogonal least squares methods. The main idea of the orthogonal transformation is to work directly with the original data matrix $\mathbf{A}$ by decomposing $\mathbf{A} \in \Re^{m \times n}$ into an upper triangular matrix $\mathbf{R} \in \Re^{m \times n}$ using an orthogonal matrix $\mathbf{Q}^T \in \Re^{m \times m}$.

$$\mathbf{Q}^T\mathbf{A} = \mathbf{R} \tag{4 - 21}$$

Using this definition, the least squares minimization problem can be rewritten as

$$\min_{x}\|\mathbf{d} - \mathbf{A}\mathbf{x}\|_2 = \min_{x}\|\mathbf{Q}^T(\mathbf{d} - \mathbf{A}\mathbf{x})\|_2 . \tag{4 - 22}$$

Since $\mathbf{Q}^T$ is an orthogonal matrix, its application to the error residual preserves the Euclidean length (does not alter the two-norm) and cannot exacerbate the condition of $\mathbf{A}$. Because we are no longer solving equations, we avoid the numerical inaccuracy associated with forming the $\mathbf{A}^T\mathbf{A}$ matrix.

To solve the least squares problem using the orthogonal transformations, we need to find the orthogonal matrix. We find the orthogonal matrix by applying a sequence of special transformations to $\mathbf{A}$ or by Gram-Schmidt orthogonalization methods. Because Gram-Schmidt orthogonalization methods are not very useful in recursive least squares, we

will not discuss them here. The orthogonal transformations used in finding the orthogonal matrix can be the Householder reflections or the Givens rotations. Both transformations can be easily applied to recursive least squares. The Householder reflections introduce zeros on a grand scale (they annihilate all but the first component of a vector) while Givens rotations introduce zeros element by element (they annihilate one element in a vector one at a time). We will only discuss the Givens rotations, since it will be used in Chapter 5 when we introduce square root filtering.

4.6 Givens Rotations

The Givens rotations (Givens 1958) are also known as plane rotations or Jacobi rotations (Jacobi 1846). It is referred to as Jacobi rotations in honor of Jacobi 1846, who proposed a method for reducing a symmetric matrix to diagonal form. It is referred to as Givens rotations in honor of Givens 1958, who proposed a method for reducing a general matrix to triangular form. Also, it is referred to "plane rotation" because multiplication by this matrix will give a plane rotation.

Let $\mathbf{G}(i, k)$ denotes a Givens rotation in the (i, k) plane, where $k > i$. The $\mathbf{G}(i, k)$ matrix is the same as the $M \times M$ identity matrix, except for the four strategic elements located on the rows *i, k* and columns *i, k*. At these locations, $c = \cos(\theta)$ and $s = \sin(\theta)$ as in the following:

$$\mathbf{G}(i,k) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix} \tag{4 - 23}$$

$$\begin{matrix} \quad\quad & i & \quad & k \end{matrix}$$

The above Givens rotation matrix is clearly orthogonal, as $\mathbf{G}^T\mathbf{G} = \mathbf{I}$. To illustrate the nature of this Givens rotation, consider

$$\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_i & \cdots & x_k & \cdots & x_M \end{bmatrix}^T, \text{ and} \tag{4 - 24}$$

$$\mathbf{y} = \begin{bmatrix} y_1 & \cdots & y_i & \cdots & y_k & \cdots & y_M \end{bmatrix}^T. \tag{4 - 25}$$

Premultiplying the vector $\mathbf{x}$ by $\mathbf{G}(i,k)^T$ yields

$$\mathbf{y} = \mathbf{G}(i,k)^T\mathbf{x}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_k \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}^T \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_k \\ \vdots \\ x_M \end{bmatrix}. \tag{4 - 26}$$

From the above, we can derive the following set of equations

$$
\begin{aligned}
y_i &= cx_i - sx_k \\
y_k &= sx_i + cx_k \\
y_j &= x_j \quad j \neq i, \text{ and } j \neq k
\end{aligned}
\qquad (4 - 27)
$$

From these equations, it is clear that we can force y_k to zero by setting

$$
c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \text{ and } s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}. \qquad (4 - 28)
$$

Note that it is not necessary to compute θ. Thus, the Givens rotation is the transformation of choice when we need to zero a specified entry in a vector. In practice, we do have to guard against overflow, and the following version of the Givens rotation (Golub & Van Loan 1996) is often used.

Givens Rotation Algorithm

Given scalars a and b, this function computes $c = \cos(\theta)$ and $s = \sin(\theta)$ so that

$$
\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \qquad (4 - 29)
$$

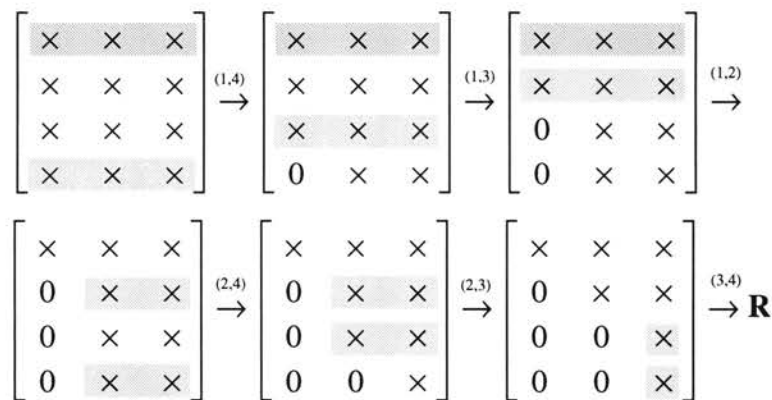
function $[c, s] =$ **givens**(a, b)
  if $b = 0$
    $c = 1; s = 0$
  else
    if $|b| > |a|$
      $\tau = -a/b; s = 1/\sqrt{1+\tau^2}; c = s\tau$
    else
      $\tau = -b/a; c = 1/\sqrt{1+\tau^2}; s = c\tau$
    end
  end

To apply the Givens rotation to a full matrix $\mathbf{A}$, it is critical to exploit the simple structure of a Givens rotation matrix when it involves a matrix multiplication. Suppose $\mathbf{A} \in \Re^{m \times n}$, then the update $\mathbf{G}(i, k)^T\mathbf{A}$ effects just two rows of $\mathbf{A}$,

$$\mathbf{A}([i, k], :) = \mathbf{G}(i, k)^T\mathbf{A}([i, k], :) . \tag{4 - 30}$$

4.6.1 Givens QR Methods

With the Givens rotation capable of zeroing a specific entry in a matrix, we can apply a sequence of Givens rotations to reduce $\mathbf{A}$ to an upper triangular matrix. The following 4 by 3 case illustrates the general idea:



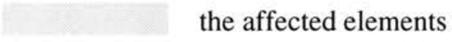

the affected elements

Figure 4 - 2 Givens QR Annihilation on a 4x3 Matrix

The annihilation begins at the top left matrix and ends at the bottom right matrix. The highlighted elements in the matrix are the elements that are affected by each annihilation. On each sequence, the annihilation shows the (i,k) element that has been zeroed. If $\mathbf{G}_j(i, k)$ denotes the j-th Givens rotation in the reduction, then $\mathbf{Q}^T\mathbf{A} = \mathbf{R}$ is

upper triangular, where $\mathbf{Q}^T$ is represented by the sequence of Givens rotations $\mathbf{G}_j(i, k)$ applied to the $\mathbf{A}$ matrix:

$$\mathbf{G}_6(3, 4)^T\mathbf{G}_5(2, 3)^T\mathbf{G}_4(3, 4)^T\mathbf{G}_3(1, 2)^T\mathbf{G}_2(2, 3)^T\mathbf{G}_1(3, 4)^T\mathbf{A} = \mathbf{R} \qquad (4 - 31)$$

Givens QR Algorithm

Given $\mathbf{A} \in \Re^{m \times n}$ with $m \geq n$, the following Givens QR algorithm overwrites $\mathbf{A}$ with $\mathbf{Q}^T\mathbf{A} = \mathbf{R}$, where $\mathbf{R}$ is upper triangular and $\mathbf{Q}$ is orthogonal.

for $j = 1:n$
    for $i = m:-1:j+1$
        $[c, s] = \mathbf{givens}(\mathbf{A}(j, j), \mathbf{A}(i, j))$

$$\mathbf{A}([j, i], j:n) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \mathbf{A}([j, i], j:n) \qquad (4 - 32)$$

    end
end

Accordingly, this algorithm requires $3n^2(m - n/3)$ flops (Golub & Van Loan 1996).

4.7 Summary

This chapter has covered the fundamental least squares techniques. It also discussed the important concept of subspace/matrix projections. Several important tools such as the Givens rotations and QR Givens algorithm were discussed. These tools will be used to facilitate algorithm development in later chapters. With this introduction to the Givens rotation and the Givens QR algorithm, we are ready to explore the time update algorithms in the next chapter.

Chapter 5

Time-Update Algorithms

5.1 Introduction	**64**
5.1 Matrix Inversion Lemma	**66**
5.2 Recursive Least Squares Algorithm	**67**
5.2.1 Time-Update for the Parameter	**69**
5.2.2 Time-Update for the Sum of Squares Errors	**71**
5.2.3 Implementation Considerations	**74**
5.3 QR Recursive Least Squares Method	**76**
5.3.1 Introduction	**76**
5.3.2 Preliminary Setup for QR-RLS Algorithm	**77**
5.3.3 Forming the QR-RLS Algorithm	**82**
5.3.4 Orthogonal Matrix Operation	**85**
5.3.5 Implementation Considerations	**87**
5.4 Results Summary	**88**

Our main mission in this chapter is to develop the basic theory behind the time update algorithms, specifically the Recursive Least Squares (RLS) algorithm and the QR Recursive Least Squares (QR-RLS) algorithm. These algorithms serve as important tools for the Recursive Least Squares with Automatic Weight Selection (RLS-AWS) algorithm and the QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) algorithm, which will be developed in Chapter 7 - 9.

5.1 Introduction

We begin the development of the RLS algorithm by reviewing some basic relations that pertain to the method of least squares. By exploiting a relation in matrix algebra known as the matrix inversion lemma, we develop the RLS algorithm. Later, we point out that the RLS algorithm suffers from numerical instability, and we introduce a more stable method called the QR-RLS algorithm. We derive the QR-RLS algorithm based on the RLS algorithm and the matrix factorization lemma. Then, we also discuss how the QR-RLS algorithm operates using the orthogonal matrix. Each of these algorithms is discussed in detail and a summary is given at the end of the chapter.

In the following, we extend the batch least squares method of Chapter 4 to the recursive least squares algorithm. To ease the derivation, we introduce the time notation k to denote the difference between the past data and the current data. Specifically, the time index $k-1$ denotes the last time step and the time index k denotes the current time step.

Let us assume that, in the last time step, we found the least squares solution to the linear model given by Eq. (5 - 1). The least squares solution for this linear model is given in Eq. (5 - 2),

$$\mathbf{A}(k-1)\mathbf{x}(k-1) = \mathbf{d}(k-1), \tag{5 - 1}$$

$$\mathbf{x}(k-1) = (\mathbf{A}^T(k-1)\mathbf{A}(k-1))^{-1}\mathbf{A}^T(k-1)\mathbf{d}(k-1). \tag{5 - 2}$$

Note that $\mathbf{A}(k-1)$ and $\mathbf{d}(k-1)$ are the data matrix and desired response vector given by:

$$\mathbf{A}(k-1) = \begin{bmatrix} \mathbf{a}^T(0) \\ \mathbf{a}^T(1) \\ \vdots \\ \mathbf{a}^T(k-1) \end{bmatrix}, \mathbf{d}(k-1) = \begin{bmatrix} d(0) \\ d(1) \\ \vdots \\ d(k-1) \end{bmatrix}. \tag{5 - 3}$$

Meanwhile, $\mathbf{x}(k-1)$ is a parameter vector at time index $k-1$.

Now, at the current time step k, a new data vector $\mathbf{a}^T(k)$ and a new desired response $d(k)$ become available, and we wish to add this new information into the linear model as in Eq. (5 - 1). This new information is incorporated by adding $\mathbf{a}^T(k)$ as a new row of the data matrix $\mathbf{A}(k-1)$ and adding $d(k)$ into the desired vector $\mathbf{d}(k-1)$:

$$\mathbf{A}(k) = \begin{bmatrix} \mathbf{A}(k-1) \\ \mathbf{a}^T(k) \end{bmatrix}, \text{ and } \mathbf{d}(k) = \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix}. \tag{5 - 4}$$

Together, they form a new linear model at the current time step k,

$$\mathbf{A}(k)\mathbf{x}(k) = \mathbf{d}(k). \tag{5 - 5}$$

Naturally, we could compute the whole least squares solution again, but this would be time consuming. We usually avoid performing such an operation by finding a way to recursively update the least squares solution. This is especially important when the new data are arriving sequentially and the least squares solution must be computed in real time. The recursive solution can be obtained by using a basic result in matrix algebra known as the matrix inversion lemma.

5.2 Matrix Inversion Lemma

Before we derive the recursive least squares algorithm, we introduce the matrix inversion lemma. Let $\mathbf{A}$ and $(\mathbf{D}+\mathbf{C}^T\mathbf{A}^{-1}\mathbf{B})$ be two square and invertible matrices, then according to the matrix inversion lemma, we may express the inverse of the $\mathbf{A}+\mathbf{B}\mathbf{D}^{-1}\mathbf{C}^T$ matrix as follows:

$$(\mathbf{A}+\mathbf{B}\mathbf{D}^{-1}\mathbf{C}^T)^{-1} = \mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}+\mathbf{C}^T\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}^T\mathbf{A}^{-1} \qquad (5\text{ - }6)$$

In the special case where $\mathbf{B}$ and $\mathbf{C}$ are vectors (denoted by $\mathbf{b}$ and $\mathbf{c}$ respectively) and $\mathbf{D}$ is a scalar, d, Eq. (5 - 6) simplifies to

$$\left(\mathbf{A}+\frac{1}{d}\mathbf{b}\mathbf{c}^T\right)^{-1} = \mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{b}(d+\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}\mathbf{c}^T\mathbf{A}^{-1} \ . \qquad (5\text{ - }7)$$

Note that $(d+\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}$ is a scalar and the inversion is just a simple division. Frequently, Eq. (5 - 6) is called the Woodbury formula (Woodbury 1950) and Eq. (5 - 7) is called the Sherman-Morrison formula (Sherman & Morrison 1949). In engineering, these formulae are often referred to as the matrix inversion lemma. For a history, literature surveys, proofs and applications of the matrix inversion lemma, see Hager (1989).

5.3 Recursive Least Squares Algorithm

In this section, we apply the matrix inversion lemma to the batch least squares algorithm and obtain a recursive algorithm. We start by finding the least squares solution of this new linear model

$$\begin{bmatrix} \mathbf{A}(k-1) \\ \mathbf{a}^T(k) \end{bmatrix} \mathbf{x}(k) = \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix} \qquad (5 - 8)$$

where the new data, $\mathbf{a}^T(k)$, and the new desired response, $d(k)$, are added into the model. To solve for this new linear model, we form a new minimization problem

$$\min_x \|\mathbf{e}(k)\|_2 = \min_x \|\mathbf{d}(k) - \mathbf{A}(k)\mathbf{x}(k)\|_2 . \qquad (5 - 9)$$

Since the least squares solution of this new linear model also satisfies Eq. (5 - 2), we could easily expand the above equation into

$$\begin{aligned} \begin{bmatrix} \mathbf{A}^T(k-1) & \mathbf{a}(k) \end{bmatrix} \begin{bmatrix} \mathbf{A}(k-1) \\ \mathbf{a}^T(k) \end{bmatrix} \mathbf{x}(k) &= \begin{bmatrix} \mathbf{A}^T(k-1) & \mathbf{a}(k) \end{bmatrix} \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix} \\ (\mathbf{A}^T(k-1)\mathbf{A}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k))\mathbf{x}(k) &= \mathbf{A}^T(k-1)\mathbf{d}(k-1) + \mathbf{a}(k)d(k) \end{aligned} \quad . \qquad (5 - 10)$$

As mentioned previously, we can take the inverse of Eq. (5 - 10) and recalculate the whole solution, but it is impractical and time consuming. A better way of obtaining the solution is to apply the matrix inversion lemma, specifically the Sherman-Morrison Formula. When we apply the Sherman-Morrison Formula to the above equation, we obtain

$$\begin{aligned} &(\mathbf{A}^T(k-1)\mathbf{A}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k))^{-1} = \\ &\quad (\mathbf{A}^T(k-1)\mathbf{A}(k-1))^{-1} - (\mathbf{A}^T(k-1)\mathbf{A}(k-1))^{-1}\mathbf{a}(k) \times .... \\ &(1 + \mathbf{a}^T(k)(\mathbf{A}^T(k-1)\mathbf{A}(k-1))^{-1}\mathbf{a}(k))^{-1}\mathbf{a}^T(k)(\mathbf{A}^T(k-1)\mathbf{A}(k-1))^{-1} \end{aligned} \quad . \qquad (5 - 11)$$

Let

$$\bar{\mathbf{H}}(k) = (\mathbf{A}^T(k-1)\mathbf{A}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k))^{-1} \tag{5 - 12}$$

be the inverse correlation matrix at the current time step and

$$\bar{\mathbf{H}}(k-1) = (\mathbf{A}^T(k-1)\mathbf{A}(k-1))^{-1}, \tag{5 - 13}$$

be the inverse correlation matrix at the previous time step, then Eq. (5 - 11) becomes

$$\bar{\mathbf{H}}(k) = \bar{\mathbf{H}}(k-1) - \frac{\bar{\mathbf{H}}(k-1)\mathbf{a}(k)\mathbf{a}^T(k)\bar{\mathbf{H}}(k-1)}{1 + \mathbf{a}^T(k)\bar{\mathbf{H}}(k-1)\mathbf{a}(k)} . \tag{5 - 14}$$

We need to find the inverse of $1 + \mathbf{a}^T(k)\bar{\mathbf{H}}(k-1)\mathbf{a}(k)$, but this term is just a scalar, and the inverse is a simple division. Let us denote the scalar term $\kappa(k)$ as

$$\kappa(k) = 1 + \mathbf{a}^T(k)\bar{\mathbf{H}}(k-1)\mathbf{a}(k), \tag{5 - 15}$$

then because $\bar{\mathbf{H}}(k-1)$ is non-negative definite (See Ogata 1987 Appendix for definition of non-negative definite), $\mathbf{a}^T(k)\bar{\mathbf{H}}(k-1)\mathbf{a}(k) \geq 0$ and

$$\kappa(k) \geq 1 \text{ and } 0 \leq \kappa^{-1}(k) \leq 1 . \tag{5 - 16}$$

Let

$$\mathbf{k}(k) = \frac{\bar{\mathbf{H}}(k-1)\mathbf{a}(k)}{1 + \mathbf{a}^T(k)\bar{\mathbf{H}}(k-1)\mathbf{a}(k)} = \kappa^{-1}(k)\bar{\mathbf{H}}(k-1)\mathbf{a}(k) \tag{5 - 17}$$

be the gain vector (for reasons that will become apparent later in the section), then Eq. (5 - 14) can be written as

$$\bar{\mathbf{H}}(k) = \bar{\mathbf{H}}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\bar{\mathbf{H}}(k-1) . \tag{5 - 18}$$

In addition, if we multiply both sides of Eq. (5 - 17) by $1 + \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)$ and expand it out, the gain vector can be written in terms of the inverse correlation matrix at the current time step and the new data vector.

$$\begin{aligned} \mathbf{k}(k) + \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k) &= \overline{\mathbf{H}}(k-1)\mathbf{a}(k) \\ \mathbf{k}(k) &= (\underbrace{\overline{\mathbf{H}}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)}_{\overline{\mathbf{H}}(k)})\mathbf{a}(k) \\ \mathbf{k}(k) &= \overline{\mathbf{H}}(k)\mathbf{a}(k) \end{aligned} \qquad (5 - 19)$$

5.3.1 Time-Update for the Parameter

Now we are ready to develop a recursive equation for updating the least squares estimates for the parameter vector $\mathbf{x}(k)$. Using Eq. (5 - 18), we can express the parameter vector update in Eq. (5 - 10) as

$$\mathbf{x}(k) = [\overline{\mathbf{H}}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)](\mathbf{A}^T(k-1)\mathbf{d}(k-1) + \mathbf{a}(k)d(k)). \qquad (5 - 20)$$

Expanding Eq. (5 - 20), we get

$$\begin{aligned} \mathbf{x}(k) = \overline{\mathbf{H}}(k-1)\mathbf{A}^T(k-1)\mathbf{d}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{A}^T(k-1)\mathbf{d}(k-1) + \\ \overline{\mathbf{H}}(k-1)\mathbf{a}(k)d(k) - \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)d(k) \end{aligned} \qquad (5 - 21)$$

From Eq. (5 - 2), $\overline{\mathbf{H}}(k-1)\mathbf{A}^T(k-1)\mathbf{d}(k-1)$ is the least squares solution for the parameter vector $\mathbf{x}(k-1)$. Therefore, substituting and rearranging the remaining terms, we reduce Eq. (5 - 21) to

$$\mathbf{x}(k) = \mathbf{x}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\mathbf{x}(k-1) + [\overline{\mathbf{H}}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)]\mathbf{a}(k)d(k). \qquad (5 - 22)$$

Notice that we can substitute Eq. (5 - 18) into Eq. (5 - 22) and we can further reduce Eq. (5 - 22) to

$$\mathbf{x}(k) = \mathbf{x}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\mathbf{x}(k-1) + \overline{\mathbf{H}}(k)\mathbf{a}(k)d(k). \tag{5 - 23}$$

Finally, using the fact that $\overline{\mathbf{H}}(k)\mathbf{a}(k)$ equals the gain vector $\mathbf{k}(k)$, as in Eq. (5 - 19), we obtain the desired recursive equation for updating the parameter vector $\mathbf{x}(k)$:

$$\begin{aligned} \mathbf{x}(k) &= \mathbf{x}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\mathbf{x}(k-1) + \mathbf{k}(k)d(k) \\ &= \mathbf{x}(k-1) + \mathbf{k}(k)(d(k) - \mathbf{a}^T(k)\mathbf{x}(k-1)) \\ &= \mathbf{x}(k-1) + \mathbf{k}(k)\xi(k) \end{aligned} \tag{5 - 24}$$

where $\xi(k)$ is called the a priori estimation error defined by

$$\xi(k) = d(k) - \mathbf{a}^T(k)\mathbf{x}(k-1) . \tag{5 - 25}$$

The term $\xi(k)$ is called the a priori estimation error because it uses the past parameters $\mathbf{x}(k-1)$ to make up the inner product $\mathbf{a}^T(k)\mathbf{x}(k-1)$, which represents an estimate of the new desired response $d(k)$. Take note that the a priori estimation error is different from the a posteriori estimation error

$$e(k) = d(k) - \mathbf{a}^T(k)\mathbf{x}(k). \tag{5 - 26}$$

5.3.2 Time-Update for the Sum of Squares Errors

The relationship between the a priori estimation error $\xi(k)$ and a posteriori estimation error $e(k)$ becomes apparent when we formulate a recursive formula for the sum of squared errors $\mathbf{e}^T(k)\mathbf{e}(k)$. This recursive formula will be shown to be

$$\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{e}^T(k-1)\mathbf{e}(k-1) + \xi(k)e(k). \tag{5 - 27}$$

Proof. *We first note that*

$$\mathbf{e}(k) = \mathbf{d}(k) - \mathbf{A}(k)\mathbf{x}(k), \tag{5 - 28}$$

and the sum of squared errors is

$$\begin{aligned} \mathbf{e}^T(k)\mathbf{e}(k) &= \mathbf{d}^T(k)\mathbf{d}(k) - \mathbf{d}^T(k)\mathbf{A}(k)\mathbf{x}(k) \\ &- \mathbf{x}^T(k)\mathbf{A}^T(k)\mathbf{d}(k) + \mathbf{x}^T(k)\mathbf{A}^T(k)\mathbf{A}(k)\mathbf{x}(k) \end{aligned}. \tag{5 - 29}$$

Note that if we substitute $\mathbf{x}(k) = (\mathbf{A}^T(k)\mathbf{A}(k))^{-1}\mathbf{A}^T(k)\mathbf{d}(k)$ *into the last term of Eq. (5 - 29), then we get*

$$\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{d}^T(k)\mathbf{d}(k) - (\mathbf{A}^T(k)\mathbf{d}(k))^T\mathbf{x}(k). \tag{5 - 30}$$

Now, let us call the term $\mathbf{A}^T(k)\mathbf{d}(k)$ *the cross correlation vector*

$$\mathbf{v}(k) = \mathbf{A}^T(k)\mathbf{d}(k). \tag{5 - 31}$$

Then Eq. (5 - 30) can be expressed as

$$\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{d}^T(k)\mathbf{d}(k) - \mathbf{v}^T(k)\mathbf{x}(k). \tag{5 - 32}$$

Since $\mathbf{d}(k) = \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix}$ *and* $\mathbf{A}(k) = \begin{bmatrix} \mathbf{A}(k-1) \\ \mathbf{a}^T(k) \end{bmatrix}$, *Eq. (5 - 31) can be written as a recursive equation*

$$\mathbf{v}(k) = \mathbf{v}(k-1) + \mathbf{a}(k)d(k). \tag{5 - 33}$$

Also, the sum of squared of the desired responses becomes

$$\mathbf{d}^T(k)\mathbf{d}(k) = \mathbf{d}^T(k-1)\mathbf{d}(k-1) + d^T(k)d(k). \tag{5 - 34}$$

Combining Eq. (5 - 34), Eq. (5 - 33), and Eq. (5 - 29), we obtain

$$\begin{aligned}\mathbf{e}^T(k)\mathbf{e}(k) &= \mathbf{d}^T(k-1)\mathbf{d}(k-1) - \mathbf{v}^T(k-1)\mathbf{x}(k-1) \\ &+ d^T(k)(d(k) - \mathbf{a}^T(k)\mathbf{x}(k-1)) - \mathbf{v}^T(k)\mathbf{k}(k)\xi(k)\end{aligned}. \tag{5 - 35}$$

Now, if we rewrite Eq. (5 - 32) in terms of the previous time step $k-1$

$$\mathbf{e}^T(k-1)\mathbf{e}(k-1) = \mathbf{d}^T(k-1)\mathbf{d}(k-1) - \mathbf{v}^T(k-1)\mathbf{x}(k-1), \tag{5 - 36}$$

and substitute Eq. (5 - 25), then we have

$$\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{e}^T(k-1)\mathbf{e}(k-1) + d^T(k)\xi(k) - \mathbf{v}^T(k)\mathbf{k}(k)\xi(k). \tag{5 - 37}$$

Using Eq. (5 - 31) and Eq. (5 - 19), we can express the last term of Eq. (5 - 37) as

$$\begin{aligned}\mathbf{v}^T(k)\mathbf{k}(k) &= \mathbf{d}^T(k)\mathbf{A}(k)\overline{\mathbf{H}}(k)\mathbf{a}(k) \\ &= [\overline{\mathbf{H}}(k)\mathbf{A}^T(k)\mathbf{d}(k)]^T\mathbf{a}(k) \\ &= \mathbf{x}^T(k)\mathbf{a}(k)\end{aligned} \tag{5 - 38}$$

Finally, substituting Eq. (5 - 38) into Eq. (5 - 37), and noting that $e(k) = d(k) - \mathbf{a}^T(k)\mathbf{x}(k)$ *from Eq. (5 - 26), we get the final equation as in Eq. (5 - 27)*

$$\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{e}^T(k-1)\mathbf{e}(k-1) + e(k)\xi(k). \tag{5 - 39}$$

Take note that

$$
\begin{aligned}
e(k) &= d(k) - \mathbf{a}^T(k)\mathbf{x}(k-1) - \mathbf{a}^T(k)\mathbf{k}(k)\xi(k) \\
&= \xi(k)(1 - \mathbf{a}^T(k)\mathbf{k}(k)) \\
&= \xi(k)(1 - \kappa^{-1}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)) \\
&= \xi(k)\left(\frac{\kappa(k) - \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)}{\kappa(k)}\right) \\
&= \xi(k)\kappa^{-1}(k)
\end{aligned}
\tag{5 - 40}
$$

which means

$$
\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{e}^T(k-1)\mathbf{e}(k-1) + \xi^2(k)\kappa^{-1}(k). \tag{5 - 41}
$$

These recursive sum of squared errors formulations in Eq. (5 - 39) and Eq. (5 - 41) have two important implications. First, the product between the a priori estimation error $\xi(k)$ and the a posteriori estimation error $e(k)$ make up the new estimation error, which contributes to the new sum of squared errors. Second, due to the fact that $0 \le \kappa^{-1}(k) \le 1$ and $\xi^2(k)$ is a non-negative scalar, the sum of squared errors $\mathbf{e}^T(k)\mathbf{e}(k)$ accumulates error as time increases. Note that due to numerical round-off error, $\xi^2(k)\kappa^{-1}(k)$ can never be zero. This also implies that as $k \to \infty$, $\mathbf{e}^T(k)\mathbf{e}(k)$ will grow without bound and the algorithm becomes unstable! Numerous studies have shown that the RLS algorithm can become divergent due to the accumulation of numerical errors (Slock & Kailath 1991, Yang 1994, Ardalan & Alexander 1987).

To ensure stability of the RLS algorithm, the exponential windowing method has been widely incorporated into the RLS algorithm. However, if we can ensure that the sum of squared errors stay within a certain bound $\mathbf{e}^T(k)\mathbf{e}(k) \le \tau$, numerical error will not be an issue. We will address this in greater detail in the next chapter.

5.3.3 Implementation Considerations

The following sequence of equations constitutes the RLS algorithm: Eq. (5 - 17), Eq. (5 - 24), Eq. (5 - 25), and Eq. (5 - 18). To complete the RLS algorithm, we need to find a way to initialize it. We cannot simply set $\overline{\mathbf{H}}(0) = \mathbf{0}$, because that would imply we have an infinite correlation matrix. One simple way of initialization, according to Haykin (1996), is to modify the correlation matrix expression slightly. We can express the inverse correlation matrix as

$$\overline{\mathbf{H}}(k) = [\mathbf{A}^T(k)\mathbf{A}(k) + \delta\mathbf{I}]^{-1} \tag{5 - 42}$$

where $\mathbf{I}$ is an identity matrix and δ is a small positive constant. Using this expression, when $k = 0$, we have

$$\overline{\mathbf{H}}(0) = \delta^{-1}\mathbf{I}. \tag{5 - 43}$$

Now we can initialize the parameters at time step $k = 0$ as

$$\mathbf{x}(0) = \mathbf{0}. \tag{5 - 44}$$

This initialization procedure incorporating Eqs. (5 - 43) and (5 - 44) is referred to as a soft constrained initialization in statistical analysis (Hubing & Alexander 1990). The positive constant δ is the only parameter required for initialization. Through practical experiments, a typical value for δ should be small compared to $0.01\sigma_A^2$, where σ_A^2 is the variance of the input data $\mathbf{A}(k)$. Note that the exact value of δ is insignificant for large data samples. It is also interesting to note that using this initialization procedure, we are no longer computing the solution that minimizes the sum of squared errors as in Eq. (5 - 9).

Instead, we are computing the solution that minimizes the sum of squared errors plus the sum of squared parameters, pre-multiplied by the positive constant δ (Sayed & Kailath, 1994):

$$\min_{x} \quad \delta\|\mathbf{x}(k)\|_2 + \|\mathbf{e}(k)\|_2 . \tag{5 - 45}$$

One of the problems encountered in applying the RLS algorithm is numerical instability, which can arise due to its serious sensitivity to round-off errors. Due to this fact, in the next section, we will develop the RLS algorithm based on the QR decomposition. This algorithm is derived from the square-root Kalman filter, which does not suffer from numerical instability.

5.4 QR Recursive Least Squares Method

5.4.1 Introduction

Prior to the 1994 paper by Sayed and Kailath, the QR-RLS algorithm was derived by using the pre-windowed version of the data matrix, which was then triangularized by applying the QR decomposition (Golub & VanLoan 1989). The paper by Sayed and Kailath reveals for the first time how this QR decomposition of a pre-windowed data matrix can be deduced directly from their square-root Kalman filter counterparts. This technique resulted in three versions of the square-root Kalman filter algorithm for RLS estimation: the QR-RLS algorithm, the extended QR-RLS algorithm and the inverse QR-RLS algorithm. The motivation for using the QR decomposition in adaptive filtering is to exploit its good numerical properties. Since we will be using the QR-RLS algorithm to derive the Recursive OLS-AWS algorithm in Chapter 7, in this chapter we will discuss the QR-RLS algorithm in detail. Readers who are interested in the extended QR-RLS algorithm and the inverse QR-RLS algorithm can refer to Sayed and Kailath (1994) or Haykin (1996) for details.

5.4.2 Preliminary Setup for QR-RLS Algorithm

To derive the QR-RLS algorithm, we first need to set up the necessary recursive equations. For reasons that will become apparent later, we are looking for these particular recursive equations

$$\mathbf{H}(k) = \mathbf{H}(k-1) + \mathbf{F}(k-1) , \tag{5 - 46}$$

$$\mathbf{H}(k)\mathbf{x}(k) = \mathbf{H}(k-1)\mathbf{x}(k-1) + \mathbf{f}(k-1) , \tag{5 - 47}$$

$$\mathbf{x}(k)\mathbf{H}(k) = \mathbf{x}(k-1)\mathbf{H}(k-1) + \mathbf{f}^T(k-1) , \text{ and} \tag{5 - 48}$$

$$\mathbf{x}^T(k)\mathbf{H}(k)\mathbf{x}(k) = \mathbf{x}^T(k-1)\mathbf{H}(k-1)\mathbf{x}(k-1) + f(k-1) , \tag{5 - 49}$$

where $\mathbf{F}(k-1)$, $\mathbf{f}(k-1)$, $\mathbf{f}^T(k-1)$ and $f(k-1)$ are terms that can be found in the RLS algorithm.

To facilitate the development of these recursive equations, Eq. (5 - 10) is repeated in the following:

$$(\mathbf{A}^T(k-1)\mathbf{A}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k))\mathbf{x}(k) = \mathbf{A}^T(k-1)\mathbf{d}(k-1) + \mathbf{a}(k)d(k) . \tag{5 - 50}$$

Note that Eq. (5 - 50) is the least squares solution for the new linear model which can be written as

$$\mathbf{A}^T(k)\mathbf{A}(k)\mathbf{x}(k) = \mathbf{A}^T(k)\mathbf{d}(k) . \tag{5 - 51}$$

Since $\mathbf{H}(k) = \mathbf{A}^T(k)\mathbf{A}(k)$ and $\mathbf{H}(k-1) = \mathbf{A}^T(k-1)\mathbf{A}(k-1)$, we can obtain our first recursive equation by comparing equations (5 - 50) and (5 - 51)

$$\boxed{\mathbf{H}(k) = \mathbf{H}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k)} . \tag{5 - 52}$$

To obtain the second recursive equation, we first note that we can express Eq. (5 - 50) as

$$\mathbf{H}(k)\mathbf{x}(k) = \mathbf{A}^T(k-1)\mathbf{d}(k-1) + \mathbf{a}(k)d(k), \tag{5 - 53}$$

and Eq. (5 - 2) as

$$\mathbf{H}(k-1)\mathbf{x}(k-1) = \mathbf{A}(k-1)\mathbf{d}(k-1). \tag{5 - 54}$$

Substituting Eq. (5 - 54) into Eq. (5 - 53), we obtain the second recursive equation

$$\boxed{\mathbf{H}(k)\mathbf{x}(k) = \mathbf{H}(k-1)\mathbf{x}(k-1) + \mathbf{a}(k)d(k)}. \tag{5 - 55}$$

It is obvious that the third recursive equation is the transpose of the second recursive equation. Therefore, it can be expressed as

$$\boxed{\mathbf{x}^T(k)\mathbf{H}^T(k) = \mathbf{x}^T(k-1)\mathbf{H}^T(k-1) + d^T(k)\mathbf{a}^T(k)}. \tag{5 - 56}$$

We are now ready to form the last recursive equation. First, we left multiply Eq. (5 - 55) by $\mathbf{x}^T(k)$,

$$\mathbf{x}^T(k)\mathbf{H}(k)\mathbf{x}(k) = \mathbf{x}^T(k)\mathbf{H}(k-1)\mathbf{x}(k-1) + \mathbf{x}^T(k)\mathbf{a}(k)d(k). \tag{5 - 57}$$

If we define

$$\kappa^{-1}(k) = \frac{1}{1 + \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)}, \tag{5 - 58}$$

then we can express Eq. (5 - 24) as

$$\mathbf{x}(k) = \mathbf{x}(k-1) + \kappa^{-1}(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)\xi(k). \tag{5 - 59}$$

Note that $\overline{\mathbf{H}}(k-1)$ is symmetric, which implies $\overline{\mathbf{H}}^T(k-1) = \overline{\mathbf{H}}(k-1)$. Therefore, we can express the transpose of $\mathbf{x}(k)$ as

$$\mathbf{x}^T(k) = \mathbf{x}^T(k-1) + \kappa^{-1}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\xi^T(k). \tag{5 - 60}$$

Left multiply both sides by $\mathbf{a}(k)$ to obtain

$$\mathbf{x}^T(k)\mathbf{a}(k) = \mathbf{x}^T(k-1)\mathbf{a}(k) + \kappa^{-1}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)\xi^T(k) \tag{5 - 61}$$

where $\xi^T(k) = d^T(k) - \mathbf{x}^T(k-1)\mathbf{a}(k)$. If we add $d^T(k)$ to both sides of the Eq. (5 - 61), we get

$$\begin{aligned}\mathbf{x}^T(k)\mathbf{a}(k) &= d^T(k) - (d^T(k) - \mathbf{x}^T(k-1)\mathbf{a}(k)) + \kappa^{-1}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)\xi^T(k)\\ &= d^T(k) - \kappa^{-1}(k)(\xi^T(k)\kappa(k) - \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)\xi^T(k))\\ &= d^T(k) - \kappa^{-1}(k)\xi^T(k)(1 + \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k) - \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k))\\ &= d^T(k) - \kappa^{-1}(k)\xi^T(k)\end{aligned} \tag{5 - 62}$$

Substitute Eq. (5 - 60) into the middle term of Eq. (5 - 57), we obtain

$$\begin{aligned}\mathbf{x}^T(k)\mathbf{H}(k)\mathbf{x}(k) &= \mathbf{x}^T(k-1)\mathbf{H}(k-1)\mathbf{x}(k-1) +\\ &\kappa^{-1}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\xi^T(k)\mathbf{H}(k-1)\mathbf{x}(k-1) + \mathbf{x}^T(k)\mathbf{a}(k)d(k).\end{aligned} \tag{5 - 63}$$

Now, if we substitute Eq. (5 - 62) into Eq. (5 - 63) and manipulate the equation as follows,

$$\begin{aligned}\mathbf{x}^T(k)\mathbf{H}(k)\mathbf{x}(k) &= \mathbf{x}^T(k-1)\mathbf{H}(k-1)\mathbf{x}(k-1) + \kappa^{-1}(k)\mathbf{a}^T(k)\mathbf{x}(k-1)\xi^T(k) +\\ &\quad + (d^T(k) - \kappa^{-1}(k)\xi^T(k))d(k)\\ \mathbf{x}^T(k)\mathbf{H}(k)\mathbf{x}(k) &= \mathbf{x}^T(k-1)\mathbf{H}(k-1)\mathbf{x}(k-1) + d^T(k)d(k) +\\ &\quad + \kappa^{-1}(k)\xi^T(k)(\underbrace{\mathbf{a}^T(k)\mathbf{x}(k-1) - d(k)}_{-\xi(k)})\end{aligned} \tag{5 - 64}$$

we obtain the final recursive equation

$$\boxed{\mathbf{x}^T(k)\mathbf{H}(k)\mathbf{x}(k) + \kappa^{-1}(k)\xi^T(k)\xi(k) = \mathbf{x}^T(k-1)\mathbf{H}(k-1)\mathbf{x}(k-1) + d^T(k)d(k)}. \tag{5 - 65}$$

Before proceeding to the derivation of the QR-RLS algorithm, we first state a matrix factorization result that plays an important role in the derivation.

Lemma 5 - 1 Matrix Factorization Lemma

Given the data matrix $\mathbf{A}(k) \in \Re^{m \times n}$ and an upper triangular matrix $\mathbf{R}(k) \in \Re^{n \times n}$ with $n \leq m$, the matrix factorization lemma states that

$$\mathbf{A}(k)^T\mathbf{A}(k) = \mathbf{R}^T(k)\mathbf{R}(k) \tag{5 - 66}$$

if and only if there exist an orthogonal matrix $\mathbf{Q}(k) \in \Re^{m \times n}$ such that

$$\mathbf{A}(k) = \mathbf{Q}(k)\mathbf{R}(k). \tag{5 - 67}$$

Proof. *This proof is shown in (Stewart 1973, Golub & Van Loan 1996, Sayed & Kailath 1994, Haykin 1996). Nevertheless, we will repeat it here. Assume that the condition Eq. (5 - 67) holds, then by multiplying the* $\mathbf{A}^T(k)$ *times both sides, and substituting Eq. (5 - 67) into the right hand side, we get*

$$\mathbf{A}^T(k)\mathbf{A}(k) = \mathbf{R}^T(k)\mathbf{Q}^T(k)\mathbf{Q}(k)\mathbf{R}(k). \tag{5 - 68}$$

Since $\mathbf{Q}^T(k)\mathbf{Q}(k) = \mathbf{I}$*, we obtain Eq. (5 - 66).*

The converse implication is proof by invoking the singular value decomposition (SVD) theorem. According to the SVD theorem (Golub & Van Loan 1996),

$$\mathbf{A}(k) = \mathbf{U}_A(k)\mathbf{\Sigma}_A(k)\mathbf{V}_A^T(k) \tag{5 - 69}$$

where $\mathbf{U}_A(k)$ *and* $\mathbf{V}_A(k)$ *are n-by-n and m-by-m unitary matrices, respectively and* $\mathbf{\Sigma}_A(k)$ *is an n-by-m matrix defined by the singular values of the matrix* $\mathbf{A}(k)$*. Similarly,* $\mathbf{R}(k)$ *can be factored as*

$$\mathbf{R}(k) = \mathbf{U}_B(k)\mathbf{\Sigma}_B(k)\mathbf{V}_B^T(k). \tag{5 - 70}$$

Eq. (5 - 66) implies that we have

$$\mathbf{U}_A(k) = \mathbf{U}_B(k) \tag{5 - 71}$$

and

$$\mathbf{\Sigma}_A(k) = \mathbf{\Sigma}_B(k) \tag{5 - 72}$$

Now, let

$$\mathbf{Q}(k) = \mathbf{U}_A(k)\mathbf{U}_B^T(k) \tag{5 - 73}$$

and substitute Eq. (5 - 71) and Eq. (5 - 72) into Eq. (5 - 69). This produces

$$\mathbf{A}(k) = \mathbf{U}_B(k)\mathbf{\Sigma}_B(k)\mathbf{V}_A^T(k). \tag{5 - 74}$$

Now multiply Eq. (5 - 73) times Eq. (5 - 70) and we get

$$\mathbf{Q}(k)\mathbf{R}(k) = \mathbf{U}_A(k)\mathbf{\Sigma}_B(k)\mathbf{V}_B^T(k) = \mathbf{A}(k). \tag{5 - 75}$$

5.4.3 Forming the QR-RLS Algorithm

So far, we have formed the necessary recursive equations and stated the matrix factorization lemma for the QR-RLS algorithm, but we have not talked about why we need these four particular equations Eq. (5 - 46) through Eq. (5 - 49). These four equations can be lumped together to form a natural positive definite squared matrix equality. Then, using the matrix factorization lemma, we can form a factored matrix equality. This factored matrix equality contains all the necessary parameters such as the correlation matrix $\mathbf{H}(k)$ and the parameter vector $\mathbf{x}(k)$, which are needed for the parameter updates.

Keeping in mind this general idea regarding what we are going to do next, we can now rewrite the four recursive equations as

$$\mathbf{R}^T(k)\mathbf{R}(k) = \mathbf{R}^T(k-1)\mathbf{R}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k), \tag{5 - 76}$$

$$\mathbf{R}^T(k)\mathbf{R}(k)\mathbf{x}(k) = \mathbf{R}^T(k-1)\mathbf{R}(k-1)\mathbf{x}(k-1) + \mathbf{a}(k)d(k), \tag{5 - 77}$$

$$\mathbf{x}^T(k)\mathbf{R}^T(k)\mathbf{R}(k) = \mathbf{x}^T(k-1)\mathbf{R}^T(k-1)\mathbf{R}(k-1) + d^T(k)\mathbf{a}^T(k) \tag{5 - 78}$$

$$\begin{aligned}\mathbf{x}^T(k)\mathbf{R}^T(k)\mathbf{R}(k)\mathbf{x}(k) + \kappa^{-T/2}(k)\xi^T(k)\kappa^{-1/2}(k)\xi(k) = \\ \mathbf{x}^T(k-1)\mathbf{R}^T(k-1)\mathbf{R}(k-1)\mathbf{x}(k-1) + d^T(k)d(k)\end{aligned} \quad . \tag{5 - 79}$$

Because of the symmetry of the above equations, we may group these recursive equations into one matrix, which forms the following matrix equality:

$$\begin{aligned}&\left[\begin{array}{c|c} \mathbf{R}^T(k-1)\mathbf{R}(k-1) + \mathbf{a}(k)\mathbf{a}^T(k) & \mathbf{R}^T(k-1)\mathbf{R}(k-1)\mathbf{x}(k-1) + \mathbf{a}(k)d(k) \\ \hline \mathbf{x}^T(k-1)\mathbf{R}^T(k-1)\mathbf{R}(k-1) + d^T(k)\mathbf{a}^T(k) & \mathbf{x}^T(k-1)\mathbf{R}^T(k-1)\mathbf{R}(k-1)\mathbf{x}(k-1) + d^T(k)d(k) \end{array}\right] = \\ &\left[\begin{array}{c|c} \mathbf{R}^T(k)\mathbf{R}(k) & \mathbf{R}^T(k)\mathbf{R}(k)\mathbf{x}(k) \\ \hline \mathbf{x}^T(k)\mathbf{R}^T(k)\mathbf{R}(k) & \mathbf{x}^T(k)\mathbf{R}^T(k)\mathbf{R}(k)\mathbf{x}(k) + \kappa^{-T/2}(k)\xi^T(k)\kappa^{-1/2}(k)\xi(k) \end{array}\right]\end{aligned} \tag{5 - 80}$$

Now we may express the matrix equality in Eq. (5 - 80) in factored form as

$$\overbrace{\begin{bmatrix} \mathbf{R}^T(k-1) & \mathbf{a}(k) \\ \mathbf{x}^T(k-1)\mathbf{R}^T(k-1) & d^T(k) \end{bmatrix}}^{\mathbf{A}_1^T(k)} \overbrace{\begin{bmatrix} \mathbf{R}(k-1) & \mathbf{R}(k-1)\mathbf{x}(k-1) \\ \mathbf{a}^T(k) & d(k) \end{bmatrix}}^{\mathbf{A}_1(k)} = \underbrace{\begin{bmatrix} \mathbf{R}^T(k) & \mathbf{0} \\ \mathbf{x}^T(k)\mathbf{R}^T(k) & \kappa^{-T/2}(k)\xi^T(k) \end{bmatrix}}_{\mathbf{R}_1^T(k)} \underbrace{\begin{bmatrix} \mathbf{R}(k) & \mathbf{R}(k)\mathbf{x}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}}_{\mathbf{R}_1(k)} \qquad (5-81)$$

where $\mathbf{R}_1(k)$ is an upper triangular matrix. The above matrix equality fits Eq. (5 - 66); therefore, from the matrix factorization lemma, there exists an orthogonal matrix $\mathbf{Q}_1(k)$ that relates the block elements above as

$$\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1 \qquad (5-82)$$

$$\begin{bmatrix} \mathbf{R}(k-1) & \mathbf{g}(k-1) \\ \mathbf{a}^T(k) & d(k) \end{bmatrix} = \mathbf{Q}_1(k) \begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}, \qquad (5-83)$$

or

$$\boxed{\mathbf{Q}_1^T(k) \begin{bmatrix} \mathbf{R}(k-1) & \mathbf{g}(k-1) \\ \mathbf{a}^T(k) & d(k) \end{bmatrix} = \begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}} \qquad (5-84)$$

where $\mathbf{g}(k) = \mathbf{R}(k)\mathbf{x}(k)$ and $\mathbf{g}(k-1) = \mathbf{R}(k-1)\mathbf{x}(k-1)$.

The block elements shown in Eq. (5 - 84) form the backbone of the QR-RLS algorithm. The main idea of this equation is to put all the recursive equations in the form of pre-array $\mathbf{A}_1(k)$ and post-array $\mathbf{R}_1(k)$ matrices.

The pre-array matrix $\mathbf{A}_1(k)$ (shown on the left hand side of Eq. (5 - 84)) consists of $\mathbf{R}(k-1)$, the past gain vector $\mathbf{g}(k-1)$, the current input data $\mathbf{a}^T(k)$ and the current desired response $d(k)$. The pre-array matrix is not a triangular matrix, due to the non-zero elements in the current input data.

The post-array matrix $\mathbf{R}_1(k)$ (shown on the right hand side of Eq. (5 - 84)) is a upper triangular matrix. This post-array matrix is the result of the orthogonal matrix operating on the pre-array matrix.

When the current input data $\mathbf{a}^T(k)$ is presented to the algorithm, its elements are placed just underneath $\mathbf{R}(k-1)$ in the pre-array. Because the only non-triangular elements in the pre-array are the input data, the orthogonal matrix only needs to operate on the input data. The orthogonal matrix operates by annihilating the input data one by one until all elements become zero entries in the lower left of the matrix. As soon as the all the input elements annihilated, a triangular matrix is obtained. This triangular matrix is called the post-array matrix and it consists of current $\mathbf{R}(k)$, the current gain vector $\mathbf{g}(k)$ and the term $\kappa^{-1/2}(k)\xi(k)$. Once the post-array is found, the computed $\mathbf{R}(k)$ and $\mathbf{g}(k)$ are substituted back to the pre-array to initiate the next iteration.

Having computed the updated block values $\mathbf{R}(k)$ and $\mathbf{g}(k)$, we may solve the least squares parameter vector $\mathbf{x}(k)$ using the formula

$$\mathbf{x}(k) = \mathbf{R}^{-1}(k)\mathbf{g}(k). \tag{5 - 85}$$

Since, $\mathbf{R}(k)$ is a triangular matrix, $\mathbf{x}(k)$ is easily solved using the method of back substitution.

5.4.4 Orthogonal Matrix Operation

So far, we have not focused on the detailed operation of the orthogonal matrix. We only know that we need to choose the orthogonal matrix so that it will produce a triangular block of zeros in the lower left of the post-array. An orthogonal matrix that fits this requirement is the Givens rotation. Through successive applications of a sequence of Givens rotations, we can develop a systematic annihilation process to zero-out non-zero elements in the lower triangle of the pre-array as prescribed in Eq. (5 - 83) or Eq. (5 - 84). Please refer to Chapter 4.5 for a detailed discussion of how the Givens rotation operates. Since the pre-array has a unique structure, where $\mathbf{R}(k-1)$ is already a triangular matrix and $\mathbf{a}^T(k)$ contains the only non-zero elements in the lower left triangle, we only need to annihilate the elements in $\mathbf{a}^T(k)$ to achieve the triangular structure of the post-array. We illustrate this idea in the following:

$$\begin{bmatrix} \mathbf{R}(k-1) & \mathbf{g}(k-1) \\ \mathbf{a}^T(k) & d(k) \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & \times & \times & \times \end{bmatrix} \to$$

$$\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} = \begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}$$

[shaded] the affected elements

Figure 5 - 1 Givens rotations applied to the pre-array in the QR-RLS algorithm

As shown in Figure 5 - 1, we annihilate the current input data $\mathbf{a}^T(k)$ from left to right. The Givens QR algorithm described in Chapter 4.5 is used, except that we skip all zero elements contained in $\mathbf{R}(k-1)$. This reduces the total amount of computation by an order of magnitude. The floating point computation is $O(n^2)$ compare to $O(n^3)$ if we were to annihilate the entire lower triangle.

Once the post-array matrix is found, the next iteration is initiated by substituting the elements $\mathbf{R}(k)$ and $\mathbf{g}(k)$ back into the pre-array, together with the next input data $\mathbf{a}^T(k+1)$ and next desired responses $d(k+1)$ as shown in Figure 5 - 2. (Note that the detailed Givens rotations operations are omitted)

$$\begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{a}^T(k+1) & d(k+1) \end{bmatrix} = \ldots \to = \begin{bmatrix} \mathbf{R}(k+1) & \mathbf{g}(k+1) \\ \mathbf{0}^T & \kappa^{-1/2}(k+1)\xi(k+1) \end{bmatrix}$$

Figure 5 - 2 QR-RLS algorithm for the next iteration

5.4.5 Implementation Considerations

To initialize the QR-RLS algorithm, we may set $\mathbf{R}(0) = \mathbf{0}$ and $\mathbf{g}(0) = \mathbf{0}$. This is the exact initialization of the QR-RLS algorithm, which covers the time period $0 \le k \le M$. Note that the soft initialization used by the RLS is not necessary, because the QR-RLS algorithm does not incur any problem when the initial values are set to zeros. With this initialization process, the QR-RLS can operate in real-time by substituting the elements ($\mathbf{R}(k)$ and $\mathbf{g}(k)$) found in the post-array back into the pre-array with a new input vector $\mathbf{a}^T(k)$ and a new desired response $d(k)$. In general, the QR-RLS algorithm is considered a better numerical procedure than the RLS algorithm because of the following properties:

1. It works directly with the incoming data vector rather than working with the correlation matrix of the input data as in the standard RLS algorithm (Gentleman & Kung 1981, Haykin 1991).
2. It uses the numerically well-behaved Givens rotation, which preserves the two-norms of the least squares solution.
3. It propagates $\mathbf{R}(k)$ rather than $\mathbf{H}(k)$ or $\overline{\mathbf{H}}(k)$. Since the condition number of $\mathbf{R}(k)$ equals the condition number of $\mathbf{A}(k)$, it results in a significant reduction in the dynamic range of the data handled by the QR-RLS.

With this numerically stable QR-RLS for the time-update algorithm, we will later discuss how we can couple this time-update algorithm and the order-update algorithm presented in Chapter 6 to form the QR-RLS-AWS algorithm.

5.5 Results Summary

Recursive Least Squares Algorithm

Initialize the algorithm by setting $\overline{\mathbf{H}}(0) = \delta^{-1}\mathbf{I}$, *and* $\mathbf{x}(0) = 0$

For $k = 1, 2, ...$,

read $\mathbf{a}(k)$ *and* $d(k)$

$$\mathbf{k}(k) = \frac{\overline{\mathbf{H}}(k-1)\mathbf{a}(k)}{1 + \mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)\mathbf{a}(k)}$$

$$\xi(k) = d(k) - \mathbf{a}^T(k)\mathbf{x}(k-1)$$

$$\mathbf{x}(k) = \mathbf{x}(k-1) + \mathbf{k}(k)\xi(k)$$

$$\overline{\mathbf{H}}(k) = \overline{\mathbf{H}}(k-1) - \mathbf{k}(k)\mathbf{a}^T(k)\overline{\mathbf{H}}(k-1)$$

QR-Recursive Least Squares Algorithm

Initialize the algorithm by setting $\mathbf{R}(0) = \mathbf{0}$, *and* $\mathbf{g}(0) = \mathbf{0}$

For $k = 1, 2, ...$,

read $\mathbf{a}(k)$ *and* $d(k)$

$$\mathbf{Q}_1^T(k)\begin{bmatrix} \mathbf{R}(k-1) & \mathbf{g}(k-1) \\ \mathbf{a}^T(k) & d(k) \end{bmatrix} = \begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}$$

Givens rotations are used to annihilate the $\mathbf{a}^T(k)$

$$\mathbf{x}(k) = \mathbf{R}^{-1}(k)\mathbf{g}(k)$$

Chapter 6

Order-Update and Subset Selection

6.1	Introduction	90
6.2	Order-Update Algorithms	91
	6.2.1 Block Matrix Inversion Lemma	92
	6.2.2 Recursive Order-Update Algorithm for LS Method	93
	6.2.3 Recursive QR Order-Update Algorithm	101
6.3	Subset Selection	105
	6.3.1 Background	105
	6.3.2 Comparison of Subset Selection Methods	108
	6.3.3 Forward Selection and Order-Update	108

In this chapter, we first develop the recursive order-update algorithms for the linear model using the least squares method and the orthogonal least squares method. Then, we introduce the concept of subset selection. Among these subset selection methods, we discuss the forward selection in detail. Later, we discuss how we can use the recursive order-update algorithms in the forward selection method. These combined algorithms, together with time-update algorithms (discussed in chapter 5), will be used in the next chapter to create the RLS-AWS and QR-RLS-AWS algorithms.

6.1 Introduction

In Chapter 5, we discussed procedures for updating the parameters of a linear model as each new data point is received. These procedures, called time-update algorithms, assume that the order (size) of the linear model remains the same. In this chapter, we will derive and analyze recursive procedures for updating the parameters of a linear model when the order of the model is increased (a new basis function is added). These order-update algorithms assume that no new data are received. In Chapter 7, we will combine the time-update algorithms and the order-update algorithms to form the complete RLS-AWS and the QR-RLS-AWS algorithms.

In addition to the order-update algorithms, this chapter will also discuss subset selection methods. Before an order-update is made, we need to select the appropriate data vector (or basis function center in the case of RBF networks) to use for the additional order. The process of selecting the data vector is called subset selection.

The order-update algorithm is originally derived from a standard least squares perspective. However, due to numerical round-off error, an improved version of this algorithm is developed based on the QR decomposition.

We will begin with a brief overview of subset selection methods. Then, we will focus on the forward selection method and will discuss it in detail. Later, we point out that the order-update algorithm can be used as the update mechanism for the forward selection method. Together, the recursive order-update algorithm and the forward subset selection

form a complete order-update algorithm, which can be used together with the time-update algorithm of Chapter 5 to form a complete adaptive training procedure.

To ease the derivations, in the rest of this chapter we introduce the subscript notation q, which denotes the current order.

6.2 Order-Update Algorithms

In this section, we will derive recursive order-update algorithms for the linear model using the batch least squares method. These methods allow us to efficiently recalculate the new least squares solution when a new data vector is added into the model. Suppose we have a linear model

$$\mathbf{A}_q\mathbf{x}_q = \mathbf{d} \tag{6 - 1}$$

where the data matrix $\mathbf{A}_q$ and the parameter vector $\mathbf{x}_q$ are given by:

$$\begin{aligned} \mathbf{A}_q &= \begin{bmatrix}\mathbf{a}_1 \ \mathbf{a}_2 \ \dots \ \mathbf{a}_q\end{bmatrix} \\ \mathbf{x}_q &= \begin{bmatrix}x_1 \ x_2 \ \dots \ x_q\end{bmatrix}^T. \end{aligned} \tag{6 - 2}$$

The batch least squares solution of this linear model is

$$\mathbf{x}_q = (\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{d}. \tag{6 - 3}$$

Note that since the order of the model (the dimension of $\mathbf{x}_q$) does not affect the desired response, $\mathbf{d}$, there is no subscript q attached.

Now, suppose a new input data vector $\mathbf{a}_{q+1}$ becomes available (a new basis function is added), and we would like to add it into the linear model Eq. (6 - 1). This new

input data vector is incorporated by adding $\mathbf{a}_{q+1}$ into the columns of the data matrix $\mathbf{A}_q$, and the size of the old parameter $\mathbf{x}_q$ has to increase by one to accommodate the added data vector:

$$\begin{aligned} \mathbf{A}_{q+1} &= \left[\mathbf{A}_q \ \mathbf{a}_{q+1}\right] \\ \mathbf{x}_{q+1} &= \left[\tilde{\mathbf{x}}_q \ x_{q+1}\right]^T \end{aligned} \quad (6 - 4)$$

Note that $\tilde{\mathbf{x}}_q$ represents the updated least squares solution of the new linear model that is updated from the old parameter $\mathbf{x}_q$. Eq. (6 - 4) forms a new linear model

$$\mathbf{A}_{q+1}\mathbf{x}_{q+1} = \mathbf{d}, \quad (6 - 5)$$

and the least squares solution of this new model is

$$\mathbf{A}_{q+1}^T\mathbf{A}_{q+1}\mathbf{x}_{q+1} = \mathbf{A}_{q+1}^T\mathbf{d}. \quad (6 - 6)$$

We could compute $\mathbf{x}_{q+1}$ using Eq. (6 - 6), but this would be time consuming. An alternative is to recursively compute $\mathbf{x}_{q+1}$ based on the previously computed parameter vector $\mathbf{x}_q$. This recursive solution can be obtained by using a basic result in block matrix algebra known as the block matrix inversion lemma.

6.2.1 Block Matrix Inversion Lemma

Before we derive the recursive order-update algorithms, we introduce the block matrix inversion lemma. This lemma plays an important role in the derivation of the recursive order-update algorithm. Let $\mathbf{H} \in \Re^{(n+m)\times(n+m)}$ be a square matrix such that

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \tag{6 - 7}$$

where $\mathbf{A} \in \Re^{n \times n}$, $\mathbf{B} \in \Re^{n \times m}$, $\mathbf{C} \in \Re^{m \times n}$ and $\mathbf{D} \in \Re^{m \times m}$, then

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \\ -(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \end{bmatrix} \tag{6 - 8}$$

provided that $|\mathbf{A}| \neq \mathbf{0}$ and $|\mathbf{D}| \neq \mathbf{0}$ (Ogata 1987). In the special case where **B** and **C** are vectors (denoted as $\mathbf{b}$ and $\mathbf{c}^T$ respectively) and **D** is a scalar, d, Eq. (6 - 8) simplifies to

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{b}(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}\mathbf{c}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{b}(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1} \\ -(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}\mathbf{c}^T\mathbf{A}^{-1} & (d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1} \end{bmatrix} \tag{6 - 9}$$

where $(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}$ is a scalar and the inversion is just a division. Readers can refer to Duncan (1944) or Hager (1989) for a proof of this block matrix inversion lemma.

6.2.2 Recursive Order-Update Algorithm for LS Method

In the following, we apply the block matrix inversion lemma to the batch least squares algorithm, and obtain a recursive order-update algorithm. We first note that the new linear model in Eq. (6 - 5) can be written as

$$\begin{bmatrix} \mathbf{A}_q & \mathbf{a}_{q+1} \end{bmatrix} \mathbf{x}_{q+1} = \mathbf{d}. \tag{6 - 10}$$

and the least squares solution to this new linear model is the solution of

$$\begin{bmatrix} \mathbf{A}_q^T \\ \mathbf{a}_{q+1}^T \end{bmatrix} \begin{bmatrix} \mathbf{A}_q & \mathbf{a}_{q+1} \end{bmatrix} \mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{A}_q^T \\ \mathbf{a}_{q+1}^T \end{bmatrix} \mathbf{d}$$

$$\begin{bmatrix} \mathbf{A}_q^T\mathbf{A}_q & \mathbf{A}_q^T\mathbf{a}_{q+1} \\ \mathbf{a}_{q+1}^T\mathbf{A}_q & \mathbf{a}_{q+1}^T\mathbf{a}_{q+1} \end{bmatrix} \mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{A}_q^T\mathbf{d} \\ \mathbf{a}_{q+1}^T\mathbf{d} \end{bmatrix} \qquad (6 - 11)$$

As mentioned previously, we can take the inverse of the left block matrix in Eq. (6 - 11) and recalculate the whole solution as shown below,

$$\mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{A}_q^T\mathbf{A}_q & \mathbf{A}_q^T\mathbf{a}_{q+1} \\ \mathbf{a}_{q+1}^T\mathbf{A}_q & \mathbf{a}_{q+1}^T\mathbf{a}_{q+1} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{A}_q^T\mathbf{d} \\ \mathbf{a}_{q+1}^T\mathbf{d} \end{bmatrix}, \qquad (6 - 12)$$

but it is impractical and time consuming. A better way of obtaining the solution is to apply the block matrix inversion lemma and find a recursive updating formula for Eq. (6 - 12).

6.2.2.1 Order-Update for the Parameter

By applying the block matrix inversion lemma we can express the inverse of the block matrix in Eq. (6 - 12) as

$$\overline{\mathbf{H}}_{q+1} = \begin{bmatrix} \mathbf{A}_q^T\mathbf{A}_q & \mathbf{A}_q^T\mathbf{a}_{q+1} \\ \mathbf{a}_{q+1}^T\mathbf{A}_q & \mathbf{a}_{q+1}^T\mathbf{a}_{q+1} \end{bmatrix}^{-1} =$$

$$\left[\begin{array}{c|c} (\mathbf{A}_q^T\mathbf{A}_q)^{-1} + (\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1} & -(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2} \\ \hline -\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1} & \rho_q^{-2} \end{array}\right] \qquad (6 - 13)$$

where

$$\boxed{\rho_q^2 = \mathbf{a}_{q+1}^T\mathbf{a}_{q+1} - \mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}}. \qquad (6 - 14)$$

Now, we can apply the inversion result of Eq. (6 - 13) to Eq. (6 - 12),

$$\mathbf{x}_{q+1} = \begin{bmatrix} (\mathbf{A}_q^T\mathbf{A}_q)^{-1} + (\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1} & -(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2T} \\ -\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1} & \rho_q^{-2} \end{bmatrix} \begin{bmatrix} \mathbf{A}_q^T \\ \mathbf{a}_{q+1}^T \end{bmatrix} \mathbf{d} \quad (6 - 15)$$

After some algebra simplification, we obtain

$$\mathbf{x}_{q+1} = \begin{bmatrix} (\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{d} - (\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T(\mathbf{d} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{d}) \\ \rho_q^{-2}\mathbf{a}_{q+1}^T(\mathbf{d} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{d}) \end{bmatrix} . \quad (6 - 16)$$

Since the optimal parameter and the error vector at order index q are $\mathbf{x}_q = (\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{d}$ and $\mathbf{e}_q = \mathbf{d} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{d}$, we can further reduce the solution to

$$\boxed{\mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{x}_q \\ 0 \end{bmatrix} + \begin{bmatrix} -(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q \\ \rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q \end{bmatrix}} \quad (6 - 17)$$

where $\rho_q^2 = \mathbf{a}_{q+1}^T(\mathbf{a}_{q+1} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1})$ is a scalar. Note that the error vector can also be calculated as

$$\mathbf{e}_q = \mathbf{d} - \mathbf{A}_q\mathbf{x}_q . \quad (6 - 18)$$

Eq. (6 - 17) is the recursive order-update for the new linear model. It utilizes the old parameter vector, $\mathbf{x}_q$, the old error vector, $\mathbf{e}_q$, the old data matrix, $\mathbf{A}_q$, and the new data vector $\mathbf{a}_{q+1}$, which are readily available. The significance of this equation is that it is does not require a new matrix inversion (we assume that $(\mathbf{A}_q^T\mathbf{A}_q)^{-1}$ already exists). A detailed discussion of the algorithm implementation will be presented in Section 6.2.2.3.

6.2.2.2 Recursive Order-Update for the Sum of Squared Errors

In the following, we derive a recursive formula for the sum of squared errors. First, we multiply $\mathbf{A}_{q+1}$ times both sides of the Eq. (6 - 17), and we get

$$\mathbf{A}_{q+1}\mathbf{x}_{q+1} = \mathbf{A}_q\mathbf{x}_q - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q + \mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q. \quad (6 - 19)$$

When we subtract both sides of Eq. (6 - 19) from the desired response $\mathbf{d}$, and note that $\mathbf{d} - \mathbf{A}_q\mathbf{x}_q = \mathbf{e}_q$ and $\mathbf{d} - \mathbf{A}_{q+1}\mathbf{x}_{q+1} = \mathbf{e}_{q+1}$ are the error vectors at order index q and $q+1$ respectively, we obtain

$$\mathbf{e}_{q+1} = \mathbf{e}_q - (\mathbf{a}_{q+1} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1})\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q. \quad (6 - 20)$$

Since, $\mathbf{a}_{q+1}^T\mathbf{e}_q$ is a scalar,

$$(\mathbf{a}_{q+1}^T\mathbf{e}_q)^T = \mathbf{a}_{q+1}^T\mathbf{e}_q, \quad (6 - 21)$$

the transpose of the error vector $\mathbf{e}_{q+1}^T$ can be written as

$$\mathbf{e}_{q+1}^T = \mathbf{e}_q^T - (\mathbf{a}_{q+1}^T - \mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q. \quad (6 - 22)$$

The sum of squared errors $\mathbf{e}_{q+1}^T\mathbf{e}_{q+1}$ is

$$\begin{aligned}\mathbf{e}_{q+1}^T\mathbf{e}_{q+1} = {} & \mathbf{e}_q^T\mathbf{e}_q - \mathbf{e}_q^T(\mathbf{a}_{q+1} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1})\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q \\ & - (\mathbf{a}_{q+1}^T - \mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{e}_q(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q) + \\ & (\mathbf{a}_{q+1}^T - \mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)(\mathbf{a}_{q+1} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1})(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q)^2.\end{aligned} \quad (6 - 23)$$

Since

$$\mathbf{a}_{q+1} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T\mathbf{a}_{q+1} = (\mathbf{I} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{a}_{q+1}, \text{ and} \quad (6 - 24)$$

$$\mathbf{a}_{q+1}^T - \mathbf{a}_{q+1}^T\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T = \mathbf{a}_{q+1}^T(\mathbf{I} - \mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T), \quad (6 - 25)$$

we can rewrite Eq. (6 - 23) as

$$\begin{aligned}\mathbf{e}_{q+1}^T\mathbf{e}_{q+1} &= \mathbf{e}_q^T\mathbf{e}_q - \mathbf{e}_q^T(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{a}_{q+1}\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q \\ &- \mathbf{a}_{q+1}^T(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{e}_q(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q) + \\ \mathbf{a}_{q+1}^T(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)&(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{a}_{q+1}(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q)^2\end{aligned} \tag{6 - 26}$$

Note that since $(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T) = (\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)$, we can easily show that the projection of the error is the error itself

$$\begin{aligned}(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{e}_q &= (\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{d} \\ &= (\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{d} \\ &= \mathbf{e}_q\end{aligned} \tag{6 - 27}$$

Hence, we can further reduce Eq. (6 - 26) to

$$\begin{aligned}\mathbf{e}_{q+1}^T\mathbf{e}_{q+1} &= \mathbf{e}_q^T\mathbf{e}_q - \mathbf{a}_{q+1}^T\mathbf{e}_q(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q) \\ -\mathbf{e}_q^T\mathbf{a}_{q+1}(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q) &+ \mathbf{a}_{q+1}^T(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{a}_{q+1}(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q)^2\end{aligned} \tag{6 - 28}$$

In fact, because $\mathbf{a}_{q+1}^T\mathbf{e}_q = \mathbf{e}_q^T\mathbf{a}_{q+1}$ is a scalar, we can write

$$\begin{aligned}\mathbf{e}_{q+1}^T\mathbf{e}_{q+1} &= \mathbf{e}_q^T\mathbf{e}_q - 2\mathbf{a}_{q+1}^T\mathbf{e}_q(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q) + \\ &\mathbf{a}_{q+1}^T(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{a}_{q+1}(\rho_q^{-2}\mathbf{a}_{q+1}^T\mathbf{e}_q)^2\end{aligned} \tag{6 - 29}$$

Also, since $\mathbf{a}_{q+1}^T(\mathbf{I}-\mathbf{A}_q(\mathbf{A}_q^T\mathbf{A}_q)^{-1}\mathbf{A}_q^T)\mathbf{a}_{q+1} = \rho_q^2$, the whole equation reduces to

$$\boxed{\mathbf{e}_{q+1}^T\mathbf{e}_{q+1} = \mathbf{e}_q^T\mathbf{e}_q - \rho_q^{-2}(\mathbf{a}_{q+1}^T\mathbf{e}_q)^2}. \tag{6 - 30}$$

Let us denote $SSE_{q+1} = \mathbf{e}_{q+1}^T\mathbf{e}_{q+1}$, $SSE_q = \mathbf{e}_q^T\mathbf{e}_q$ and $err_q = \rho_q^{-2}(\mathbf{a}_{q+1}^T\mathbf{e}_q)^2$, then we can write

$$\boxed{SSE_{q+1} = SSE_q - err_q}. \tag{6 - 31}$$

The term

$$err_q = \rho_q^{-2}(\mathbf{a}_{q+1}^T \mathbf{e}_q)^2 \qquad (6 - 32)$$

is called the error reduction term since it measures the error reduction caused by the added data vector.

This recursive sum of squared errors formulation in Eq. (6 - 30) has three important implications. First, due to the fact that $(\mathbf{a}_{q+1}^T \mathbf{e}_q)^2$ and ρ_q^{-2} are non-negative scalars, the error reduction term err_q is a non-negative scalar.

$$\rho_q^{-2}(\mathbf{a}_{q+1}^T \mathbf{e}_q)^2 \geq 0 \qquad (6 - 33)$$

Second, the new sum of squared errors will always be less than or equal to the old sum of squared errors

$$SSE_{q+1} \leq SSE_q. \qquad (6 - 34)$$

Lastly, the error reduction term can never be greater than the old sum of squared errors.

$$SSE_q \geq \rho_q^{-2}(\mathbf{a}_{q+1}^T \mathbf{e}_q)^2. \qquad (6 - 35)$$

These implications imply that order-update will decrease the sum of squared errors provided that the added data vector $\mathbf{a}_{q+1}^T$ is selected properly. We will discuss how we can choose this added data vector in section 6.3. In the next section, we will discuss how to implement this recursive order-update algorithm.

6.2.2.3 Implementation Considerations

To facilitate the implementation of the algorithm, intermediate calculations such as the added data parameter $\mathbf{w}_q$ and the orthogonal complement projection of the added data $\hat{\mathbf{a}}_{q+1}$ are performed:

$$\mathbf{w}_q = \mathbf{H}_q \mathbf{A}_q^T \mathbf{a}_{q+1} \tag{6 - 36}$$

$$\hat{\mathbf{a}}_{q+1} = \mathbf{a}_{q+1} - \mathbf{A}_q \mathbf{w}_q . \tag{6 - 37}$$

With these definitions, we can now write the inverse correlation matrix update as

$$\overline{\mathbf{H}}_{q+1} = \begin{bmatrix} \overline{\mathbf{H}}_q + \rho_q^{-2} \mathbf{w}_q \mathbf{w}_q^T & -\mathbf{w}_q \rho_q^{-2} \\ -\rho_q^{-2} \mathbf{w}_q^T & \rho_q^{-2} \end{bmatrix} \tag{6 - 38}$$

and optimal parameter update as

$$\mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{x}_q \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{w}_q \rho_q^{-2} \mathbf{a}_{q+1}^T \mathbf{e}_q \\ \rho_q^{-2} \mathbf{a}_{q+1}^T \mathbf{e}_q \end{bmatrix} \tag{6 - 39}$$

where

$$\rho_q^2 = \mathbf{a}_{q+1}^T \hat{\mathbf{a}}_{q+1} . \tag{6 - 40}$$

Hence, we can recursively update these equations in the following sequences: Eq. (6 - 36), Eq. (6 - 37), Eq. (6 - 40), Eq. (6 - 38) and Eq. (6 - 39). The sum of squared errors and the next iteration error vector can be calculated using Eq. (6 - 31) and Eq. (6 - 18), respectively.

Example 6 - 1

Let $\mathbf{A}_1 = \begin{bmatrix}1 & 2 & 0\end{bmatrix}^T$ *and* $\mathbf{d} = \begin{bmatrix}5 & 5 & 1\end{bmatrix}^T$ *be the data vector and the desired response. Then, the inverse correlation matrix, the parameter vector and the error vector are calculated as* $\mathbf{H}_1 = 1/5$, $\mathbf{x}_1 = \mathbf{H}_1\mathbf{A}_1^T\mathbf{d} = 3$ *and* $\mathbf{e}_1 = \mathbf{d} - \mathbf{A}_1\mathbf{x}_1 = \begin{bmatrix}2 & -1 & 1\end{bmatrix}^T$.

Now, we would like to add a new vector $\mathbf{a}_2 = \begin{bmatrix}0 & 1 & -1\end{bmatrix}^T$ *into* $\mathbf{A}_1$ *and form a new data matrix* $\mathbf{A}_2 = \begin{bmatrix}\mathbf{A}_1 & \mathbf{a}_2\end{bmatrix}$. *Using the recursive order-update algorithm, we can calculate* $\mathbf{w}_1 = \overline{\mathbf{H}}_1\mathbf{A}_1^T\mathbf{a}_2 = \frac{2}{5}$, $\hat{\mathbf{a}}_2 = \mathbf{a}_2 - \mathbf{A}_1\mathbf{w}_1 = \begin{bmatrix}-\frac{2}{5} & \frac{1}{5} & -1\end{bmatrix}^T$, $\rho_1^2 = \mathbf{a}_2^T\hat{\mathbf{a}}_2 = \frac{6}{5}$ *and update the error reduction term* $err_2 = \frac{10}{3}$. *Then, we can update the inverse correlation matrix* $\overline{\mathbf{H}}_2 = \frac{1}{6}\begin{bmatrix}2 & -2 \\ -2 & 5\end{bmatrix}$, *the parameter vector* $\mathbf{x}_2 = \frac{1}{3}\begin{bmatrix}11 \\ -5\end{bmatrix}$ *and the updated error vector* $\mathbf{e}_2 = \frac{1}{3}\begin{bmatrix}4 & -2 & -2\end{bmatrix}^T$. □

To calculate the inverse correlation matrix, we have to successively apply Eq. (6 - 38). If each successive expansion accumulates a small amount of rounding error, the inverse correlation matrix may lose its positive definiteness after many iterations and become unstable. Fletcher (1969) has suggested occasionally re-starting the recurrence, and Ben-Israel & Greville (1965) have suggested an iterative method to improve the numerical accuracy of the inverse correlation matrix. However, both suggestions may result

in more computation. Due to this fact, we have developed a recursive order-update algorithm based on the QR decomposition. This new algorithm is described in the following section.

6.2.3 Recursive QR Order-Update Algorithm

6.2.3.1 QR Recursive Order-Update for Q, R, and Parameter

By using the QR decomposition described in section 4.4.2, we can decompose the original data matrix $\mathbf{A}_q \in \Re^{m \times n}$ into

$$\mathbf{A}_q = \mathbf{Q}_q \mathbf{R}_q \tag{6 - 41}$$

where $\mathbf{Q}_q \in \Re^{m \times n}$ is an orthogonal matrix and $\mathbf{R}_q \in \Re^{n \times n}$ is an upper triangular matrix. Applying this QR decomposition to Eq. (6 - 11), we obtain

$$\begin{bmatrix} \mathbf{R}_q^T \mathbf{R}_q & \mathbf{R}_q^T \mathbf{Q}_q^T \mathbf{a}_{q+1} \\ \mathbf{a}_{q+1}^T \mathbf{Q}_q \mathbf{R}_q & \mathbf{a}_{q+1}^T \mathbf{a}_{q+1} \end{bmatrix} \mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{R}_q^T \mathbf{Q}_q^T \mathbf{d} \\ \mathbf{a}_{q+1}^T \mathbf{d} \end{bmatrix}. \tag{6 - 42}$$

Note that since $\mathbf{Q}_q$ is an orthogonal matrix, $\mathbf{Q}_q^T \mathbf{Q}_q = \mathbf{I}$, we can factor the left hand side matrix into

$$\begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix}^T \begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix} \mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{R}_q^T \mathbf{Q}_q^T \mathbf{d} \\ \mathbf{a}_{q+1}^T \mathbf{d} \end{bmatrix} \tag{6 - 43}$$

where $\mathbf{r}_q = \mathbf{Q}_q^T \mathbf{a}_{q+1}$, and $\rho_q = \sqrt{\mathbf{a}_{q+1}^T \mathbf{a}_{q+1} - \mathbf{r}_q^T \mathbf{r}_q}$. (In here, ρ_q is the square root of Eq. (6 - 14). We can show this fact by applying the QR decomposition to Eq. (6 - 14).) Eq. (6 - 43) also implies that

$$\mathbf{R}_{q+1} = \begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix}. \qquad (6\text{ - }44)$$

By applying the inverse twice to Eq. (6 - 43), we find the new parameter as

$$\mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix}^{-T} \begin{bmatrix} \mathbf{R}_q^T \mathbf{Q}_q^T \mathbf{d} \\ \mathbf{a}_{q+1}^T \mathbf{d} \end{bmatrix}. \qquad (6\text{ - }45)$$

Using the block matrix inversion lemma, we can simplify the inverse matrices to

$$\begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix}^{-T} = \begin{bmatrix} \bar{\mathbf{R}}_q^T & \mathbf{0} \\ -\rho_q^{-1} \mathbf{r}_q^T \bar{\mathbf{R}}_q^T & \rho_q^{-1} \end{bmatrix} = \bar{\mathbf{R}}_{q+1}^T \text{, and} \qquad (6\text{ - }46)$$

$$\boxed{\begin{bmatrix} \mathbf{R}_q & \mathbf{r}_q \\ \mathbf{0} & \rho_q \end{bmatrix}^{-1} = \begin{bmatrix} \bar{\mathbf{R}}_q & -\rho_q^{-1} \bar{\mathbf{R}}_q \mathbf{r}_q \\ \mathbf{0} & \rho_q^{-1} \end{bmatrix} = \bar{\mathbf{R}}_{q+1}}, \qquad (6\text{ - }47)$$

where $\bar{\mathbf{R}}_q = \mathbf{R}_q^{-1}$, and $\bar{\mathbf{R}}_{q+1} = \mathbf{R}_{q+1}^{-1}$. Substituting Eq. (6 - 46) and Eq. (6 - 47) into Eq. (6 - 45), we can simplify the parameter update equation to

$$\boxed{\mathbf{x}_{q+1} = \begin{bmatrix} \mathbf{x}_q - \bar{\mathbf{R}}_q \mathbf{r}_q \rho_q^{-2} \mathbf{a}_{q+1}^T \mathbf{e}_q \\ \rho_q^{-2} \mathbf{a}_{q+1}^T \mathbf{e}_q \end{bmatrix}} \qquad (6\text{ - }48)$$

where

$$\mathbf{r}_q = \mathbf{Q}_q^T \mathbf{a}_{q+1}, \qquad (6\text{ - }49)$$

$$\rho_q = \sqrt{\mathbf{a}_{q+1}^T \mathbf{a}_{q+1} - \mathbf{a}_{q+1}^T \mathbf{Q}_q \mathbf{Q}_q^T \mathbf{a}_{q+1}} = \sqrt{\mathbf{a}_{q+1}^T \mathbf{a}_{q+1} - \mathbf{r}_q^T \mathbf{r}_q} \text{, and} \qquad (6\text{ - }50)$$

$$\mathbf{e}_q = \mathbf{d} - \mathbf{A}_q \mathbf{x}_q. \qquad (6\text{ - }51)$$

Both Eq. (6 - 48) and Eq. (6 - 17) yield the same result except that Eq. (6 - 48) is computed using the QR decomposition, which does not suffer from the numerical inaccuracy of the previous algorithm.

In the following, we show that the orthogonal matrix $\mathbf{Q}_{q+1}$ can be updated recursively: Let $\mathbf{A}_{q+1} = \mathbf{Q}_{q+1}\mathbf{R}_{q+1}$, then

$$\begin{bmatrix}\mathbf{Q}_q\mathbf{R}_q & \mathbf{a}_{q+1}\end{bmatrix} = \mathbf{Q}_{q+1}\mathbf{R}_{q+1}. \qquad (6 - 52)$$

If we right multiply Eq. (6 - 52) by $\mathbf{R}_{q+1}^{-1}$, we obtain

$$\begin{bmatrix}\mathbf{Q}_q\mathbf{R}_q & \mathbf{a}_{q+1}\end{bmatrix}\mathbf{R}_{q+1}^{-1} = \mathbf{Q}_{q+1}. \qquad (6 - 53)$$

We note that $\overline{\mathbf{R}}_{q+1} = \mathbf{R}_{q+1}^{-1}$ and we can substitute Eq. (6 - 47) into Eq. (6 - 53),

$$\mathbf{Q}_{q+1} = \begin{bmatrix}\mathbf{Q}_q\mathbf{R}_q & \mathbf{a}_{q+1}\end{bmatrix}\begin{bmatrix}\overline{\mathbf{R}}_q & -\rho_q^{-1}\overline{\mathbf{R}}_q\mathbf{r}_q \\ \mathbf{0} & \rho_q^{-1}\end{bmatrix} \qquad (6 - 54)$$

which simplifies to

$$\boxed{\mathbf{Q}_{q+1} = \begin{bmatrix}\mathbf{Q}_q \,\vert\, \rho_q^{-1}(\mathbf{a}_{q+1} - \mathbf{Q}_q\mathbf{r}_q)\end{bmatrix}}. \qquad (6 - 55)$$

Meanwhile, using a similar derivation to the one used in section 6.2.2.2, we obtain a recursive sum of squared errors formula, as in Eq. (6 - 30). The only difference is in the calculation of ρ_q, which uses the orthogonal matrix in Eq. (6 - 50).

6.2.3.2 Implementation Considerations

Eq. (6 - 30), Eq. (6 - 47), Eq. (6 - 55), and Eq. (6 - 48) are calculated one by one to update the order. The following example illustrates the calculation.

Example 6 - 2

We repeat Example 6 - 1 with $\mathbf{A}_1 = \begin{bmatrix}1 & 2 & 0\end{bmatrix}^T$, $\mathbf{d} = \begin{bmatrix}5 & 5 & 1\end{bmatrix}^T$, $\mathbf{x}_1 = 3$, *and* $\mathbf{e}_1 = \begin{bmatrix}2 & -1 & 1\end{bmatrix}^T$. *Then, we can find* $\mathbf{Q}_1 = \frac{1}{\sqrt{5}}\begin{bmatrix}1 & 2 & 0\end{bmatrix}^T$, *and* $\overline{\mathbf{R}}_1 = \frac{1}{\sqrt{5}}$. *Again, we would like to add* $\mathbf{a}_2 = \begin{bmatrix}0 & 1 & -1\end{bmatrix}^T$ *into the model to form a new data vector* $\mathbf{A}_2 = \begin{bmatrix}\mathbf{A}_1 & \mathbf{a}_2\end{bmatrix}$.

Now, we find $\mathbf{r}_1 = \mathbf{Q}_1^T\mathbf{a}_2 = \frac{2}{\sqrt{5}}$, *and* $\rho_1 = \sqrt{\mathbf{a}_2^T\mathbf{a}_2 - \mathbf{r}_1^T\mathbf{r}_1} = \sqrt{\frac{6}{5}}$ *which we can use in calculating the error reduction term* err_1 *and the new inverse* $\overline{\mathbf{R}}_2$:

$$err_1 = \rho_1^{-2}(\mathbf{a}_2^T\mathbf{e}_1)^2 = \frac{10}{3},\ \text{and}\ \overline{\mathbf{R}}_2 = \begin{bmatrix}\overline{\mathbf{R}}_1 & -\rho_1^{-1}\overline{\mathbf{R}}_1\mathbf{r}_1 \\ \mathbf{0} & \rho_1^{-1}\end{bmatrix} = \frac{1}{\sqrt{5}}\begin{bmatrix}1 & -\frac{2}{\sqrt{6}} \\ 0 & \frac{5}{\sqrt{6}}\end{bmatrix}.$$

Then, we use the $\overline{\mathbf{R}}_1$ *to update the parameter vector*

$$\mathbf{x}_2 = \begin{bmatrix}\mathbf{x}_1 - \overline{\mathbf{R}}_1\mathbf{r}_1\rho_1^{-2}\mathbf{a}_2^T\mathbf{e}_1 \\ \rho_1^{-2}\mathbf{a}_2^T\mathbf{e}_1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}11 \\ -5\end{bmatrix}.$$

Finally, we calculate the new error vector $\mathbf{e}_2$ *and update the orthogonal matrix* $\mathbf{Q}_2$ *for the next iteration:* $\mathbf{e}_2 = \frac{1}{3}\begin{bmatrix}4 & -2 & -2\end{bmatrix}^T$, *and* $\mathbf{Q}_2 = \left[\mathbf{Q}_1 \,\vdots\, \rho_1^{-1}(\mathbf{a}_2 - \mathbf{Q}_1\mathbf{r}_1)\right] = \begin{bmatrix}1 & -\frac{2}{\sqrt{6}} \\ 2 & \frac{1}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{6}}\end{bmatrix}$. □

Since the algorithm uses the QR decomposition to calculate the inverse, it is numerically more accurate than the algorithm in section 6.2.2. The only drawback in this algorithm is that it requires extra storage for the orthogonal matrix $\mathbf{Q}_q$.

6.3 Subset Selection

6.3.1 Background

In section 6.2, we developed an efficient algorithm for increasing the order of the model by adding a data vector $\mathbf{a}_{q+1}$. In this section, we will concentrate on how to choose which data vector to add. If the wrong data vector is chosen, it will only reduce the squared error by a small amount. The correct data vector will result in the largest reduction in the squared error.

To find good data vectors, we turn our attention to a statistical method known as subset selection (Weisberg 1980, Cohen 1983, Miller 1990). The objective of subset selection, as its name implies, is to select a small subset of input data vectors from a larger set. Ideally, subset selection ensures that we select an optimal set of data vectors for the

linear model. Simultaneously, it excludes many data vectors that do not affect the desired response. The following figure demonstrates this idea.

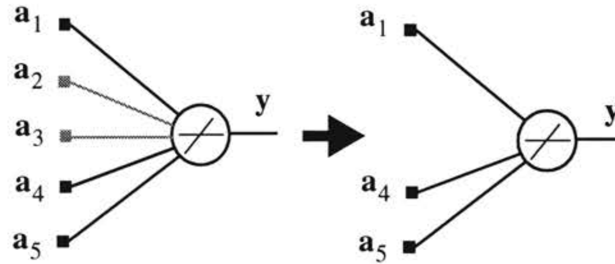


Figure 6 - 1 Ideal Subset Selection

Suppose we have input data vectors $\mathbf{a}_1 \ldots \mathbf{a}_5$ and we know that $\mathbf{a}_2$ and $\mathbf{a}_3$ are not contributing significantly to the model. Hence, it is best to exclude these two data vectors from the model. By excluding the less significant data vectors, the model will not only compute faster, but also requires fewer parameters. It is generally best to use the simplest model that explains the data.

In the following, we will summarize several commonly used subset selection methods, which have been documented in several journals and books. It should be noted there are many more subset selection algorithms (Dixon *et al.* 1988, Miller 1984, 1990, SAS 1985) than are discussed in this chapter, but we choose to limit our presentation here.

Exhaustive Search Method:

The exhaustive search method evaluates the sum of squared errors for all combinations of data vectors and selects the subset with the minimum error. When the number of data vectors is large, this algorithm becomes too expensive. Some authors feel that it is not useful to look at all possible models, since some models would not be meaningful (Draper & Smith 1981). Therefore, several algorithms have been developed

which find the best subset without computing all possible models (Furnival & Wilson 1974, Hocking 1976). However, like the exhaustive search method, there are many possible combinations that have to be sifted through when the number of data vectors is large (Hoerl *et al.* 1986). In general, exhaustive search algorithms are considered too computational intensive for large data sets.

Backward Elimination Method:

The backward elimination method starts with a model that includes all possible input data vectors. Then, it eliminates one input data vector at a time from the model. At each elimination step, the eliminated input data vector is selected in such a way that it results in the smallest increase in the sum of squared errors of the model. This elimination process continues until a stopping rule is satisfied.

Forward Selection Method

The forward selection method begins with no data vector in the model. Then, it moves data vectors into the model one at a time from a set of input data vectors. At each step, forward selection finds one data vector from all possible input data vectors and moves it into the model. The criterion for the selected data vector is that it will produce the largest reduction in the sum of squared errors when moved into the model. Forward selection continues until a stopping rule is satisfied.

Stepwise (Efroymson's) Method:

In forward selection, a data vector selected at an early stage may become unimportant in a later stage. Similarly, in backward elimination, a data vector deleted at an early stage may become important in a later stage. Hence, an idea to combine these two

methods is proposed by Efroymson (1960). After each data vector is added into the model, a test is made to see if any of the data vectors in the model can be deleted without appreciably increasing the sum of squared errors.

6.3.2 Comparison of Subset Selection Methods

Several extensive studies of these subset selection methods are conducted in (Miller 1990, Biondini *et al.* 1977, Derksen 1992). In these studies, the exhaustive search method and the backward elimination method are the worst performers in terms of computational time. Forward selection gives the fastest results, but not necessarily the optimal subset. The stepwise method gives the most accurate results, but requires more computation than forward selection. In fact, Berk (1978) has empirically shown that the average difference between the sum of squared errors of the stepwise and extensive search methods rarely exceeded 7 percent. Although the stepwise method yields better results than forward selection, for simplicity and computational reasons, we will only consider the forward selection method here.

6.3.3 Forward Selection and Order-Update

In the following, we will first discuss the forward selection method in detail and then explain its relationship to the recursive order-update method of section 6.2. The forward selection method assumes that there exists a large input data set $\underline{\mathbf{A}} = \left[\underline{\mathbf{a}}_1 \; \underline{\mathbf{a}}_2 \; \cdots \; \underline{\mathbf{a}}_n\right]$, where all the data vectors in the set are potential candidates for the

model. To begin, it finds the first data vector, $\underline{\mathbf{a}}_i$, that produces the largest reduction in the sum of squared errors

$$J = \mathbf{e}^T\mathbf{e} = (\mathbf{d} - \underline{\mathbf{a}}_i x)^T(\mathbf{d} - \underline{\mathbf{a}}_i x), \tag{6 - 56}$$

where the least squares solution is given as

$$x = (\underline{\mathbf{a}}_i^T\underline{\mathbf{a}}_i)^{-1}\underline{\mathbf{a}}_i^T\mathbf{d}. \tag{6 - 57}$$

According to Miller (1990), Eq. (6 - 56) can be rewritten as

$$J = \mathbf{e}^T\mathbf{e} = \mathbf{d}^T\mathbf{d} - \frac{(\underline{\mathbf{a}}_i^T\mathbf{d})^2}{\underline{\mathbf{a}}_i^T\underline{\mathbf{a}}_i}. \tag{6 - 58}$$

It is clear from Eq. (6 - 58) that the first data vector selected has to maximize the error reduction term

$$err = max\left[\frac{(\underline{\mathbf{a}}_i^T\mathbf{d})^2}{\underline{\mathbf{a}}_i^T\underline{\mathbf{a}}_i}\right] \quad i = 1\ldots n. \tag{6 - 59}$$

Suppose $\underline{\mathbf{a}}_1$ produces the largest error reduction, then $\underline{\mathbf{a}}_1$ will be removed from $\underline{\mathbf{A}}$ and added into the model $\mathbf{A}$. To find the next data vector, select another data vector, $\underline{\mathbf{a}}_j$, from $\underline{\mathbf{A}}$. Since the error vector $\mathbf{e} = \mathbf{d} - \underline{\mathbf{a}}_1 x$ is orthogonal to $\underline{\mathbf{a}}_1$, forward selection searches the space of $\underline{\mathbf{a}}_j$ that are orthogonal $\underline{\mathbf{a}}_1$ to find the amount of additional error reduction. Specifically, it forms

$$\underline{\mathbf{a}}_j = \underline{\mathbf{a}}_j - \underline{\mathbf{a}}_1(\underline{\mathbf{a}}_1^T\underline{\mathbf{a}}_1)^{-1}\underline{\mathbf{a}}_1^T\underline{\mathbf{a}}_j, \tag{6 - 60}$$

and substitutes it into Eq. (6 - 59) to find the data vector which maximizes the error reduction term. This calculation is repeated for each data vector in $\underline{\mathbf{A}}$, and the one that maximizes the error reduction is added into the model. This process is repeated until the total error reduction reaches a preselected value.

Note that we have referred to, Eq. (6 - 59) from the forward selection method and Eq. (6 - 30) from the order-update algorithm by the same name: error reduction. We can verify that these terms are equivalent by substituting Eq. (6 - 60) into Eq. (6 - 59) using $\underline{\mathbf{a}}_j = \mathbf{a}_{q+1}$ and $\underline{\mathbf{a}}_1 = \mathbf{A}_q$. Hence, the order-update algorithms discussed in section 6.2 can be used as the update mechanism for the forward selection method.

Currently, there are several variations for computing the error reduction term in Eq. (6 - 59): the Gauss-Jordan pivoting method (Miller 1990), and Modified/Classical Gram-Schmidt methods - also called the Orthogonal Least Squares (OLS) methods (Chen 1991). However, none of these methods have provided a convenient way of incorporating the time-update algorithms, which is why we have developed the order-update algorithms described in this chapter. By coupling the recursive order-update algorithms with the forward selection method, we have created two new methods which can update the forward selection using the parameters from the time-update algorithms. In the next chapter, we will show how we can combine forward selection method with RLS and QR-RLS methods.

Chapter 7

Time- and Order- Update

7.1 Introduction 112
7.2 The Subset Selection Model 113
7.3 Recursive Time- and Order- Update 116
7.3.1 Recursive Least Squares with Automatic Weight Selection (RLS-AWS) Algorithm 122
7.3.2 QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) Algorithm 125
7.4 Fixing Centers 127
7.4.1 Centers Selects from Fixed Range/Grid 127
7.4.2 Centers Selected from Time Point 128
7.4.3 Centers Selection for the New Algorithms 128
7.5 Preliminary Results 129
7.5.1 Compare QR-RLS-AWS and RLS-AWS 130
7.5.2 Accuracy Test 131
7.5.3 Batch and Recursive Test 132
7.6 Summary 133

This chapter proposes a new time- and order- update framework. This framework combines the time-update algorithms from chapter 5 and the order-update algorithms from chapter 6 and creates two new algorithms called the Recursive Least Squares with Automatic Weight Selection (RLS-AWS) algorithm and QR-Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) algorithm. Some preliminary results are discussed as well.

7.1 Introduction

In this chapter, we introduce two new algorithms called Recursive Least Squares with Automatic Weight Selection (RLS-AWS) algorithm and QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) algorithm. Both algorithms are based upon the time-update algorithms in Chapter 5 and the order-update algorithms in Chapter 6.

To explain how these new algorithms work, we will begin with a short discussion of the subset selection model. Then, the recursive time- and order- update framework are discussed. The actual implementation of the two new algorithms will be provided along with some preliminary results of these new algorithms.

For notation, we will combine the time-update notation described in Chapter 5 and order-update notation described in Chapter 6. The subscript q represents the order index, while the bracket (k) represents the time index. For example, $\mathbf{A}_q(k)$ represents the input data matrix at order q and time k. Detailed notation will be discussed in the following.

7.2 The Subset Selection Model

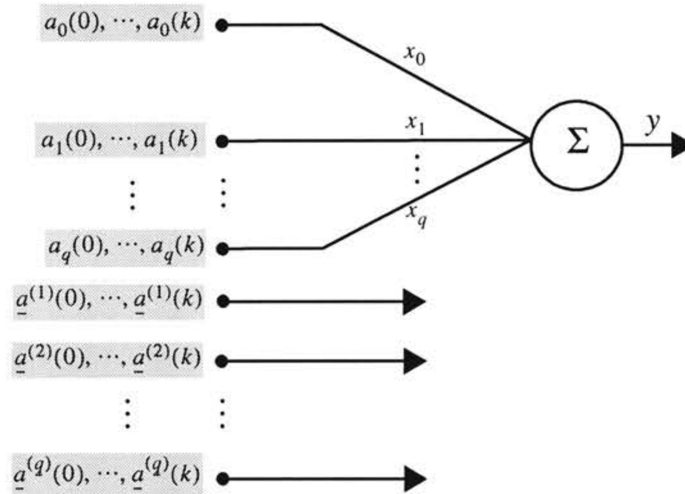


Figure 7 - 1 The Subset Selection Model

Figure 7 - 1 depicts the architecture of the subset selection model. As shown, the linear model is formed by using one part of the input data. Specifically, the output of the model is

$$y(k) = \sum_{i=0}^{q} a_i(k) x_i . \tag{7 - 1}$$

If we accumulate the time data from $0 \ldots k$, we can express the model as matrix and vectors

$$\underbrace{\begin{bmatrix} y(0) \\ y(1) \\ \\ y(k) \end{bmatrix}}_{\mathbf{y}_q(k)} = \underbrace{\begin{bmatrix} a_0(0) \; a_1(0) \; \ldots \; a_q(0) \\ a_0(1) \; a_1(1) \; \ldots \; a_q(1) \\ \\ a_0(k) \; a_1(k) \; \ldots \; a_q(k) \end{bmatrix}}_{\mathbf{A}_q(k)} \underbrace{\begin{bmatrix} x_1(k) \\ x_2(k) \\ \\ x_q(k) \end{bmatrix}}_{\mathbf{x}_q(k)} . \tag{7 - 2}$$

Note that the data arrives sequentially in time, and at each time point a new row is added to the selected input data matrix $\mathbf{A}_q(k)$. For example, at time point k, the following row is added to the input data matrix:

$$ {}_q\mathbf{a}(k) = \left[a_0(k)\ a_1(k)\ \dots\ a_q(k)\right]^T. \tag{7 - 3}$$

In addition, the columns of $\mathbf{A}_q(k)$ represent the various orders of the model. For example, when the order is increased to q, the following column is added to the selected input data matrix:

$$ \mathbf{a}_q(k) = \left[a_q(0)\ a_q(1)\ \dots\ a_q(k)\right]^T. \tag{7 - 4}$$

Meanwhile, let us assume that there are a set of $\underline{q}$ potential nodes for the RBF network that have not been selected. The data for these potential nodes are contained in the potential input data matrix:

$$ \underline{\mathbf{A}}_{\underline{q}}(k) = \begin{bmatrix} \underline{a}_1(0) & \underline{a}_2(0) & \dots & \underline{a}_{\underline{q}}(0) \\ \underline{a}_1(1) & \underline{a}_2(1) & \dots & \underline{a}_{\underline{q}}(1) \\ \vdots & \vdots & & \vdots \\ \underline{a}_1(k) & \underline{a}_2(k) & \dots & \underline{a}_{\underline{q}}(k) \end{bmatrix} \quad {}_{\underline{q}}\underline{\mathbf{a}}^T(k), \quad \underline{\mathbf{a}}_{\underline{q}}(k) \tag{7 - 5}$$

Each of the potential input data vectors are stored as a column. A superscript is used to denote the index of the potential data vector. For example,

$$ \underline{\mathbf{a}}_{\underline{q}}(k) = \left[\underline{a}_{\underline{q}}(0)\ \underline{a}_{\underline{q}}(1)\ \dots\ \underline{a}_{\underline{q}}(k)\right]^T \tag{7 - 6}$$

denotes the $\underline{q}$-th column of $\underline{\mathbf{A}}_{q}(k)$. This $\underline{\mathbf{A}}_{q}(k)$ matrix contains a set of potential data vectors ranging from $1 \ldots \underline{q}$. [Note that there is no bias contained in these potential input data vectors, so the index begins with 1.]

$$\underline{\mathbf{A}}_{q}(k) = \left[\underline{\mathbf{a}}_1(k)\ \underline{\mathbf{a}}_2(k)\ \ldots\ \underline{\mathbf{a}}_{q}(k)\right] \qquad (7 - 7)$$

As with the selected input data matrix $\mathbf{A}_q(k)$, the k-th time data in $\underline{\mathbf{A}}_{q}(k)$ corresponds to the last row of the potential input data matrix. A left subscript denotes the row of the matrix $\underline{\mathbf{A}}_{q}(k)$.

$${}_{\underline{q}}\underline{\mathbf{a}}(k) = \left[\underline{a}_1(k)\ \underline{a}_2(k)\ \ldots\ \underline{a}_{q}(k)\right]^T. \qquad (7 - 8)$$

Keep in mind that the main idea here is to have the algorithms select the best input data vector from a set of potential data vectors $\left[\underline{\mathbf{a}}_1(k)\ \underline{\mathbf{a}}_2(k)\ \ldots\ \underline{\mathbf{a}}_{q}(k)\right]$ and add it into the model to improve the RBF network performance. Simultaneously, the algorithm will utilize only the new time point and old computed parameters to update the model. In the following, we will look at the general idea of how we implement the time- and order- update together.

7.3 Recursive Time- and Order- Update

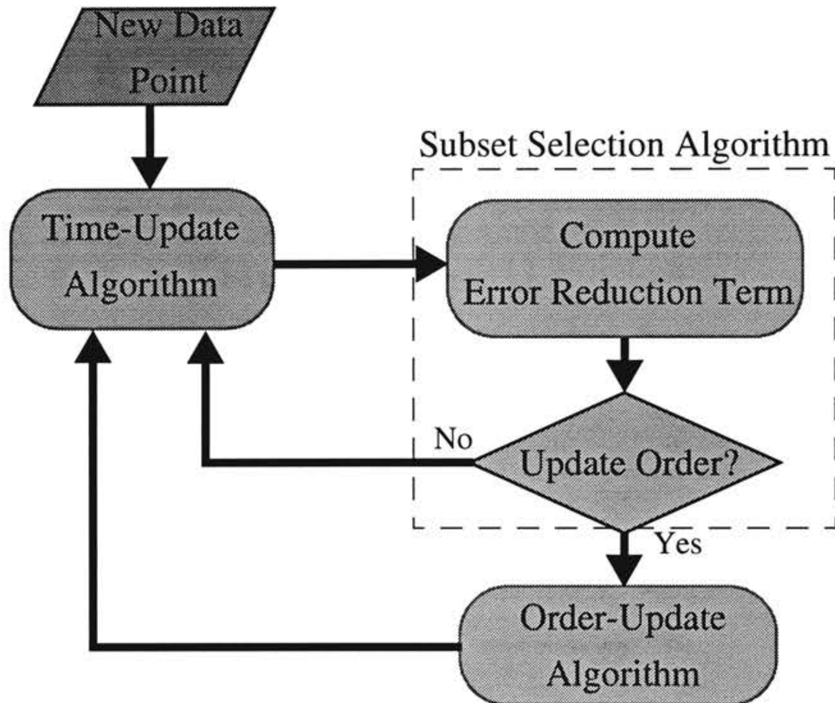


Figure 7 - 2 The Time- and Order- Update Algorithm Flow Chart

Figure 7 - 2 shows a general framework for the time- and order- update algorithm. This framework feeds a new time point into a time-update algorithm, then into the subset selection and order-update algorithms. The whole process is repeated at each time point. In the following, we will explain the operations contained in each block.

New Data Point

As each new data point is presented to the network, this produces a new row of the selected input data vector ${}_{q}\mathbf{a}(k)$, a new row of the potential input data vector ${}_{\underline{q}}\mathbf{a}(k)$, and a new desired response $d(k)$. These new data are presented to a time-update algorithm.

Time-Update Algorithm

Two time-update algorithms, the RLS algorithm and the QR-RLS algorithm discussed in Chapter 5, can be used to accommodate the time adjustment. In the RLS algorithm, the necessary parameters are the inverse correlation matrix $\overline{\mathbf{H}}_q(k)$, the weights and bias $\mathbf{x}_q(k)$, and the sum of squared errors $SSE_q(k)$. In the QR-RLS algorithm, the parameters are the R-factor $\mathbf{R}_q(k)$, the gain vector $\mathbf{g}_q(k)$, and the sum of squared errors $SSE_q(k)$. [Note that some intermediate steps to obtain these parameters are not shown here. Details are provided in section 7.3.1 and section 7.3.2.] After the time-update, the time-updated parameters are presented to the subset selection and order-update algorithms.

Subset Selection and Order-Update Algorithms

The role of the subset selection and order-update algorithms are to ensure the time-update algorithm provides adequate network performance. The subset selection algorithm first computes the error reduction terms for every potential input data vector, and determines if an order-update is necessary.

Subset Selection:

Recall from Chapter 6 that the error reduction term measures the error reduction caused by the added data vector (potential RBF node). Because we have a set of potential input data vectors, we need to repeat this calculation for every one of them $\underline{\mathbf{a}}_i(k)$ $i = 1 \ldots \underline{q}$.

Consider the error reduction equation, Eq. (6 - 32). With slight modification, we can rewrite it to compute the error reduction for every potential data vector $\underline{\mathbf{a}}_i(k)$ $i = 1 \ldots \underline{q}$ using the time- and order- indexing method:

$$err_q^{(i)}(k) = \rho_q^{(i)^{-2}}(k)(\underline{\mathbf{a}}_i^T(k)\mathbf{e}_q(k))^2 \quad i = 1 \ldots \underline{q} \tag{7 - 9}$$

where

$$\mathbf{e}_q(k) = \mathbf{d}(k) - \mathbf{A}_q(k)(\mathbf{A}_q^T(k)\mathbf{A}_q(k))^{-1}\mathbf{A}_q^T(k)\mathbf{d}(k), \text{ and} \tag{7 - 10}$$

$$\rho_q^{(i)^2}(k) = \underline{\mathbf{a}}_i^T(k)\underline{\mathbf{a}}_i(k) - \underline{\mathbf{a}}_i^T(k)\mathbf{A}_q(k)(\mathbf{A}_q^T(k)\mathbf{A}_q(k))^{-1}\mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k). \tag{7 - 11}$$

It is not hard to see that some of the terms in Eq. (7 - 10) and Eq. (7 - 11) are actually the time updated parameters which have already been calculated by the time-update algorithm. Using these facts, we can simply substitute the time updated parameters into the error reduction terms. Specifically, the error reduction terms can be computed as:

Error Reduction Calculation if RLS Algorithm is Used

$$err_q^{(i)}(k) = \rho_q^{(i)^{-2}}(k)(\underline{\mathbf{a}}_i^T(k)(\mathbf{d}(k) - \mathbf{A}_q(k)\mathbf{x}_q(k)))^2 \quad i = 1 \ldots \underline{q} \tag{7 - 12}$$

where

$$\rho_q^{(i)^2}(k) = \underline{\mathbf{a}}_i^T(k)\underline{\mathbf{a}}_i(k) - \underline{\mathbf{a}}_i^T(k)\mathbf{A}_q(k)\overline{\mathbf{H}}_q(k)\mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k) \tag{7 - 13}$$

Error Reduction Calculation if QR-RLS Algorithm is Used

$$err_q^{(i)}(k) = \rho_q^{(i)^{-2}}(k)(\underline{\mathbf{a}}_i^T(k)(\mathbf{d}(k) - \mathbf{A}_q(k)\mathbf{R}_q^{-1}(k)\mathbf{g}_q(k)))^2 \quad i = 1 \ldots \underline{q} \tag{7 - 14}$$

where

$$\rho_q^{(i)^2}(k) = \underline{\mathbf{a}}_i^T(k)\underline{\mathbf{a}}_i(k) - \underline{\mathbf{a}}_i^T(k)\mathbf{A}_q(k)\mathbf{R}_q^{-1}(k)\mathbf{R}_q^{-T}(k)\mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k) \tag{7 - 15}$$

The above method saves significant computation, as we do not perform the matrix inversion. However, because $\mathbf{A}_q(k)$, $\underline{\mathbf{A}}_{\underline{q}}(k)$ and $\mathbf{d}(k)$ appear in the calculation, we do have to store the data matrix, potential data matrix and the desired response vector.

$$\mathbf{A}_q(k) = \begin{bmatrix} \mathbf{A}_q(k-1) \\ {}_q\mathbf{a}(k) \end{bmatrix}, \underline{\mathbf{A}}_{\underline{q}}(k) = \begin{bmatrix} \underline{\mathbf{A}}_{\underline{q}}(k-1) \\ {}_{\underline{q}}\underline{\mathbf{a}}(k) \end{bmatrix}, \mathbf{d}(k) = \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix} \quad (7 - 16)$$

The time updated parameters, together with the data matrices described in Eq. (7 - 16), are used in calculating the error reduction terms. Once computed, the best error reduction term is picked (The best error reduction produces the largest number).

$$err_q(k) = max[err_q^{(i)}(k)] \quad (7 - 17)$$

Update Order:

To decide whether an order-update is needed, the sum of squared errors is calculated using the time updated sum of squared errors $SSE_q(k)$.

$$SSE_{q+1}(k) = SSE_q(k) - err_q(k) \quad (7 - 18)$$

The resulting sum of squared errors $SSE_{q+1}(k)$ is compared to a pre-selected threshold value γ. If $SSE_{q+1}(k) \geq \gamma$, then the network performance is inadequate, and we proceed to perform an order-update. Otherwise, no update is necessary.

Order-Update Algorithm:

If an order-update is necessary, we first need to identify which potential data vector produces the largest error reduction. Assuming that $\tilde{i}$ is the index that produces the largest

error reduction term, $\tilde{i} = index[max[err_q^{(i)}(k)]]$, then we need to move $\underline{\mathbf{a}}_{\tilde{i}}(k)$ from $\underline{\mathbf{A}}_q(k)$ to $\mathbf{A}_q(k)$ as $\underline{\mathbf{a}}_{\tilde{i}}(k)$ becomes part of the linear model. If the RLS algorithm is used, the order-update will update the inverse correlation matrix $\overline{\mathbf{H}}_{q+1}(k)$ using Eq. (6 - 38), the weights $\mathbf{x}_{q+1}(k)$ using Eq. (6 - 39), and the sum of squared errors $SSE_{q+1}(k)$ using Eq. (7 - 18). If the QR-RLS algorithm is used, the order-update will update the R-factor $\mathbf{R}_{q+1}(k)$, the gain vector $\mathbf{g}_{q+1}(k)$, and the sum of squared errors $SSE_{q+1}(k)$ using Eq. (7 - 18). Note that in Chapter 6, we rely on the orthogonal matrix $\mathbf{Q}_q(k)$ to obtain an accurate recursive QR order-update algorithm. We have not come up with update equations for $\mathbf{R}_{q+1}(k)$ and $\mathbf{g}_{q+1}(k)$. In the following, we will derive these update equations from Chapter 6.

To find an update for $\mathbf{R}_{q+1}(k)$, we use the fact that $\mathbf{Q}_q^T(k) = \mathbf{R}_q^{-T}(k)\mathbf{A}_q^T(k)$ from Eq. (6 - 41), and from Eq. (6 - 44) we can write

$$\mathbf{R}_{q+1}(k) = \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{R}_q^{-T}(k)\mathbf{A}_q^T(k)\underline{\mathbf{a}}_{\tilde{i}}(k) \\ \mathbf{0} & \rho_q(k) \end{bmatrix}. \tag{7 - 19}$$

Meanwhile, the gain vector $\mathbf{g}_q(k)$ can be derived from the weight update equation. Recall from Eq. (6 - 47), the weight update equation is

$$\mathbf{x}_{q+1}(k) = \begin{bmatrix} \mathbf{x}_q(k) \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{R}_q^{-1}(k)\mathbf{Q}_q^T(k)\underline{\mathbf{a}}_{\tilde{i}}(k)\rho_q^{-2}(k)\underline{\mathbf{a}}_{\tilde{i}}^{\ T}(k)\mathbf{e}_q(k) \\ \rho_q^{-2}(k)\underline{\mathbf{a}}_{\tilde{i}}^{\ T}(k)\mathbf{e}_q(k) \end{bmatrix}. \tag{7 - 20}$$

If we multiply $\mathbf{R}_{q+1}(k)$ in Eq. (7 - 19) times the weight update equation above, we obtain

$$\mathbf{R}_{q+1}(k)\mathbf{x}_{q+1}(k) = \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{R}_q^{-T}(k)\mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k) \\ \mathbf{0} & \rho_q(k) \end{bmatrix} \left(\begin{bmatrix} \mathbf{x}_q(k) \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{R}_q^{-1}(k)\mathbf{Q}_q^T(k)\underline{\mathbf{a}}_i(k)\rho_q^{-2}(k)\underline{\mathbf{a}}_i^T(k)\mathbf{e}_q(k) \\ \rho_q^{-2}(k)\underline{\mathbf{a}}_i^T(k)\mathbf{e}_q(k) \end{bmatrix} \right) \quad (7 - 21)$$

Make use of the fact that $\mathbf{Q}_q^T(k) = \mathbf{R}_q^{-T}(k)\mathbf{A}_q^T(k)$ from Eq. (6 - 41), $\mathbf{g}_q(k) = \mathbf{R}_q(k)\mathbf{x}_q(k)$ and $\mathbf{g}_{q+1}(k) = \mathbf{R}_{q+1}(k)\mathbf{x}_{q+1}(k)$, then we obtain the gain vector update equation

$$\mathbf{g}_{q+1}(k) = \begin{bmatrix} \mathbf{g}_q(k) \\ \rho_q^{-1}(k)(\underline{\mathbf{a}}_i^T(k)\mathbf{d}(k) - \underline{\mathbf{a}}_i^T(k)\mathbf{A}_q(k)\mathbf{R}_q^{-1}(k)\mathbf{g}_q(k)) \end{bmatrix}. \quad (7 - 22)$$

Once the necessary parameters are updated, the parameters are presented back to the time-update algorithm, as shown in Figure 7 - 2. Another cycle will start as soon as another data point is received.

Theoretically, (Golub & Van Loan 1996) updating the orthogonal matrix $\mathbf{Q}_q(k)$ yields numerical results that are more accurate. However, it requires storing the orthogonal matrix $\mathbf{Q}_q(k)$. This matrix $\mathbf{Q}_q(k)$ is of the size of the data matrix $\mathbf{A}_q(k)$ and it can be very large because it is time dependent. On-line implementation by storing the $\mathbf{Q}_q(k)$ matrix is possible but impractical. Meanwhile, updating $\mathbf{R}_q(k)$ and $\mathbf{g}_q(k)$ will be less numerically accurate (theoretically) but it does not require storage of the $\mathbf{Q}_q(k)$ matrix. Chen *et al.* (1991) have pointed out that the OLS algorithm can avoid numerical ill-conditioning, such as the near linear dependency caused by some RBF nodes being too close together. In light of this fact, we would expect the algorithm to work properly if the condition of the data

matrix $\mathbf{A}_q(k)$ is not too severe. However, it is important to know under what condition these algorithms will work properly. Hence, we will also conduct an empirical numerical study in Chapter 10.

In the next two sections, we will discuss the implementation of two time- and order-update algorithms.

7.3.1 Recursive Least Squares with Automatic Weight Selection (RLS-AWS)

In this section, we will discuss the implementation of recursive least squares with automatic weight selection (RLS-AWS). This algorithm combines the recursive least squares (RLS) algorithm in Chapter 5 and the order-update algorithm described in Eq. (7 - 12) and Eq. (7 - 13). This algorithm can begin with a bias as the only node and recursively add more nodes as time goes on; or begin with no parameter, in which case the order-update is used to select the first parameters. We can initialize the inverse correlation matrix to $\overline{\mathbf{H}}_0(0) = \delta^{-1}\mathbf{I}$ where δ is a small positive constant. Below is the implementation of the RLS-AWS algorithm.

RLS-AWS Algorithm

Initialization: $q = 0$, $\overline{\mathbf{H}}_0(0) = \delta^{-1}\mathbf{I}$, and $\mathbf{x}_0(0) = 0$

For $k = 1, 2, ...$

Read New Data: $\{{}_q\mathbf{a}(k), d(k), {}_{\underline{q}}\mathbf{a}(k)\}$

Time-Update: RLS Algorithm:

$$\kappa_q(k) = 1 + {}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)$$

$$\mathbf{k}_q(k) = \kappa_q^{-1}(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)$$

$$\xi_q(k) = d(k) - {}_q\mathbf{a}^T(k)\mathbf{x}_q(k-1)$$

$$\mathbf{x}_q(k) = \mathbf{x}_q(k-1) + \mathbf{k}_q(k)\xi_q(k)$$

$$\overline{\mathbf{H}}_q(k) = \overline{\mathbf{H}}_q(k-1) - \mathbf{k}_q(k)\,{}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)$$

$$SSE_q(k) = SSE_q(k-1) + \xi_q^2(k)\kappa_q^{-1}(k)$$

Store data matrix, potential data matrix and desired response

$$\mathbf{A}_q(k) = \begin{bmatrix}\mathbf{A}_q(k-1)\\ {}_q\mathbf{a}(k)\end{bmatrix},\ \underline{\mathbf{A}}_{\underline{q}}(k) = \begin{bmatrix}\underline{\mathbf{A}}_{\underline{q}}(k-1)\\ {}_{\underline{q}}\underline{\mathbf{a}}(k)\end{bmatrix},\ \mathbf{d}(k) = \begin{bmatrix}\mathbf{d}(k-1)\\ d(k)\end{bmatrix}$$

Parameters $\mathbf{x}_q(k)$, $\overline{\mathbf{H}}_q(k)$, $SSE_q(k)$, $\mathbf{A}_q(k)$, $\underline{\mathbf{A}}_{\underline{q}}(k)$, *and* $\mathbf{d}(k)$ *are passed to the order-update algorithm.*

Order-Update Algorithm:

Compute Error Reduction Term

For $1 \le i \le \underline{q}$ *compute*

$$\mathbf{u}_q^{(i)}(k) = \mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k)\ ,$$

$$\phi_q^{(i)}(k) = \underline{\mathbf{a}}_i^T(k)\mathbf{d}(k) - \mathbf{u}_q^{(i)T}(k)\mathbf{x}_q(k)\ ,$$

$$\mathbf{z}_q^{(i)}(k) = \overline{\mathbf{H}}_q(k)\mathbf{u}_q^{(i)}(k),$$

$$\rho_q^{(i)^2}(k) = \underline{\mathbf{a}}_i^T(k)\underline{\mathbf{a}}_i(k) - \mathbf{u}_q^{(i)^T}(k)\mathbf{z}_q^{(i)}(k)$$

$$err_q^{(i)}(k) = \rho_q^{(i)^{-2}}(k)\phi_q^{(i)^2}(k)$$

Update Order?

Pick $err_q(k) = max[err_q^{(i)}(k)]$ *for* $1 \le i \le \underline{q}$

$$SSE_{q+1}(k) = SSE_q(k) - err_q(k)$$

If $SSE_{q+1}(k) \ge \gamma$, *update parameters. Otherwise, no update necessary*

Update Parameters

Keep $SSE_{q+1}(k)$. *Let* $\tilde{i} = index[max[err_q^{(i)}(k)]]$, *then move* $\underline{\mathbf{a}}_{\tilde{i}}(k)$ *from*

$\underline{\mathbf{A}}_{\underline{q}}(k)$ *to* $\mathbf{A}_q(k)$ *which produces* $\mathbf{A}_{q+1}(k)$ *and* $\underline{\mathbf{A}}_{\underline{q}-1}(k)$. *Then update*

$$\rho_q^{-2}(k) = \rho_q^{(\tilde{i})^2}(k),\ \phi_q(k) = \phi_q^{(\tilde{i})}(k),\ \mathbf{z}_q(k) = \mathbf{z}_q^{(\tilde{i})}(k),$$

$$\overline{\mathbf{H}}_{q+1}(k) = \left[\begin{array}{c|c} \overline{\mathbf{H}}_q(k) + \mathbf{z}_q(k)\rho_q^{-2}(k)\mathbf{z}_q^T(k) & -\mathbf{z}_q(k)\rho_q^{-2}(k) \\ \hdashline -\rho_q^{-2}(k)\mathbf{z}_q^T(k) & \rho_q^{-2}(k) \end{array}\right]$$

$$\mathbf{x}_{q+1}(k) = \begin{bmatrix} \mathbf{x}_q(k) \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{z}_q(k)\rho_q^{-2}(k)\phi_q(k) \\ \rho_q^{-2}(k)\phi_q(k) \end{bmatrix}$$

7.3.2 QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS)

The QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS) algorithm combines the QR recursive least squares (QR-RLS) algorithm in Chapter 5 and the order-update algorithm as described in Eq. (7 - 19) and Eq. (7 - 22). Similar to the RLS algorithm, this algorithm can begin with a bias as the only parameter or can begin with no parameter (order-update is used to select parameter), and recursively adds more parameters as time goes on. The initialization procedure is the same as the QR-RLS algorithm, where we initialize $\mathbf{R}_0(0) = 0$. Below is the implementation of the QR-RLS-AWS algorithm.

QR-RLS-AWS Algorithm

Initialization: $q = 0,\ \mathbf{R}_q(0) = \mathbf{0},\ and\ \mathbf{g}_q(0) = \mathbf{0}$

For $k = 1, 2, ...$

Read New Data: $\{{}_q\mathbf{a}(k), d(k), {}_{\underline{q-}}\mathbf{a}(k)\}$

Time-Update: QR-RLS Algorithm

Solve using Givens Rotation

$$\mathbf{Q}_1^T(k)\begin{bmatrix}\mathbf{R}_q(k-1) & \mathbf{g}_q(k-1)\\ {}_q\mathbf{a}^T(k) & d(k)\end{bmatrix} = \begin{bmatrix}\mathbf{R}_q(k) & \mathbf{g}_q(k)\\ \mathbf{0}^T & \kappa_q^{-1/2}(k)\xi_q(k)\end{bmatrix}$$

$$SSE_q(k) = SSE_q(k-1) + \kappa^{-1}(k)\xi^2(k)$$

Store data matrix, potential data matrix and desired response

$$\mathbf{A}_q(k) = \begin{bmatrix} \mathbf{A}_q(k-1) \\ {}_q\mathbf{a}(k) \end{bmatrix}, \underline{\mathbf{A}}_{\underline{q}}(k) = \begin{bmatrix} \underline{\mathbf{A}}_{\underline{q}}(k-1) \\ {}_{\underline{q}}\underline{\mathbf{a}}(k) \end{bmatrix}, \mathbf{d}(k) = \begin{bmatrix} \mathbf{d}(k-1) \\ d(k) \end{bmatrix}$$

Parameters $\mathbf{g}_q(k)$, $\mathbf{R}_q(k)$, $SSE_q(k)$, $\mathbf{A}_q(k)$, $\underline{\mathbf{A}}_{\underline{q}}(k)$, *and* $\mathbf{d}(k)$ *are passed to the order-update algorithm.*

Order-Update:

Compute Error Reduction Term

For $1 \le i \le \underline{q}$

$$\mathbf{u}_q^{(i)}(k) = \mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k) ,$$

$$\mathbf{r}_q^{(i)}(k) = \mathbf{R}_q^{-T}(k)\mathbf{u}_q^{(i)}(k), \textit{ Solve using back-substitution}$$

$$\phi_q^{(i)}(k) = \underline{\mathbf{a}}_i^T(k)\mathbf{d}(k) - \mathbf{r}_q^{(i)^T}(k)\mathbf{g}_q(k)$$

$$\rho_q^{(i)^2}(k) = \underline{\mathbf{a}}_i^T(k)\underline{\mathbf{a}}_i(k) - \mathbf{r}_q^{(i)^T}(k)\mathbf{r}_q^{(i)}(k)$$

$$err_q^{(i)}(k) = \rho_q^{(i)^{-2}}(k)\phi_q^{(i)^2}(k)$$

Update Order?

Pick $err_q(k) = max[err_q^{(i)}(k)]$ *for* $1 \le i \le \underline{q}$

$$SSE_{q+1}(k) = SSE_q(k) - err_q(k)$$

If $SSE_{q+1}(k) \ge \gamma$, *update parameters. Otherwise, no update necessary.*

Update Parameters:

Keep $SSE_{q+1}(k)$. *Let* $\tilde{i} = index[max[err_q^{(i)}(k)]]$, *then move* $\underline{\mathbf{a}}_{\tilde{i}}(k)$ *from* $\underline{\mathbf{A}}_q(k)$

to $\mathbf{A}_q(k)$ *which produces* $\mathbf{A}_{q+1}(k)$ *and* $\underline{\mathbf{A}}_{q-1}(k)$. *Also, update*

$$\rho_q^{-2}(k) = \rho_q^{(\tilde{i})^2}(k),\ \phi_q(k) = \phi_q^{(\tilde{i})}(k),\ \mathbf{r}_q(k) = \mathbf{r}_q^{(\tilde{i})}(k)$$

For parameter updates, implement Eq. (7 - 19) and Eq. (7 - 22)

$$\mathbf{R}_{q+1}(k) = \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{r}_q(k) \\ \mathbf{0} & \rho_q(k) \end{bmatrix}$$

$$\mathbf{g}_{q+1}(k) = \begin{bmatrix} \mathbf{g}_q(k) \\ \rho_q^{-1}(k)\phi_q(k) \end{bmatrix}$$

7.4 Fixing Centers

To implement the two algorithms in the RBF network, we need to define how we fix the parameters of the RBF hidden layer. Specifically, how do we select the RBF fixed centers and fixed standard deviation. In general, there are two techniques.

7.4.1 Centers Selects from Fixed Range/Grid

This technique equally spaces the centers in the input space. If the RBF network has two inputs, then the centers are placed according to a grid (Fabri & Kadirkamanathan 1996, Sanner 1993, Sanner & Soltine 1992). By placing the centers in a grid fashion over the input space, the gradient descent method can be used to train this network in real-time. However, there are two drawbacks to this method. First, due to substantial centers placed over the

input spaces, the network size is very large. Therefore, this type of network suffers from the curse of dimensionality - the number of hidden nodes increase exponentially with respect to the dimension of the input spaces. Second, the gradient descent method has a slow rate of convergence. Algorithms with fast rate of convergence utilize second order information, such as the RLS and QR-RLS, are impractical due to the large number of weights. Despite these drawbacks, this method is good for low dimensional inputs, in control applications.

7.4.2 Centers Selected from Time Point

Centers selected from time points is a technique used in the original RBF network (Powell 1987a). In the original paper, the RBF centers are chosen from every time data point, which usually produces a very large set. Lowe (1989) considers a subset of centers randomly selected from the training data sets. This approach is considered to be "sensible" by the author, if the training data are distributed in a representative manner. However, arbitrarily selected centers are clearly unsatisfactory. Chen et al. (1991) developed an alternative procedure based on the orthogonal least squares method. The procedure first assumes that a set of potential RBF centers are chosen from every time point. Then, the one by one in a rational way, the RBF centers are chosen until an adequate network has been constructed.

7.4.3 Centers Selection for the New Algorithms

Because the two new algorithms are an extension of the orthogonal least squares method, we will consider center selection from the time point data. In the following we will

assume that as a new time point becomes available, a potential center is placed on that time point.

Hence, there are as many potential centers as the number of time points, and it is up to the algorithm to pick the best centers for the network. In the following we will give some preliminary results based on this methodology.

7.5 Preliminary Results

In this section, we will use a single-input/single-output function to test the function approximation capabilities of our new algorithms. This function to be approximated is

$$y(k) = \sin(k) + \cos(2k) \text{ with sampling interval } = \pi/20 \tag{7 - 23}$$

The standard deviation and the threshold criteria is selected as

$$\sigma = 1 \text{ and } \gamma = 10^{-3}. \tag{7 - 24}$$

Meanwhile, the RBF network begins with no parameter; thus, subset selection is used to select the centers at time point $k = 0$. As mentioned previously, when the current data is presented, an RBF potential center based on that data point is created as well. In the following, all the figures have two subplots. In the top plot, + (plus marks) are all the potential centers, o (circle marks) are the selected centers and - (solid lines) are the RBF network outputs. The bottom plot shows the sum of squared errors.

7.5.1 Compare QR-RLS-AWS and RLS-AWS

Our first experiment is to compare the QR-RLS-AWS algorithm and the RLS-AWS algorithm. Figure 7 - 3 shows the result of both algorithms. Only one figure is shown because both algorithms yield identical RBF network output. Even the sum of squared errors is the same. This comes as no surprise since both algorithms are derived from the least squares method. The reason why we developed the QR-RLS-AWS algorithm is because of its numerical accuracy.

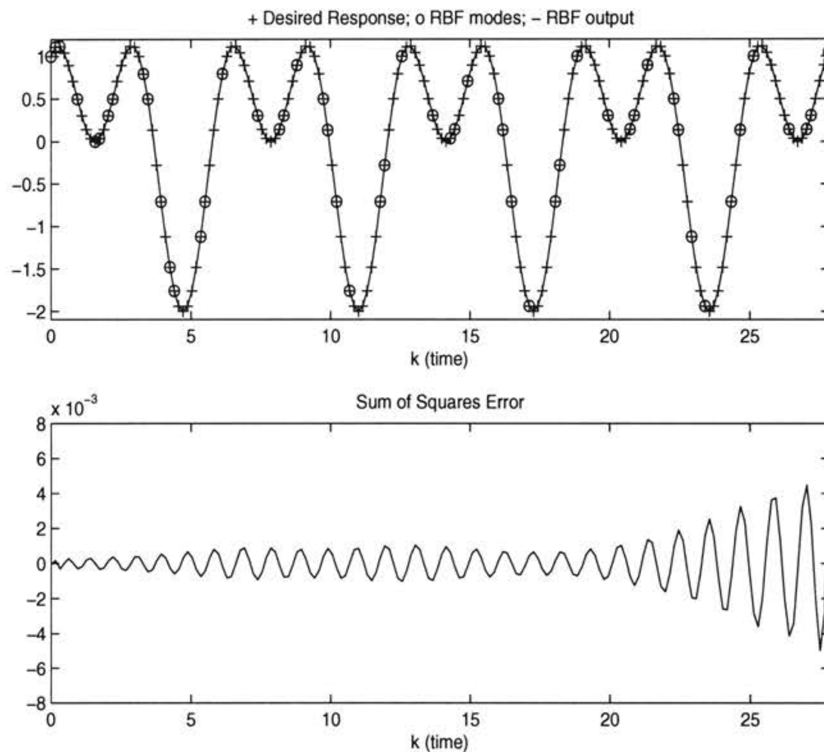


Figure 7 - 3 The QR-RLS-AWS and RLS-AWS algorithms Result

7.5.2 Accuracy Test

To test if the QR-RLS-AWS is numerically more accurate, we reduce the threshold value. In fact, a threshold value of $\gamma = 10^{-4}$ is enough to show the numerical instability of the RLS-AWS. Figure 7 - 4 shows this phenomena. As shown, erratic RBF network output behavior emerges after the some time passes.

Using the same threshold value, we tested the QR-RLS-AWS algorithm. As shown in Figure 7 - 5, it remains stable.

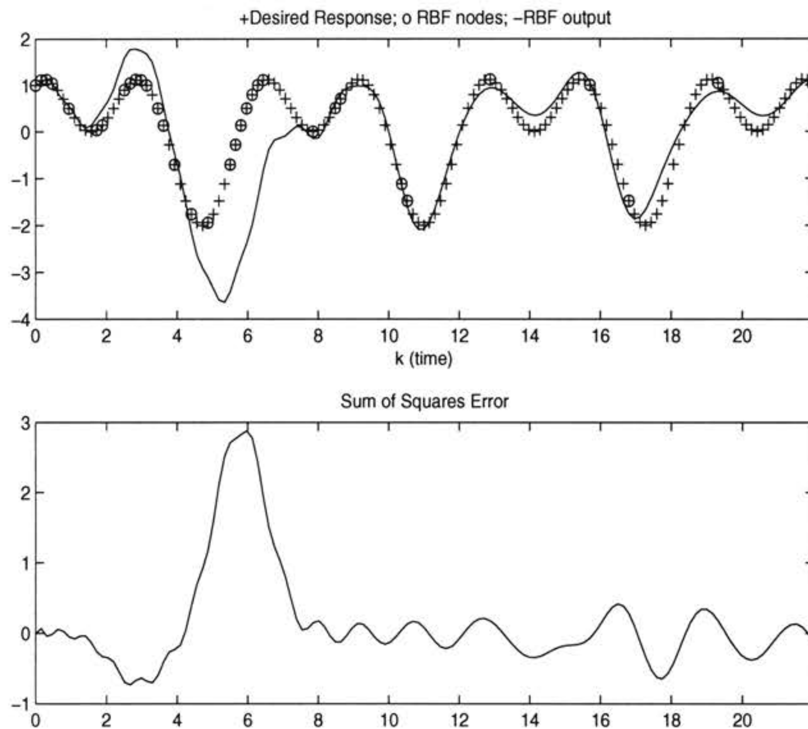


Figure 7 - 4 RLS-AWS Algorithm Instability

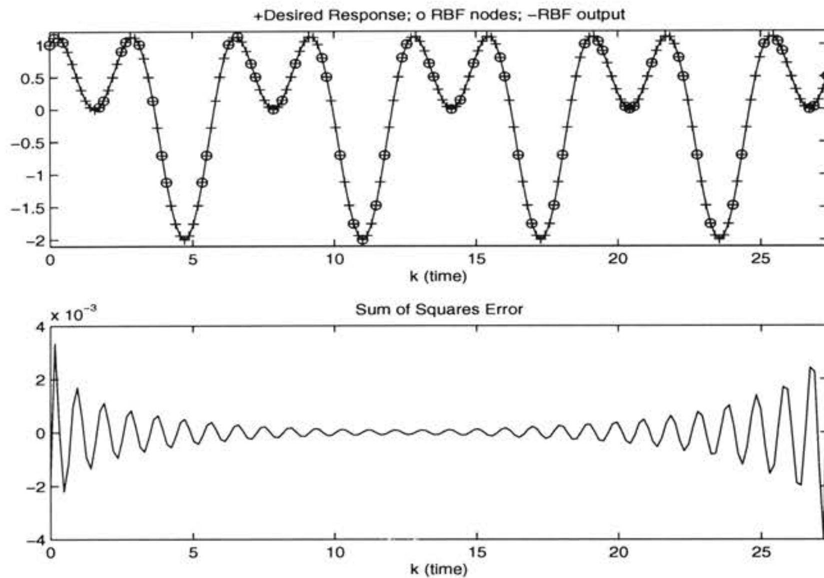


Figure 7 - 5 QR-RLS-AWS Algorithm No Blow Out

7.5.3 Batch and Recursive Test

Since the two new algorithms are extensions of the batch orthogonal least squares method (Chen et al. 1991), a test is conducted to compare them. The batch result is shown in Figure 7 - 6. If we compare the number of centers selected by the batch OLS method to the new algorithms (Figure 7 - 3), we can see that the batch OLS method selects fewer centers, 44 compared to 53, out of 178 possible centers. This result is not surprising if we consider the fact that the algorithm is recursive in nature, and the future time points are not available to the new algorithm. Another difficulty arises when a center selected at an early stage becomes unimportant in a later stage. This algorithm is not designed to take out insignificant centers. Hence, more centers are selected using the new algorithms.

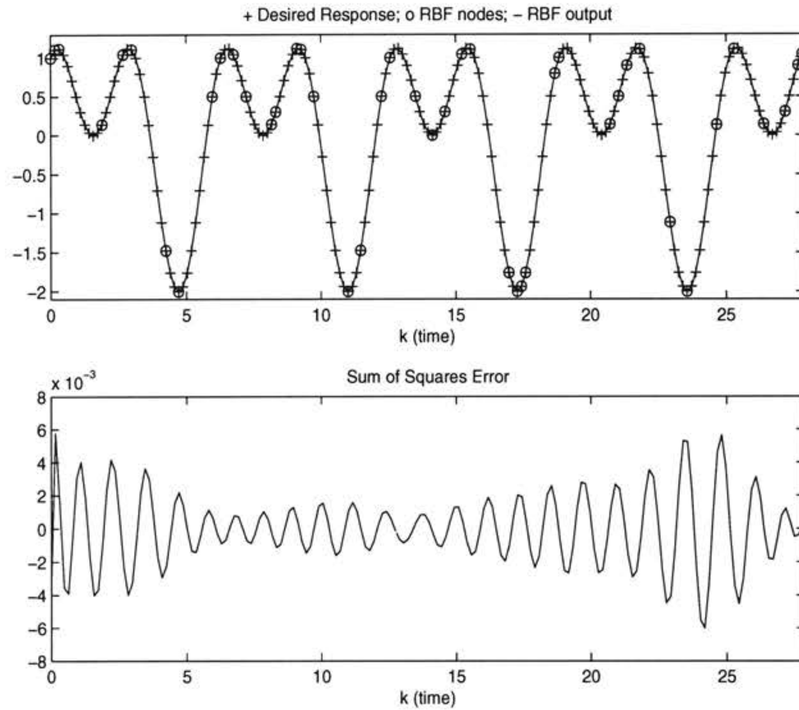


Figure 7 - 6 Batch OLS algorithm

7.6 Summary

Keep in mind that these preliminary results are tested on a simple 1 dimensional function. Further tests are needed to verify the performance capability of these algorithms. Also, there is still room for improvement in algorithm design and implementation. We will explore these improvement in the next chapters.

Chapter 8

RLS-AWS Algorithm Improvement

8.1 Introduction 135
8.2 Alleviate the Storage Requirement 136
8.2.1 Time-Update Correlation Matrix 136
8.2.2 Improvement to Forward Selection Method 139
8.2.3 Restructuring Time-Update Correlation Matrix 141
8.3 Order-Decrease-Update Algorithms 142
8.3.1 Block Matrix Inversion Lemma for Matrix Downdate 144
8.3.2 Recursive Order-Decrease-Update Algorithm for LS Method 147
8.4 Recursive Backward Elimination 154
8.5 Recursive Efroymson Algorithm 155
8.5.1 Batch Efroymson Algorithm 155
8.5.2 Recursive Efroymson Algorithm 157
8.5.3 RLS-AWS Algorithm: Efroymson Method 161
8.6 Implementation Consideration 167
8.6.1 Exponential Windowing 167
8.6.2 Reduce Computational Time 169
8.7 Summary 170

In this chapter, we devote our efforts to improve one of the recursive time- and order-update algorithms proposed in chapter 7: the RLS-AWS algorithm. These improvements include alleviating the storage requirement, improving the algorithm's subset selection solution, and reducing the computation. Subsequently, we make this algorithm practical for real-time usage.

8.1 Introduction

In this chapter, we will discuss several ways to improve the performance of the Recursive Least Squares with Automatic Weights Selection (RLS-AWS) algorithm.

We first tackle the storage improvement of the algorithm by devising a way to store time independent terms, which are needed for the RLS-AWS calculation, instead of the time dependent terms such as the data matrix and the potential data matrix. These time independent terms form a matrix, which we call the time-update correlation matrix. With this modification, we improve the storage requirement of RLS-AWS algorithm to a fixed size. We will also explain how this modification can be applied to the RLS-AWS algorithm with the recursive forward selection method and the recursive Efroymson method.

Second, we improve the algorithm's subset selection solution by developing a recursive Efroymson method for the RLS-AWS algorithm. In chapter 7, the RLS-AWS algorithm works by adding the best potential nodes into the network to ensure adequate network performance. In addition to adding the best potential nodes, the recursive Efroymson method also deletes under-performing nodes from the network to ensure a smaller network. In chapter 7, we explained how we can add the best potential nodes when the order of the model is increased (a new radial basis function node is added). In this chapter, we will explain how we can remove the least important nodes from the network when the order of the model is decreased (a new radial basis function node is removed). Because both procedures are order-update methods, we will call the order-update in chapter 6 and chapter 7 as order-increase-update and order-update in this chapter as order-decrease-

update. Later in the section, we will combine the order-decrease-update and the order-increase-update to form the recursive Efroymson method for the RLS-AWS algorithm.

Lastly, we will discuss how we can reduce computation by utilizing the localized character of the RBF network. In addition, we will incorporate an exponential window to the new algorithms.

8.2 Alleviate the Storage Requirement

One of the major limitations of the RLS-AWS algorithm described in section 7.3.1 is the requirement to store the data matrix $\mathbf{A}_q(k)$ and the potential data matrix $\underline{\mathbf{A}}_q(k)$. Because these data matrices accumulate data at every time step, the sizes of these data matrices become very large rapidly over time. Hence, it is impractical to implement this algorithm in a real-time system if we are required to store these data matrices.

8.2.1 Time-Update Correlation Matrix

In this section, a modification is made to the implementation of this RLS-AWS algorithm. With this modification, we reduce the storage requirement to a fixed size and completely eliminate the dependency on time making real-time implementation possible.

The basic idea of this modification is instead of storing the data matrix, we store matrices and vectors that are not time dependent but that are sufficient for the error reduction calculation. These matrices and vectors can be combined to form one big matrix

and are updated by the newly arrived time data vector ${}_{q}\mathbf{a}(k)$, the newly arrived potential time data vector ${}_{q-}\mathbf{a}(k)$ and the newly arrived desired response $d(k)$. In the following, we will show how we form this time-update correlation matrix and how we use it in the RLS-AWS algorithm.

The correlation of the data matrix is defined as

$$\mathbf{A}_q^T(k)\mathbf{A}_q(k). \tag{8 - 1}$$

The data matrix in Eq. (8 - 1) can be rewritten as

$$\mathbf{A}_q(k) = \begin{bmatrix} \mathbf{A}_q(k-1) \\ {}_{q}\mathbf{a}(k) \end{bmatrix}, \tag{8 - 2}$$

where $\mathbf{A}_q(k-1)$ is the data matrix of previous time step and ${}_{q}\mathbf{a}(k)$ is the current time data vector. If we substitute Eq. (8 - 2) into Eq. (8 - 1), we obtain a recursive equation for the correlation matrix

$$\mathbf{A}_q^T(k)\mathbf{A}_q(k) = \mathbf{A}_q^T(k-1)\mathbf{A}_q(k-1) + {}_{q}\mathbf{a}(k){}_{q}\mathbf{a}^T(k). \tag{8 - 3}$$

Let the previous time step correlation matrix be $\mathbf{H}_q(k-1) = \mathbf{A}_q^T(k-1)\mathbf{A}_q(k-1)$, then the current time step correlation matrix is

$$\mathbf{H}_q(k) = \mathbf{H}_q(k-1) + {}_{q}\mathbf{a}(k){}_{q}\mathbf{a}^T(k). \tag{8 - 4}$$

Eq. (8 - 4) implies that if we stored the previous time step correlation matrix $\mathbf{H}_q(k-1)$, we can obtain the updated correlation matrix using the current time data vector ${}_{q}\mathbf{a}(k)$. The

important thing here to remember is that the size of the correlation matrix $\mathbf{H}_q(k)$ is $q \times q$ (the storage size is fixed), and its size is not varying with time.

Using the same technique, we can derive recursive equations for $\mathbf{A}_q^T(k)\mathbf{d}(k)$, $\underline{\mathbf{A}}_q^T(k)\underline{\mathbf{A}}_q(k)$, $\mathbf{A}_q^T(k)\underline{\mathbf{A}}_q(k)$, $\underline{\mathbf{A}}_q^T(k)\mathbf{A}_q(k)$ and $\underline{\mathbf{A}}_q^T(k)\mathbf{d}(k)$. In fact, we can build these recursive equations into one matrix as follows

$$\begin{bmatrix} \mathbf{A}_q^T(k) \\ \underline{\mathbf{A}}_q^T(k) \end{bmatrix} \begin{bmatrix} \mathbf{A}_q(k) & \underline{\mathbf{A}}_q(k) & \mathbf{d}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_q^T(k-1) \\ \underline{\mathbf{A}}_q^T(k-1) \end{bmatrix} \begin{bmatrix} \mathbf{A}_q(k-1) & \underline{\mathbf{A}}_q(k-1) & \mathbf{d}(k-1) \end{bmatrix} + \begin{bmatrix} {}_q\mathbf{a}(k) \\ {}_{\underline{q}}\mathbf{a}(k) \end{bmatrix} \begin{bmatrix} {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\mathbf{a}^T(k) & d(k) \end{bmatrix} \quad (8-5)$$

$$\begin{bmatrix} \mathbf{A}_q^T(k)\mathbf{A}_q(k) & \mathbf{A}_q^T(k)\underline{\mathbf{A}}_q(k) & \mathbf{A}_q^T(k)\mathbf{d}(k) \\ \underline{\mathbf{A}}_q^T(k)\mathbf{A}_q(k) & \underline{\mathbf{A}}_q^T(k)\underline{\mathbf{A}}_q(k) & \underline{\mathbf{A}}_q^T(k)\mathbf{d}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_q^T(k-1)\mathbf{A}_q(k-1) & \mathbf{A}_q^T(k-1)\underline{\mathbf{A}}_q(k-1) & \mathbf{A}_q^T(k-1)\mathbf{d}(k-1) \\ \underline{\mathbf{A}}_q^T(k-1)\mathbf{A}_q(k-1) & \underline{\mathbf{A}}_q^T(k-1)\underline{\mathbf{A}}_q(k-1) & \underline{\mathbf{A}}_q^T(k-1)\mathbf{d}(k-1) \end{bmatrix} + \begin{bmatrix} {}_q\mathbf{a}(k) \\ {}_{\underline{q}}\mathbf{a}(k) \end{bmatrix} \begin{bmatrix} {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\mathbf{a}^T(k) & d(k) \end{bmatrix} . \quad (8-6)$$

Let $\mathbf{H}_q(k) = \mathbf{A}_q^T(k)\mathbf{A}_q(k)$, $\underline{\mathbf{H}}_q(k) = \underline{\mathbf{A}}_q^T(k)\underline{\mathbf{A}}_q(k)$, $\mathbf{U}_q^{(q)}(k) = \mathbf{A}_q^T(k)\underline{\mathbf{A}}_q(k)$, [Note that $\mathbf{U}_q^{(q)T}(k) = \underline{\mathbf{A}}_q^T(k)\mathbf{A}_q(k)$], $\underline{\mathbf{v}}_q(k) = \underline{\mathbf{A}}_q^T(k)\mathbf{d}(k)$, and $\mathbf{v}_q(k) = \mathbf{A}_q^T(k)\mathbf{d}(k)$ then

$$\begin{bmatrix} \mathbf{H}_q(k) & \mathbf{U}_q^{(\underline{q})}(k) & \mathbf{v}_q(k) \\ \mathbf{U}_q^{(\underline{q})^T}(k) & \underline{\mathbf{H}}_{\underline{q}}(k) & \underline{\mathbf{v}}_{\underline{q}}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{H}_q(k-1) & \mathbf{U}_q^{(\underline{q})}(k-1) & \mathbf{v}_q(k-1) \\ \mathbf{U}_q^{(\underline{q})^T}(k-1) & \underline{\mathbf{H}}_{\underline{q}}(k-1) & \underline{\mathbf{v}}_{\underline{q}}(k-1) \end{bmatrix} + \begin{bmatrix} {}_q\mathbf{a}(k) \\ {}_{\underline{q}}\underline{\mathbf{a}}(k) \end{bmatrix} \begin{bmatrix} {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & d(k) \end{bmatrix} \tag{8 - 7}$$

Eq. (8 - 7) is called the time-update correlation matrix. Its function is to perform a time-update calculation so that $\mathbf{U}_q^{(\underline{q})}(k)$, $\mathbf{H}_q(k)$, $\underline{\mathbf{H}}_{\underline{q}}(k)$, $\mathbf{v}_q(k)$, and $\underline{\mathbf{v}}_{\underline{q}}(k)$ are updated. The size of this matrix is $(q+\underline{q}) \times (q+\underline{q}+1)$; however, because it is symmetric, we only need to store the upper triangular elements of $\mathbf{H}_q(k)$, and $\underline{\mathbf{H}}_{\underline{q}}(k)$. ($\mathbf{U}_q^{(\underline{q})}(k)$, $\mathbf{v}_q(k)$ and $\underline{\mathbf{v}}_{\underline{q}}(k)$ are stored in full). Total storage reduces to $(q+\underline{q})^2/2 + 3(q+\underline{q})/2$. In addition, the update in Eq. (8 - 7) requires ~ $O((q+\underline{q})^2)$ flops.

8.2.2 Improvement of the Forward Selection Method

Eq. (8 - 7) is the key to the storage savings as the updated matrix contains all the necessary terms to compute the error reduction equations. In the following, we repeat the error reduction equations from Chapter 7, and show how these terms in error reduction are associated with the terms in Eq. (8 - 7).

$$err_{q+1}^{(i)}(k) = \rho_{q+1}^{(i)}{}^{-2}(k)\phi_{q+1}^{(i)}{}^{2}(k) \quad i = 1 \ldots \underline{q} \tag{8 - 8}$$

where

$$\phi_{q+1}^{(i)} = \underbrace{\underline{\mathbf{a}}_i^T(k)\mathbf{d}(k)}_{\underline{v}_q^{(i)}(k)} - \underbrace{\underline{\mathbf{a}}_i^T(k)\mathbf{A}_q(k)}_{\mathbf{u}_q^{(i)T}(k)} \underbrace{(\mathbf{A}_q^T(k)\mathbf{A}_q(k))^{-1}}_{\overline{\mathbf{H}}_q(k)} \underbrace{\mathbf{A}_q^T(k)\mathbf{d}(k)}_{\mathbf{v}_q(k)}, \text{ and} \quad (8 - 9)$$

$$\rho_{q+1}^{(i)\,2}(k) = \underbrace{\underline{\mathbf{a}}_i^T(k)\underline{\mathbf{a}}_i(k)}_{\underline{h}_q^{(i,i)}(k)} - \underbrace{\underline{\mathbf{a}}_i^T(k)\mathbf{A}_q(k)}_{\mathbf{u}_q^{(i)T}(k)} \underbrace{(\mathbf{A}_q^T(k)\mathbf{A}_q(k))^{-1}}_{\overline{\mathbf{H}}_q(k)} \underbrace{\mathbf{A}_q^T(k)\underline{\mathbf{a}}_i(k)}_{\mathbf{u}_q^{(i)}(k)}. \quad (8 - 10)$$

Specifically, $\underline{v}_q^{(i)}(k)$ is the ith element of the $\underline{\mathbf{v}}_q(k)$ vector, $\underline{h}_q^{(i,i)}(k)$ is the diagonal element associated with the ith row and the ith column of the $\underline{\mathbf{H}}_q(k)$ matrix, and $\mathbf{u}_q^{(i)}(k)$ is the ith column vector of the $\mathbf{U}_q^{(q)}(k)$ matrix. Together with the inverse Hessian matrix $\mathbf{H}_q(k)$ update of Eq. (6 - 13), we have all the necessary terms required for the error reduction calculation. Keep in mind since $\mathbf{x}_q(k) = \mathbf{H}_q(k)\mathbf{v}_q(k)$, and $\mathbf{x}_q(k)$ is readily available from the time-update (RLS) algorithm, we can use it directly in this computation. In term of floating point operation, the forward selection requires ~ $O(2\underline{q}q^2)$.

Take note that Eq. (8 - 7) not only contains all the necessary terms for the order-increase-update algorithm, but it also contains all the necessary terms for the order-decrease-update. Specifically, $\begin{bmatrix} \mathbf{U}_q^{(q)}(k) \\ \underline{\mathbf{H}}_q(k) \end{bmatrix}$ is needed for order-increase-update and $\begin{bmatrix} \mathbf{H}_q(k) \\ \mathbf{U}_q^{(q)T}(k) \end{bmatrix}$ is needed for order-decrease-update. Meanwhile, $\begin{bmatrix} \mathbf{v}_q(k) \\ \underline{\mathbf{v}}_q(k) \end{bmatrix}$ is essential for both

order-update methods. However, if only the order-increase-update is necessary, we will only need

$$\begin{bmatrix} \mathbf{U}_q^{(\underline{q})}(k) & \mathbf{v}_q(k) \\ \underline{\mathbf{H}}_{\underline{q}}(k) & \underline{\mathbf{v}}_{\underline{q}}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{U}_q^{(\underline{q})}(k-1) & \mathbf{v}_q(k-1) \\ \underline{\mathbf{H}}_{\underline{q}}(k-1) & \underline{\mathbf{v}}_{\underline{q}}(k-1) \end{bmatrix} + \begin{bmatrix} {}_q\mathbf{a}(k) \\ {}_{\underline{q}}\mathbf{a}(k) \end{bmatrix} \begin{bmatrix} {}_{\underline{q}}\mathbf{a}^T(k) & d(k) \end{bmatrix}. \qquad (8-11)$$

8.2.3 Restructuring Time-Update Correlation Matrix

It is important to note that once we determine that an order-increase-update is necessary, we will need to move the selected potential node into the model. This change directly affects the time-update correlation matrix. In particular, we will need to restructure those terms associated with the selected potential node in the time-update correlation matrix. In the following, we will show how we restructure the time-update correlation matrix Eq. (8 - 11) when we only need an order-increase-update.

Let the $\underline{q}^{th}$ node be the selected potential node for the order-increase-update. Before the restructuring, we have

$$\mathbf{U}_q^{(\underline{q})}(k) = \begin{bmatrix} \mathbf{u}_q^{(1)}(k) & \dots & \mathbf{u}_q^{(\underline{q}-1)}(k) & \mathbf{u}_q^{(\underline{q})}(k) \end{bmatrix}, \; \underline{\mathbf{v}}_{\underline{q}}(k) = \begin{bmatrix} \underline{v}_1(k) & \underline{v}_2(k) & \dots & \underline{v}_{\underline{q}}(k) \end{bmatrix}^T, \qquad (8-12)$$

$$\underline{\mathbf{H}}_{\underline{q}}(k) = \begin{bmatrix} \underline{h}_{\underline{q}}^{(1,1)}(k) & \underline{h}_{\underline{q}}^{(1,2)}(k) & \dots & \underline{h}_{\underline{q}}^{(1,\underline{q})}(k) \\ \underline{h}_{\underline{q}}^{(2,1)}(k) & \underline{h}_{\underline{q}}^{(2,2)}(k) & \dots & \underline{h}_{\underline{q}}^{(2,\underline{q})}(k) \\ & & & \\ \underline{h}_{\underline{q}}^{(\underline{q},1)}(k) & \underline{h}_{\underline{q}}^{(\underline{q},2)}(k) & \dots & \underline{h}_{\underline{q}}^{(\underline{q},\underline{q})}(k) \end{bmatrix}, \qquad (8-13)$$

and

$$\begin{bmatrix} \mathbf{U}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \underline{\mathbf{H}}_q(k) & \underline{\mathbf{v}}_q(k) \end{bmatrix} = \begin{bmatrix} \mathbf{u}_q^{(1)}(k) & \dots & \mathbf{u}_q^{(q-1)}(k) & \mathbf{u}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \underline{h}_q^{(1,1)}(k) & \dots & \underline{h}_q^{(1,q-1)}(k) & \underline{h}_q^{(1,q)}(k) & \underline{v}_1(k) \\ & & & & \\ \underline{h}_q^{(q-1,1)}(k) & \dots & \underline{h}_q^{(2,q-1)}(k) & \underline{h}_q^{(q-1,q)}(k) & \underline{v}_{q-1}(k) \\ \underline{h}_q^{(q,1)}(k) & \dots & \underline{h}_q^{(q,q-1)}(k) & \underline{h}_q^{(q,q)}(k) & \underline{v}_q(k) \end{bmatrix} . \quad (8 - 14)$$

The highlighted terms are the terms involved. After the restructuring, we will have

$$\begin{bmatrix} \mathbf{U}_{q+1}^{(q-1)}(k) & \mathbf{v}_{q+1}(k) \\ \underline{\mathbf{H}}_{q-1}(k) & \underline{\mathbf{v}}_{q-1}(k) \end{bmatrix} = \left[\begin{array}{ccc|c} \mathbf{u}_q^{(1)}(k) & \dots & \mathbf{u}_q^{(q-1)}(k) & \mathbf{v}_q(k) \\ \underline{h}_q^{(q,1)}(k) & \dots & \underline{h}_q^{(q,q-1)}(k) & \underline{v}_q(k) \\ \hline \underline{h}_q^{(1,1)}(k) & \dots & \underline{h}_q^{(1,q-1)}(k) & \underline{v}_1(k) \\ & & & \\ \underline{h}_q^{(q-1,1)}(k) & \dots & \underline{h}_q^{(2,q-1)}(k) & \underline{v}_{q-1}(k) \end{array}\right] . \quad (8 - 15)$$

Specifically, we will move the $\underline{q}^{th}$ row and delete the $\underline{q}^{th}$ column of the time-update correlation matrix of Eq. (8 - 11).

8.3 Order-Decrease-Update Algorithms

In this section, we will derive the order-decrease-update algorithms for the linear model using the batch least squares method. These methods allow efficient recalculation of the new least squares solution when an existing node is removed from the model. Suppose we have a linear model

$$\mathbf{A}_q(k)\mathbf{x}_q(k) = \mathbf{d}(k) \tag{8 - 16}$$

where the data matrix $\mathbf{A}_q(k)$ and the parameter vector $\mathbf{x}_q(k)$ are given by:

$$\begin{aligned} \mathbf{A}_q(k) &= \left[\mathbf{a}_1(k) \ \dots \ \mathbf{a}_{q-1}(k) \ \mathbf{a}_q(k)\right] \\ \mathbf{x}_q(k) &= \left[x_1(k) \ \dots \ x_{q-1}(k) \ x_q(k)\right]^T \end{aligned} . \tag{8 - 17}$$

The batch least squares solution of this linear model is given by

$$\mathbf{x}_q(k) = \mathbf{H}_q(k)\mathbf{A}_q^T(k)\mathbf{d}(k) \tag{8 - 18}$$

where

$$\mathbf{H}_q(k) = (\mathbf{A}_q^T(k)\mathbf{A}_q(k))^{-1} . \tag{8 - 19}$$

Now, suppose we find that the existing node $\mathbf{a}_q(k)$ is no longer useful to the linear model, and we would like to remove it. By removing $\mathbf{a}_q(k)$ from the data matrix, the last column of $\mathbf{A}_q(k)$ is removed.

$$\mathbf{A}_{q-1}(k) = \left[\mathbf{a}_1(k) \ \dots \ \mathbf{a}_{q-1}(k)\right] \tag{8 - 20}$$

Also, the size of $\mathbf{x}_q(k)$ will shrink by one. If we were to recompute the batch least squares solution for this changes, the optimal solution for this new parameter will be given by

$$\mathbf{x}_{q-1}(k) = \mathbf{H}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{d}(k) \tag{8 - 21}$$

where $\mathbf{H}_{q-1}(k) = (\mathbf{A}_{q-1}^T(k)\mathbf{A}_{q-1}(k))^{-1}$. Unfortunately, the computation of $\mathbf{x}_{q-1}(k)$ using Eq. (8 - 21) is very time consuming. A better alternative is to recursively compute

$\mathbf{x}_{q-1}(k)$ based on the previously computed $\mathbf{x}_q(k)$ and $\mathbf{H}_q(k)$. This recursive solution can be obtained by modifying the block matrix inversion lemma in chapter 6 for matrix downdate.

8.3.1 Block Matrix Inversion Lemma for Matrix Downdating

Similar to the recursive order-increase-update algorithm, the block matrix inversion lemma plays an important role in the derivation of the recursive order-decrease-update algorithm. In this section, we modify the block matrix inversion lemma for a matrix downdate. We repeat the special case of the block matrix inversion lemma here so that we can use it to derive a version of the block matrix inversion lemma for the matrix-downdate. The block matrix inversion lemma for the matrix update assumes that we know $\mathbf{A}^{-1}$ and we have $\mathbf{H} \in \Re^{(n+m)\times(n+m)}$ (a square matrix) such that

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}^T & d \end{bmatrix} \tag{8 - 22}$$

where $\mathbf{A} \in \Re^{n\times n}$, $\mathbf{b} \in \Re^{n\times 1}$, $\mathbf{c}^T \in \Re^{1\times n}$, and $d \in \Re^{1\times 1}$. Then, $\mathbf{H}^{-1}$ can be found as

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{b}(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}\mathbf{c}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{b}(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1} \\ -(d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1}\mathbf{c}^T\mathbf{A}^{-1} & (d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b})^{-1} \end{bmatrix}. \tag{8 - 23}$$

The objective of the block matrix inversion lemma for the matrix downdate (the opposite of the block matrix inversion lemma for matrix-update) is to obtain $\mathbf{A}^{-1}$ while

assuming that we have $\mathbf{H}^{-1}$. To show this result, we first simplify Eq. (8 - 23) by letting

$\Delta = d - \mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{b}\Delta^{-1}\mathbf{c}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{b}\Delta^{-1} \\ -\mathbf{c}^T\mathbf{A}^{-1}\Delta^{-1} & \Delta^{-1} \end{bmatrix}. \qquad (8 - 24)$$

By rearranging Eq. (8 - 24), we get

$$\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} = \mathbf{H}^{-1} - \begin{bmatrix} \mathbf{A}^{-1}\mathbf{b}\Delta^{-1}\mathbf{c}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{b}\Delta^{-1} \\ -\mathbf{c}^T\mathbf{A}^{-1}\Delta^{-1} & \Delta^{-1} \end{bmatrix}. \qquad (8 - 25)$$

Since the second term on the right hand side of Eq. (8 - 25) can be factored into

$$\begin{bmatrix} \mathbf{A}^{-1}\mathbf{b}\Delta^{-1}\mathbf{c}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{b}\Delta^{-1} \\ -\mathbf{c}^T\mathbf{A}^{-1}\Delta^{-1} & \Delta^{-1} \end{bmatrix} = \begin{bmatrix} -\mathbf{A}^{-1}\mathbf{b}\Delta^{-1} \\ \Delta^{-1} \end{bmatrix} \begin{bmatrix} -\mathbf{c}^T\mathbf{A}^{-1}\Delta^{-1} \\ \Delta^{-1} \end{bmatrix}^T / \Delta^{-1}, \qquad (8 - 26)$$

we have

$$\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} = \mathbf{H}^{-1} - \begin{bmatrix} -\mathbf{A}^{-1}\mathbf{b}\Delta^{-1} \\ \Delta^{-1} \end{bmatrix} \begin{bmatrix} -\mathbf{c}^T\mathbf{A}^{-1}\Delta^{-1} \\ \Delta^{-1} \end{bmatrix}^T / \Delta^{-1}. \qquad (8 - 27)$$

The significance of Eq. (8 - 27) is that the factorized vectors and scalar are elements in $\mathbf{H}^{-1}$; in other words, we can obtain $\mathbf{A}^{-1}$ by using only the elements in $\mathbf{H}^{-1}$. Specifically, Δ^{-1} is the last diagonal element of $\mathbf{H}^{-1}$, $\begin{bmatrix} -\mathbf{A}^{-1}\mathbf{b}\Delta^{-1} \\ \Delta^{-1} \end{bmatrix}$ is the last row vector of $\mathbf{H}^{-1}$, and $\begin{bmatrix} -\mathbf{c}^T\mathbf{A}^{-1}\Delta^{-1} \\ \Delta^{-1} \end{bmatrix}^T$ is the last column vector of $\mathbf{H}^{-1}$. In term of floating point operation, this matrix downdate requires ~ $O(q(q+2))$.

In addition, this order-decrease-update procedure also applies to any order. That is if we have

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}_{11}^{\#} & \mathbf{A}_{12}^{\#} \\ \mathbf{A}_{21}^{\#} & \mathbf{A}_{22}^{\#} \end{bmatrix}, \mathbf{H} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{b}_1 & \mathbf{A}_{12} \\ \mathbf{c}_1^T & d & \mathbf{c}_2^T \\ \mathbf{A}_{21} & \mathbf{b}_2 & \mathbf{A}_{22} \end{bmatrix} \text{ and } \mathbf{H}^{-1} = \begin{bmatrix} \mathbf{A}_{11}^* & \mathbf{b}_1^* & \mathbf{A}_{12}^* \\ \mathbf{c}_1^{*T} & d^* & \mathbf{c}_1^{*T} \\ \mathbf{A}_{21}^* & \mathbf{b}_2^* & \mathbf{A}_{22}^* \end{bmatrix}, \text{ then}$$

$$\begin{bmatrix} \mathbf{A}_{11}^{\#} & \mathbf{0} & \mathbf{A}_{12}^{\#} \\ \mathbf{0}^T & 0 & \mathbf{0}^T \\ \mathbf{A}_{21}^{\#} & \mathbf{0} & \mathbf{A}_{22}^{\#} \end{bmatrix} = \mathbf{H}^{-1} - \begin{bmatrix} \mathbf{b}_1^* \\ d^* \\ \mathbf{b}_2^* \end{bmatrix} \begin{bmatrix} \mathbf{c}_1^{*T} & d^* & \mathbf{c}_1^{*T} \end{bmatrix} / d^* . \qquad (8 - 28)$$

To prove Eq. (8 - 28), let us assume that

$$\mathbf{B} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{b}_2 \\ \mathbf{c}_1^T & \mathbf{c}_2^T & d \end{bmatrix}, \qquad (8 - 29)$$

and there exist a row permutation matrix $\mathbf{P}_{row}$ *and a column permutation matrix* $\mathbf{P}_{col}$ *such that* $\mathbf{B} = \mathbf{P}_{row}\mathbf{H}\mathbf{P}_{col}$. *Then, according to Eq. (8 - 27), we have*

$$\begin{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} & \begin{matrix} \mathbf{0} \\ \mathbf{0} \end{matrix} \\ \mathbf{0}^T \; \mathbf{0}^T & 0 \end{bmatrix} = \mathbf{B}^{-1} - \begin{bmatrix} -\mathbf{b}_1^* \\ -\mathbf{b}_2^* \\ d^* \end{bmatrix} \begin{bmatrix} -\mathbf{c}_1^* & -\mathbf{c}_2^* & d^* \end{bmatrix} / d^* . \qquad (8 - 30)$$

Since $\mathbf{B} = \mathbf{P}_{row}\mathbf{H}\mathbf{P}_{col}$, *we get* $\mathbf{B}^{-1} = \mathbf{P}_{col}^{-1}\mathbf{H}^{-1}\mathbf{P}_{row}^{-1}$. *Because* $\mathbf{P}_{row}^{-1} = \mathbf{P}_{row}^T$ *and* $\mathbf{P}_{col}^{-1} = \mathbf{P}_{col}^T$, *we get*

$$\begin{bmatrix} \mathbf{A}_{11}^{\#} & \mathbf{A}_{12}^{\#} & \mathbf{0} \\ \mathbf{A}_{21}^{\#} & \mathbf{A}_{22}^{\#} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{0}^T & 0 \end{bmatrix} = \mathbf{P}_{col}^T \mathbf{H}^{-1} \mathbf{P}_{row}^T - \begin{bmatrix} -\mathbf{b}_1^* \\ -\mathbf{b}_2^* \\ d^* \end{bmatrix} \begin{bmatrix} -\mathbf{c}_1^* & -\mathbf{c}_2^* & d^* \end{bmatrix} / d^* . \qquad (8 - 31)$$

Left multiply equation above by $\mathbf{P}_{col}$ *and right multiply equation above by* $\mathbf{P}_{row}$*, we get*

$$\mathbf{P}_{col} \begin{bmatrix} \mathbf{A}_{11}^{\#} & \mathbf{A}_{12}^{\#} & \mathbf{0} \\ \mathbf{A}_{21}^{\#} & \mathbf{A}_{22}^{\#} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{0}^T & 0 \end{bmatrix} \mathbf{P}_{row} = \mathbf{H}^{-1} - \left(\mathbf{P}_{col} \begin{bmatrix} -\mathbf{b}_1^* \\ -\mathbf{b}_2^* \\ d^* \end{bmatrix} \right) \left(\begin{bmatrix} -\mathbf{c}_1^* & -\mathbf{c}_2^* & d^* \end{bmatrix} \mathbf{P}_{row} \right) / d^* . \qquad (8 - 32)$$

Multiply out Eq. (8 - 32), we obtain Eq. (8 - 28).

8.3.2 Recursive Order-Decrease-Update Algorithm for the LS Method

In the following, we apply the result of the matrix downdate in section 8.3.1 to the batch least squares algorithm, and obtain a recursive order-decrease-update algorithm. We first note that the old inverse Hessian matrix before the downdating, taken directly from Eq. (6 - 13), is

$$\bar{\mathbf{H}}_q(k) = \begin{bmatrix} \mathbf{A}_{q-1}^T \mathbf{A}_{q-1} & \mathbf{A}_{q-1}^T \mathbf{A}_{q-1} \mathbf{a}_q \\ \mathbf{a}_q^T \mathbf{A}_{q-1} & \mathbf{a}_q^T \mathbf{a}_q \end{bmatrix}^{-1}$$

$$\begin{bmatrix} (\mathbf{A}_{q-1}^T \mathbf{A}_{q-1})^{-1} + (\mathbf{A}_{q-1}^T \mathbf{A}_{q-1})^{-1} \mathbf{A}_{q-1}^T \mathbf{a}_q \rho_q^{-2} \mathbf{a}_q^T \mathbf{A}_{q-1} (\mathbf{A}_{q-1}^T \mathbf{A}_{q-1})^{-1} & -(\mathbf{A}_{q-1}^T \mathbf{A}_{q-1})^{-1} \mathbf{A}_{q-1}^T \mathbf{a}_q \rho_q^{-2} \\ -\rho_q^{-2} \mathbf{a}_q^T \mathbf{A}_{q-1} (\mathbf{A}_{q-1}^T \mathbf{A}_{q-1})^{-1} & \rho_q^{-2} \end{bmatrix} \qquad (8 - 33)$$

where $\rho_q^2(k) = \mathbf{a}_q^T(k)\mathbf{a}_q(k) - \mathbf{a}_q^T(k)\mathbf{A}_{q-1}(k)\mathbf{H}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{a}_q(k)$.

Now, we apply the inversion result of Eq. (8 - 27) to Eq. (8 - 33), we get

$$
{}^{q}\bar{\mathbf{H}}_{q-1}(k) = \begin{bmatrix} \bar{\mathbf{H}}_{q-1}(k) & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} = \bar{\mathbf{H}}_q(k) -
$$

$$
\begin{bmatrix} -\bar{\mathbf{H}}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{a}_q(k)\rho_q^{-2}(k) \\ \rho_q^{-2}(k) \end{bmatrix} \begin{bmatrix} -\bar{\mathbf{H}}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{a}_q(k)\rho_q^{-2}(k) \\ \rho_q^{-2}(k) \end{bmatrix}^T / \rho_q^{-2}(k) \quad . \quad (8 - 34)
$$

Here, we introduce ${}^{q}\bar{\mathbf{H}}_{q-1}(k)$ as a matrix with zero elements inserted to the qth row and column of $\bar{\mathbf{H}}_{q-1}(k)$. The left superscript indicates which row and column in $\bar{\mathbf{H}}_{q-1}(k)$ get zero elements inserted. Due to the fact that the inverse Hessian matrix is symmetric, we can further simplify Eq. (8 - 34) to

$$
{}^{q}\bar{\mathbf{H}}_{q-1}(k) = \bar{\mathbf{H}}_q(k) - \bar{\mathbf{h}}_q^{(q)}(k)\bar{\mathbf{h}}_q^{(q)^T}(k) / \bar{h}_q^{(q,q)}(k) \qquad (8 - 35)
$$

where

$$
\bar{\mathbf{h}}_q^{(q)}(k) = \begin{bmatrix} -\bar{\mathbf{H}}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{a}_q(k)\rho_q^{-2}(k) \\ \rho_q^{-2}(k) \end{bmatrix}, \text{ and } \bar{h}_q^{(q,q)}(k) = \rho_q^{-2}(k) . \qquad (8 - 36)
$$

Keep in mind that $\bar{\mathbf{h}}_q^{(q)}(k)$ and $\bar{h}_q^{(q,q)}(k)$ are the qth column vector and the qth diagonal element of $\bar{\mathbf{H}}_q(k)$; therefore, the ${}^{q}\bar{\mathbf{H}}_{q-1}(k)$ can be updated using only the elements in $\bar{\mathbf{H}}_q(k)$. Similarly, we can extend this result to the removal of the ith order of $\bar{\mathbf{H}}_q(k)$ using Eq. (8 - 28)

$$
{}^{i}\bar{\mathbf{H}}_{q-1}(k) = \bar{\mathbf{H}}_q(k) - \bar{\mathbf{h}}_q^{(i)}(k)\bar{\mathbf{h}}_q^{(i)^T}(k) / \bar{h}_q^{(i,i)}(k) \quad \text{, where } 1 \le i \le q . \qquad (8 - 37)
$$

Take note that $\bar{\mathbf{h}}_q^{(i)}(k)$ is the ith column vector of $\overline{\mathbf{H}}_q(k)$ and $\bar{h}_q^{(i,i)}(k)$ is the ith diagonal element of $\overline{\mathbf{H}}_q(k)$.

8.3.2.1 Recursive Order-Decrease-Update for the Parameter

Applying the result of Eq. (8 - 37) to Eq. (8 - 21), we obtain the optimal parameter for the new linear model with q^{th} order removed

$$\mathbf{x}_{q-1}(k) = \mathbf{H}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{d}(k). \qquad (8 - 38)$$

Keep in mind that we will need to extract $\mathbf{H}_{q-1}(k)$ from ${}^q\overline{\mathbf{H}}_{q-1}(k)$ to compute Eq. (8 - 38). Also, if we were to use the time-update correlation matrix for Eq. (8 - 38), we will need to extract $\mathbf{v}_{q-1}(k) = \mathbf{A}_{q-1}^T(k)\mathbf{d}(k)$ from $\mathbf{v}_q(k) = \mathbf{A}_q^T(k)\mathbf{d}(k)$ because $\mathbf{v}_q(k)$ is provided by the time-update correlation matrix. However, with the following modification, we can compute the new parameter using ${}^q\overline{\mathbf{H}}_{q-1}(k)$ and $\mathbf{v}_q(k)$ directly.

$${}^q\mathbf{x}_{q-1}(k) = {}^q\overline{\mathbf{H}}_{q-1}(k)\mathbf{v}_q(k) \qquad (8 - 39)$$

The result of Eq. (8 - 39) is the optimal solution of the new parameter ${}^q\mathbf{x}_{q-1}(k)$ but with a zero element inserted to the qth row of $\mathbf{x}_{q-1}(k)$. Again, the left superscript indicates which row in $\mathbf{x}_{q-1}(k)$ gets a zero element inserted.

Now, if we were to apply the generalized form of the matrix downdate Eq. (8 - 37), then we obtain

$${}^i\mathbf{x}_{q-1}(k) = {}^i\overline{\mathbf{H}}_{q-1}(k)\mathbf{v}_q(k). \qquad (8 - 40)$$

8.3.2.2 Recursive Order-Decrease-Update for the Sum of Squared Errors

Recall from section 6.2.2.2 that the sum of squared error of a linear model will always be reduced in value when an order is added. This result is verified by the derivation of the recursive sum of squared error formula in Eq. (6 - 30) when we consider adding a new node to a linear model.

Now, assume that a new node is included in the model (that means we have q nodes in the model), and we would like to remove the last (q^{th}) node. The same recursive sum of squared errors formula applied in the recursive order-update can be used to find the new sum of squared errors. This is done by the rearranging the recursive sum of squared error formula in Eq. (6 - 30)

$$\mathbf{e}_{q-1}^{T}(k)\mathbf{e}_{q-1}(k) = \mathbf{e}_{q}^{T}(k)\mathbf{e}_{q}(k) + {\rho_{q-1}^{(q)}}^{-2}(k){\phi_{q-1}^{(q)}}^{2}(k) \tag{8 - 41}$$

and we get

$$SSE_{q-1}^{(q)}(k) = SSE_{q}(k) + err_{q-1}^{(q)}(k). \tag{8 - 42}$$

Note that we change the order index from $q+1$ to $q-1$ to reflect an order-decrease-update.

Notice that instead of having a minus sign in the recursive sum of squared errors formula, we have a plus sign. This implies that if we take out the qth node, the new sum of squared error $SSE_{q-1}^{(q)}(k)$ will increase by the amount given by $err^{(q)}(k)$. The same error reduction term $err^{(q)}(k)$, used by the order-increase-update, is also a measurement of the

error increase for the order-decrease-update. Therefore, applying Eq. (8 - 35) to calculate the error reduction term will give us an idea of how much the qth node contributes to the linear model. Specifically, we need to use Eq. (8 - 35) to calculate the inverse Hessian matrix of ${}^{q}\overline{\mathbf{H}}_{q-1}(k)$, then extract $\mathbf{H}_{q-1}(k)$ from ${}^{q}\overline{\mathbf{H}}_{q-1}(k)$ and use it in the following error reduction calculation

$$err_{q-1}^{(q)}(k) = \rho_{q-1}^{(q)\,-2}(k)\phi_{q-1}^{(q)\,2}(k) \tag{8 - 43}$$

where

$$\phi_{q-1}^{(q)}(k) = \mathbf{a}_q^T(k)\mathbf{d}(k) - \mathbf{a}_q^T(k)\mathbf{A}_{q-1}(k)\mathbf{H}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{d}(k)\text{, and} \tag{8 - 44}$$

$$\rho_{q-1}^{(q)\,2}(k) = \mathbf{a}_q^T(k)\mathbf{a}_q(k) - \mathbf{a}_q^T(k)\mathbf{A}_{q-1}(k)\overline{\mathbf{H}}_{q-1}(k)\mathbf{A}_{q-1}^T(k)\mathbf{a}_q(k)\,. \tag{8 - 45}$$

Together with the inverse Hessian matrix $\mathbf{H}_{q-1}(k)$ computed in Eq. (8 - 35), we have all the necessary terms required for the error reduction calculation.

We can extend the result of Eq. (8 - 45) to calculate the error reduction for any i^{th} node available in the linear model using Eq. (8 - 37) instead of Eq. (8 - 35). Also, we can use the time-update correlation matrix in Eq. (8 - 7) to compute the necessary terms needed for the error reduction. To accommodate these changes, we modify Eq. (8 - 44) and Eq. (8 - 45) as follow:

$$err_{q-1}^{(i)}(k) = \rho_{q-1}^{(i)\,-2}(k)\phi_{q-1}^{(i)\,2}(k) \tag{8 - 46}$$

$$\phi_{q-1}^{(i)}(k) = \underbrace{\mathbf{a}_i^T(k)\mathbf{d}(k)}_{v_q^{(i)}(k)} - \underbrace{\mathbf{a}_i^T(k)\mathbf{A}_q(k)}_{\mathbf{h}_q^{(i)T}(k)}\ {}^{i}\overline{\mathbf{H}}_{q-1}(k)\ \underbrace{\mathbf{A}_q^T(k)\mathbf{d}(k)}_{\mathbf{v}_q(k)}\text{, and} \tag{8 - 47}$$

$$\rho_{q-1}^{(i)\,2}(k) = \underbrace{\mathbf{a}_i^T(k)\mathbf{a}_i(k)}_{h_q^{(i,i)}(k)} - \underbrace{\mathbf{a}_i^T(k)\mathbf{A}_q(k)}_{\mathbf{h}_q^{(i)T}(k)} \overset{i-}{\mathbf{H}}_{q-1}(k) \underbrace{\mathbf{A}_q^T(k)\mathbf{a}_i(k)}_{\mathbf{h}_q^{(i)}(k)} \quad . \tag{8 - 48}$$

This modification works by using the fact that

$$\mathbf{A}_q(k)\overset{i-}{\mathbf{H}}_{q-1}(k)\mathbf{A}_q^T(k) = \mathbf{A}_{q-1}(k)\overline{\mathbf{H}}_{q-1}(k)\mathbf{A}_{q-1}^T(k). \tag{8 - 49}$$

With this modification, we can apply the time-update correlation matrix to the error reduction calculation. Specifically, $v_q^{(i)}(k)$ is the ith element of the $\mathbf{v}_q(k)$ vector, $h_q^{(i,i)}(k)$ is the diagonal element associated with the ith row and the ith column of the $\mathbf{H}_q(k)$ matrix, and $\mathbf{h}_q^{(i)}(k)$ is the ith column vector of the $\mathbf{H}_q(k)$ matrix. Also, $\overset{i-}{\mathbf{H}}_{q-1}(k)$ is used directly in the error reduction calculation, which avoid the trouble of extracting $\mathbf{H}_{q-1}(k)$ from $\overset{i-}{\mathbf{H}}_{q-1}(k)$. In term of floating operations, the computation of the order-decrease-update of the ith node requires ~ $O(2q^2 + 7q)$. In the following, we will show an example using this order-decrease-update algorithm.

Example 8 - 1

Assume that we have the following data matrix and desired response

$$\mathbf{A}_3(4) = \begin{bmatrix} 0.1 & -0.2 & 0.3 \\ -0.5 & 0.3 & 0.3 \\ 0.8 & -0.2 & 0.4 \\ 0.6 & -0.8 & 0.9 \end{bmatrix}, \text{ and } \mathbf{d}(4) = \begin{bmatrix} 0.1 \\ -0.2 \\ -0.5 \\ 1 \end{bmatrix}. \tag{8 - 50}$$

Then, the time-update correlation matrix is

$$\left[\mathbf{H}_3(4) \,\vdots\, \mathbf{v}_3(4)\right] = \begin{bmatrix} 1.26 & -0.81 & 0.74 & \vdots & 0.31 \\ -0.81 & 0.81 & -0.77 & \vdots & -0.78 \\ 0.74 & -0.77 & 1.15 & \vdots & 0.67 \end{bmatrix}. \tag{8 - 51}$$

[Note that because there is no unselected node, the time-update correlation matrix does not contain the terms $\underline{\mathbf{H}}_q(k)$, $\mathbf{U}_q(k)$ *and* $\underline{\mathbf{v}}_q(k)$ *.] The inverse correlation matrix is given by*

$$\overline{\mathbf{H}}_3(4) = \begin{bmatrix} 2.2329 & 2.3852 & 0.1602 \\ 2.3852 & 5.9443 & 2.4452 \\ 0.1602 & 2.4452 & 2.4037 \end{bmatrix}. \tag{8 - 52}$$

Now, if we wish to remove the last node, we can apply Eq. (8 - 37) with $i = 3$

$${}^{3}\overline{\mathbf{H}}_2(4) = \overline{\mathbf{H}}_3(4) - \begin{bmatrix} 0.1602 \\ 2.4452 \\ 2.4037 \end{bmatrix} \frac{\begin{bmatrix} 0.1602 & 2.4452 & 2.4037 \end{bmatrix}}{2.4037} = \begin{bmatrix} 2.2222 & 2.2222 & 0 \\ 2.2222 & 3.4568 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{8 - 53}$$

Using the solution in Eq. (8 - 53), we can calculate the new parameter as

$${}^{3}\mathbf{x}_2(4) = \begin{bmatrix} 2.2222 & 2.2222 & 0 \\ 2.2222 & 3.4568 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.31 \\ -0.78 \\ 0.67 \end{bmatrix} = \begin{bmatrix} -1.0444 \\ -2.0074 \\ 0 \end{bmatrix}, \; \mathbf{x}_2(4) = \begin{bmatrix} -1.0444 \\ -2.0074 \end{bmatrix}, \tag{8 - 54}$$

and the error reduction term as

$$\phi_3^{(3)}(4) = 0.67 - \begin{bmatrix} 0.74 \\ -0.77 \\ 1.15 \end{bmatrix}^T \begin{bmatrix} 2.2222 & 2.2222 & 0 \\ 2.2222 & 3.4568 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.31 \\ -0.78 \\ 0.67 \end{bmatrix} = -0.1028, \tag{8 - 55}$$

$$\rho_3^{(3)^2}(4) = 1.15 - \begin{bmatrix} 2.2329 \\ 2.3852 \\ 0.1602 \end{bmatrix}^T \begin{bmatrix} 2.2222 & 2.2222 & 0 \\ 2.2222 & 3.4568 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2.2329 \\ 2.3852 \\ 0.1602 \end{bmatrix} = 0.416. \tag{8 - 56}$$

$$err_3^{(3)}(4) = \frac{(-0.1028)^2}{0.416} = 0.0254. \tag{8 - 57}$$

The result of the order-decrease-update for $i = 2$ and $i = 1$ is as follows

$$^{2}\overline{\mathbf{H}}_2(4) = \begin{bmatrix} 1.2758 & 0 & -0.8209 \\ 0 & 0 & 0 \\ -0.8209 & 0 & 1.3978 \end{bmatrix}, \; {}^{2}\mathbf{x}_2(4) = \begin{bmatrix} -0.1545 \\ 0 \\ 0.6821 \end{bmatrix}, \; err_3^{(2)}(4) = 0.8583, \tag{8 - 58}$$

$$^{1}\overline{\mathbf{H}}_2(4) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3.3963 & 2.2741 \\ 0 & 2.2741 & 2.3922 \end{bmatrix}, \; {}^{1}\mathbf{x}_2(4) = \begin{bmatrix} 0 \\ -1.1255 \\ -0.1710 \end{bmatrix}, \; err_3^{(1)}(4) = 0.5041. \tag{8 - 59}$$

8.4 Recursive Backward Elimination

In Example 8 - 1, the calculation of error reduction $err_3^{(3)}(4)$ yields the smallest value when we remove the third node. This seems to suggest that removing this node from the model is probably a good idea since it does not contribute much to the model. Indeed, this idea is called the backward elimination. We have briefly discussed the batch backward elimination in chapter 6. As discussed, this method calculates the error reduction for all the nodes for a set of data, then removes the node that yields the smallest computed error reduction term one at a time until an error criterion is met. In term of floating point operation, this operation requires ~ $O(2q^3 + 7q^2)$.

It is true that the batch backward elimination method works well in finding a small subset in a linear model (Miller, 1990). Unfortunately, it is not true if we were to use this

backward elimination method for real-time operation. This is because recursive backward elimination can only increase the sum of squared errors of a linear model, as we explained in section 8.3.2.2. Therefore, it cannot compensate for the new changes induces by the newly added real-time data.

At this point, the reader might ask why we introduce the recursive backward elimination method if it does not work in real-time. The reason is that backward elimination can work together with forward selection to form the Efroymson algorithm.

8.5 Recursive Efroymson Algorithm

In the following, we will first discuss the batch Efroymson algorithm (Most of the batch Efroymson materials are taken from Miller (1990), and Efroymson (1960). Readers can consult these books and journals for detailed discussion). Then, we will use some of the procedures from the batch Efroymson algorithm to develop the recursive Efroymson algorithm.

8.5.1 Batch Efroymson Algorithm

As described in chapter 6, the batch Efroymson algorithm works by combining forward selection and backward elimination. Specifically, the batch Efroymson algorithm will first use the forward selection method to find the best potential node among all the unselected nodes. The best potential node is the node that produce the largest decrease in

the sum of squared errors. The sum of squared errors before, SSE_q, and after, SSE_{q+1}, the addition of this node are then used in the calculation of the following ratio

$$W_e = \frac{SSE_q - SSE_{q+1}}{SSE_{q+1}/(N-q-2)} \qquad (8 - 60)$$

where N is the total number of data points and q is the total number of nodes in the model. The calculated W_e is compared to the F-to-enter value F_e. If $W_e > F_e$, the node is added to the model. Otherwise, no node is added.

Right after forward selection, backward elimination is applied to see if any of the previously selected nodes can be deleted without appreciably increasing the sum of squared errors. The node that incurs the least sum of squared error is picked. Then, the sum of squared errors before, SSE_q, and after, SSE_{q-1}, the deletion of that selected node are used in the calculation of the following ratio

$$W_d = \frac{SSE_{q-1} - SSE_q}{SSE_q/(N-q-1)}. \qquad (8 - 61)$$

This W_d value is calculated and compare to F-to-delete value F_d. If $W_d < F_d$, the node is removed from the model. Otherwise, no node is removed. This process is repeated until no further additions and deletions are possible which satisfy the criteria.

According to Miller (1990), this process is guaranteed to converge as long as we set $F_d < F_e$. Also, the stopping rule is derived assuming that W_e and W_d have an F-distribution under the null hypothesis; that is, the model is the true model and subject to error residuals being independent and normally distributed. Typically, F_e and F_d values

are taken from an F-distribution table. A typical value for F_e is 2 and a typical value for F_d is 1.5. However, according to Miller (1990), these values can be manually set as long as $F_d < F_e$. Specifically, large F_e and small F_d will yield a smaller model with a larger sum of squared errors; small F_e and large F_d will yield a larger model with a smaller sum of squared errors. We can also use other techniques such as Mallow's (1973) statistic, adjusted R^2 statistic, or Akaike's (1969) Information Criterion (AIC) to derive different stopping rules.

8.5.2 Recursive Efroymson Algorithm

While the batch Efroymson algorithm is an off-line learning method, the recursive Efroymson algorithm operates in on-line fashion. That means it will have to deal with a time-update while simultaneously applying the Efroymson algorithm to add and/or delete orders from the model.

8.5.2.1 Stopping Rule

Using the error reduction calculations for forward selection, Eq. (8 - 8), and backward elimination, Eq. (8 - 46),

$$err_{q+1}^{(i)}(k) = SSE_q(k) - SSE_{q+1}^{(i)}(k) \text{ for } i = 1 < i < q \text{, and} \qquad (8 - 62)$$

$$err_{q-1}^{(i)}(k) = SSE_{q-1}^{(i)}(k) - SSE_q(k) \text{ for } i = 1 < i < q \text{,} \qquad (8 - 63)$$

the recursive Efroymson is a straight forward extension to the batch Efroymson algorithm. The result of Eq. (8 - 62) and Eq. (8 - 63) can be used directly in the W_e and W_d calculations

$$W_e = \frac{err_{q+1}^{(i)}(k)}{SSE_{q+1}^{(i)}(k)/(k-q-2)} \tag{8 - 64}$$

$$W_d = \frac{err_{q-1}^{(i)}(k)}{SSE_q(k)/(k-q-1)}. \tag{8 - 65}$$

The calculated W_e and W_d are compared to F_e and F_d. If $W_e > F_e$, the node is added to the model. If $W_d < F_d$, the node is removed from the model. Otherwise, no node is removed or added.

Keep in mind that because we need to accommodate for the real-time operation, an assumption is made such that these stopping rules are applied only once for each time-point. In other words, we will have only one deletion and/or deletion takes place in one time-step. This assumption assumes that the repeating process of the recursive Efroymson algorithm takes place as time-point increases. Note that this repeating process is vital for the error convergence of the algorithm. In Chapter 10, we will run numerous simulation tests to validate this assumption.

8.5.2.2 Restructuring the Time-Update Correlation Matrix

Similar to section 8.2.3, we need to restructure the time-update correlation matrix when we use the recursive Efroymson method for the order-increase-update and order-decrease-update. Here, we made an assumption that if a node is removed from the model

(even the node that has just been added), that node is added back to the potential nodes for future selection. This assumption assumes that it is possible for the algorithm to reselect it again in the future.

With this assumption, we revisit the restructuring of the time-update correlation matrix. Suppose we expanded $\mathbf{U}_q^{(q)}(k)$, $\underline{\mathbf{H}}_q(k)$, and $\underline{\mathbf{v}}_q(k)$, as shown in Eq. (8 - 12) and Eq. (8 - 13), of the time-update correlation matrix

$$\left[\begin{array}{c|c|c} \mathbf{H}_q(k) & \mathbf{U}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \hline \mathbf{U}_q^{(q)^T}(k) & \underline{\mathbf{H}}_q(k) & \underline{\mathbf{v}}_q(k) \end{array}\right] = \left[\begin{array}{c|cccc|c} \mathbf{H}_q(k) & \mathbf{u}_q^{(1)}(k) & \ldots & \mathbf{u}_q^{(q-1)}(k) & \mathbf{u}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \hline \mathbf{u}_q^{(1)^T}(k) & \underline{h}_q^{(1,1)}(k) & \ldots & \underline{h}_q^{(1,q-1)}(k) & \underline{h}_q^{(1,q)}(k) & \underline{v}_1(k) \\ & & & & & \\ \mathbf{u}_q^{(q-1)^T}(k) & \underline{h}_q^{(q-1,1)}(k) & \ldots & \underline{h}_q^{(2,q-1)}(k) & \underline{h}_q^{(q-1,q)}(k) & \underline{v}_{q-1}(k) \\ \mathbf{u}_q^{(q)^T}(k) & \underline{h}_q^{(q,1)}(k) & \ldots & \underline{h}_q^{(q,q-1)}(k) & \underline{h}_q^{(q,q)}(k) & \underline{v}_q(k) \end{array}\right] \quad (8 - 66)$$

and we determine that it is necessary to add the $\underline{q}^{th}$ potential node into the model. Then we will need to move the $\underline{q}^{th}$ row and the $\underline{q}^{th}$ column of the time-update correlation matrix of Eq. (8 - 66). The highlighted terms are the affected elements. After the restructuring, we obtain

$$\left[\begin{array}{c|c|c} \mathbf{H}_{q+1}(k) & \mathbf{U}_{q+1}^{(q-1)}(k) & \mathbf{v}_{q+1}(k) \\ \hline \mathbf{U}_{q+1}^{(q-1)^T}(k) & \underline{\mathbf{H}}_{q-1}(k) & \underline{\mathbf{v}}_{q-1}(k) \end{array}\right] = \left[\begin{array}{cc|ccc|c} \mathbf{H}_q(k) & \mathbf{u}_q^{(q)}(k) & \mathbf{u}_q^{(1)}(k) & \ldots & \mathbf{u}_q^{(q-1)}(k) & \mathbf{v}_q(k) \\ \mathbf{u}_q^{(q)^T}(k) & \underline{h}_q^{(q,q)}(k) & \underline{h}_q^{(q,1)}(k) & \ldots & \underline{h}_q^{(q,q-1)}(k) & \underline{v}_q(k) \\ \hline \mathbf{u}_q^{(1)^T}(k) & \underline{h}_q^{(1,q)}(k) & \underline{h}_q^{(1,1)}(k) & & \underline{h}_q^{(1,q-1)}(k) & \underline{v}_1(k) \\ & & & \ldots & & \\ \mathbf{u}_q^{(q-1)^T}(k) & \underline{h}_q^{(q-1,q)}(k) & \underline{h}_q^{(q-1,1)}(k) & \ldots & \underline{h}_q^{(2,q-1)}(k) & \underline{v}_{q-1}(k) \end{array}\right]. \quad (8 - 67)$$

Note that the above method assumes that we will add the selected potential node to the $q+1$ position of the data matrix. It is also possible to add the selected node to other positions but we will not discuss that here.

Similarly, suppose we expand the matrix

$$\mathbf{U}_q^{(q)}(k) = \left[{}_1\mathbf{u}_q(k)\ {}_2\mathbf{u}_q(k)\ \dots\ {}_q\mathbf{u}_q(k)\right]^T,\ \mathbf{v}_q(k) = \left[v_1(k)\ v_2(k)\ \dots\ v_q(k)\right]^T, \quad (8-68)$$

$$\text{and } \mathbf{H}_q(k) = \begin{bmatrix} h_q^{(1,1)}(k) & h_q^{(1,2)}(k) & \dots & h_q^{(1,q)}(k) \\ h_q^{(2,1)}(k) & h_q^{(2,2)}(k) & \dots & h_q^{(2,q)}(k) \\ & & & \\ h_q^{(q,1)}(k) & h_q^{(q,2)}(k) & \dots & h_q^{(q,q)}(k) \end{bmatrix}. \quad (8-69)$$

(Note that ${}_i\mathbf{u}_q(k)$ is the row-wise expansion of $\mathbf{U}_q^{(q)}(k)$ with the left subscript to indicate i^{th} row). Now, the time-update correlation matrix can be written as

$$\left[\begin{array}{c:c:c} \mathbf{H}_q(k) & \mathbf{U}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \hdashline \mathbf{U}_q^{(q)T}(k) & \underline{\mathbf{H}}_q(k) & \underline{\mathbf{v}}_q(k) \end{array}\right] = \left[\begin{array}{cccc:c:c} h_q^{(1,1)}(k) & h_q^{(1,2)}(k) & \dots & h_q^{(1,q)}(k) & {}_1\mathbf{u}_q{}^T(k) & v_1(k) \\ h_q^{(2,1)}(k) & h_q^{(2,2)}(k) & \dots & h_q^{(2,q)}(k) & {}_2\mathbf{u}_q{}^T(k) & v_2(k) \\ & & & & & \\ h_q^{(q,1)}(k) & h_q^{(q,2)}(k) & \dots & h_q^{(q,q)}(k) & {}_q\mathbf{u}_q{}^T(k) & v_q(k) \\ \hdashline {}_1\mathbf{u}_q(k) & {}_2\mathbf{u}_q(k) & \dots & {}_q\mathbf{u}_q(k) & \underline{\mathbf{H}}_q(k) & \underline{\mathbf{v}}_q(k) \end{array}\right]. \quad (8-70)$$

Suppose we want to remove the 1st node in the model. We remove it from the model and append the unwanted node to the last column of the potential data matrix. To accommodate this change, we need to restructure the time-update correlation matrix as follows:

$$
\left[\begin{array}{c|c|c}
\mathbf{H}_{q-1}(k) & \mathbf{U}_{q-1}^{(q+1)}(k) & \mathbf{v}_{q-1}(k) \\
\hline
\mathbf{U}_{q-1}^{(q+1)^T}(k) & \underline{\mathbf{H}}_{q+1}(k) & \underline{\mathbf{v}}_{q+1}(k)
\end{array}\right] =
\left[\begin{array}{ccc|cc|c}
h_q^{(2,2)}(k) & \dots & h_q^{(2,q)}(k) & {}_2\mathbf{u}_q{}^T(k) & h_q^{(2,1)}(k) & v_2(k) \\
 & & & & & \\
h_q^{(q,2)}(k) & \dots & h_q^{(q,q)}(k) & {}_q\mathbf{u}_q{}^T(k) & h_q^{(q,1)}(k) & v_q(k) \\
\hline
{}_2\mathbf{u}_q(k) & \dots & {}_q\mathbf{u}_q(k) & \underline{\mathbf{H}}_q(k) & {}_1\mathbf{u}_q{}^T(k) & \underline{\mathbf{v}}_q(k) \\
h_q^{(1,2)}(k) & \dots & h_q^{(1,q)}(k) & {}_1\mathbf{u}_q(k) & h_q^{(1,1)}(k) & v_1(k)
\end{array}\right]. \qquad (8-71)
$$

8.5.3 RLS-AWS Algorithm: Efroymson Method

Using the stopping rules and the time-update correlation matrix above, we are now ready to describe the complete RLS-AWS with Efroymson method.

Figure 8 - 1 shows the new framework for the RLS-AWS algorithm with Efroymson method. This framework is similar to the general time- and order- update framework except that we have added the time-update correlation matrix block and we use a different subset selection algorithm: Efroymson method.

As shown in Figure 8 - 1, the algorithm begins by feeding new real-time data $\{{}_q\mathbf{a}(k), d(k), {}_{\underline{q}}\mathbf{a}(k)\}$ to the time-update algorithm and the time-update correlation matrix. Then, the time-update algorithm will produce parameters $\mathbf{x}_q(k)$, $\overline{\mathbf{H}}_q(k)$, and $SSE_q(k)$ for the Efroymson algorithm. Simultaneously, the time-update correlation matrix will produce parameters $\mathbf{H}_q(k)$, $\underline{\mathbf{H}}_q(k)$, $\mathbf{U}_q^{(q)}(k)$, $\mathbf{v}_q(k)$, and $\underline{\mathbf{v}}_q(k)$ for the Efroymson algorithm. Using the parameters from both the time-update algorithm and the time-update correlation matrix, the Efroymson algorithm will proceed to add/delete order. In the following, we will explain the operations contained in the Efroymson algorithm.

8.5.3.1 Subset Selection with Efroymson Method

The role of the Efroymson algorithm is to ensure the time-update algorithm provides adequate network performance. To ensure adequate network performance, we have adopted the following rule: we will first apply the deletion of nodes. If no deletion of nodes is necessary, then we will apply the addition of nodes. If the addition of the nodes is not necessary, then no order-update is necessary.

Hence, the algorithm will first determine if an order-decrease-update is necessary. This is done by computing the error reduction terms for every selected node. Then, it selects the least error reduction term and uses it to calculate W_d in Eq. (8 - 65). This W_d value is compare to a preselected F-to-delete value to determine if an order-decrease-update is necessary. If an order-decrease-update is necessary, we will update the parameters and restructure the time-update correlation matrix.

If an order-decrease-update is not necessary, then the algorithm will first determine if an order-increase-update is necessary. This is done by computing the error reduction terms for every potential node. Then, it selects the largest error reduction term and uses it to calculate W_e in Eq. (8 - 64). This W_e value is compared to a preselected F-to-enter value to determine if an order-increase-update is necessary. If an order-increase-update is necessary, we will update the parameters and restructure the time-update correlation matrix. At this stage, if order-increase-update is not necessary, then the algorithm will not have any order-update. The cycle repeats with a new data point. In the following, we summarize the whole algorithm.

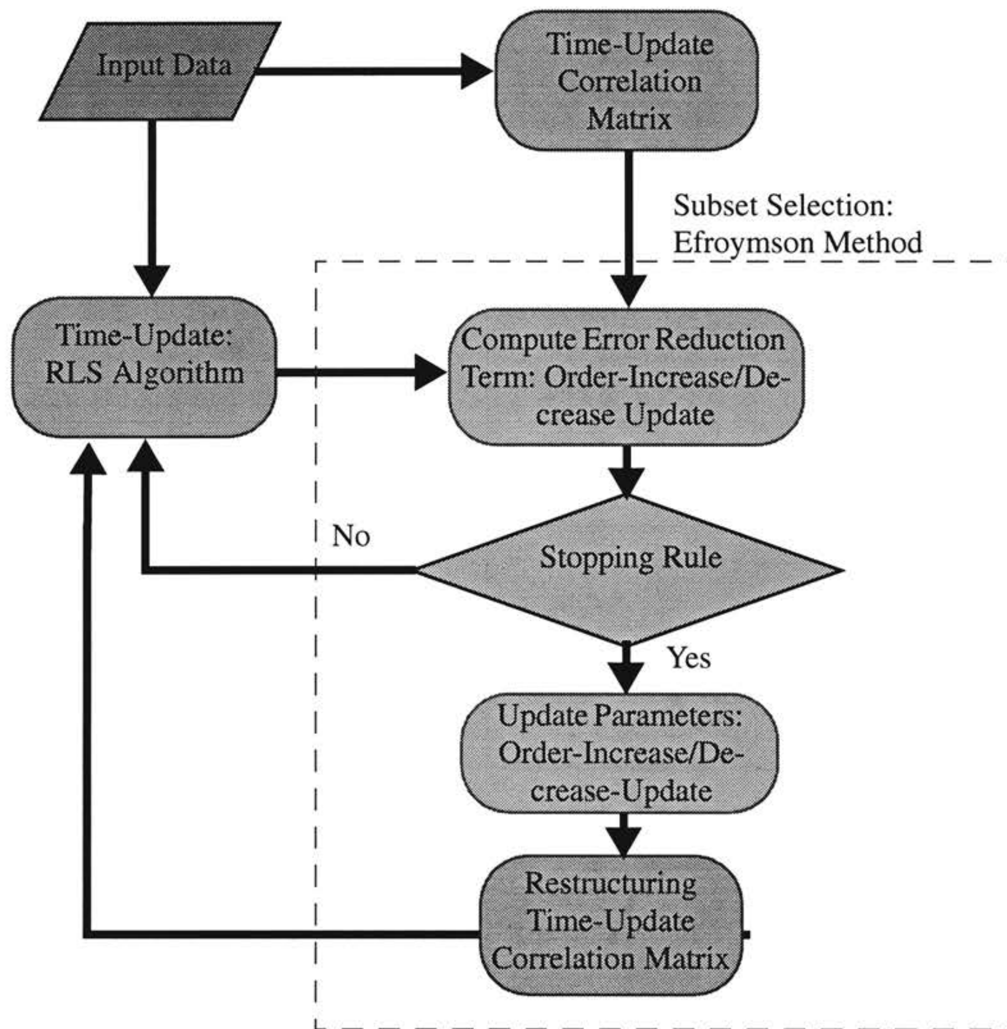


Figure 8 - 1 Flow Chart for RLS-AWS with Efroymson Method

Recursive Efroymson Algorithm

Initialization:

$q = 0$, $\bar{\bar{\mathbf{H}}}_0(0) = \delta^{-1}$, $\mathbf{x}_0(0) = 0$, $\mathbf{H}_0(0) = 0$, $\mathbf{U}_0^{(q)}(0) = \mathbf{0}_{1 \times q}$,

$\underline{\mathbf{H}}_q(0) = \mathbf{0}_{q \times q}$, $\mathbf{v}_0(0) = 0$, $\underline{\mathbf{v}}_q(0) = \mathbf{0}_{1 \times q}$

Input Data:

$\{{}_q\mathbf{a}(k), d(k), {}_{\underline{q}-}\mathbf{a}(k)\}$

Time-Update Correlation Matrix:

$$\begin{bmatrix} \mathbf{H}_q(k) & \mathbf{U}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \mathbf{U}_q^{(q)T}(k) & \underline{\mathbf{H}}_{\underline{q}}(k) & \underline{\mathbf{v}}_{\underline{q}}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{H}_q(k-1) & \mathbf{U}_q^{(q)}(k-1) & \mathbf{v}_q(k-1) \\ \mathbf{U}_q^{(q)T}(k-1) & \underline{\mathbf{H}}_{\underline{q}}(k-1) & \underline{\mathbf{v}}_{\underline{q}}(k-1) \end{bmatrix} +$$

$$\begin{bmatrix} {}_q\mathbf{a}(k) \\ {}_{\underline{q}}\underline{\mathbf{a}}(k) \end{bmatrix} \begin{bmatrix} {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & d(k) \end{bmatrix}$$

Parameters $\mathbf{H}_q(k)$, $\underline{\mathbf{H}}_{\underline{q}}(k)$, $\mathbf{U}_q^{(q)}(k)$, $\mathbf{v}_q(k)$, *and* $\underline{\mathbf{v}}_{\underline{q}}(k)$ *are passed to compute error reduction terms in subset selection.*

Time-Update: RLS Algorithm

$$\kappa_q(k) = 1 + {}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)$$

$$\mathbf{k}_q(k) = \frac{\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)}{1 + {}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)} = \kappa_q^{-1}(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)$$

$$\xi_q(k) = d(k) - {}_q\mathbf{a}^T(k)\mathbf{x}_q(k-1)$$

$$\mathbf{x}_q(k) = \mathbf{x}_q(k-1) + \mathbf{k}_q(k)\xi_q(k)$$

$$\overline{\mathbf{H}}_q(k) = \overline{\mathbf{H}}_q(k-1) - \mathbf{k}_q(k)\,{}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)$$

$$SSE_q(k) = SSE_q(k-1) + \xi_q^2(k)\kappa_q^{-1}(k)$$

Parameters $\mathbf{x}_q(k)$, $\overline{\mathbf{H}}_q(k)$, $SSE_q(k)$ *are passed to compute error reduction terms in subset selection.*

Compute Error Reduction Terms

For Order-Increase-Update:

For $1 \le i \le \underline{q}$ *compute*

$$\phi_{q+1}^{(i)}(k) = \underline{v}^{(i)}(k) - \mathbf{u}_q^{(i)T}(k)\mathbf{x}_q(k)$$

$$\mathbf{z}_q^{(i)}(k) = \overline{\mathbf{H}}_q(k)\mathbf{u}_q^{(i)}(k)$$

$$\rho_{q+1}^{(i)\,2}(k) = \underline{h}_{\underline{q}}^{(i,i)}(k) - \mathbf{u}_q^{(i)T}(k)\mathbf{z}_q^{(i)}(k)$$

$$err_{q+1}^{(i)}(k) = \rho_{q+1}^{(i)\,-2}(k)\phi_{q+1}^{(i)\,2}(k)$$

end

For Order-Decrease-Update:

For $i = 1 \ldots q$ *compute*

$$^{i}\overline{\mathbf{H}}_{q-1}(k) = \overline{\mathbf{H}}_q(k) - \bar{\mathbf{h}}_q^{(i)}(k)\bar{\mathbf{h}}_q^{(i)T}(k)/\bar{h}_q^{(i,i)}(k)$$

$\bar{\mathbf{h}}_q^{(i)}(k)$ *is the* i^{th} *column vector of* $\overline{\mathbf{H}}_q(k)$, *and*

$\bar{h}_q^{(i,i)}(k)$ *is the* i^{th} *diagonal element of* $\overline{\mathbf{H}}_q(k)$.

$$\phi_{q-1}^{(i)}(k) = v_q^{(i)}(k) - \mathbf{h}_q^{(i)T}(k)\,{}^{i}\overline{\mathbf{H}}_{q-1}(k)\mathbf{v}_q(k), \textit{ and}$$

$$\rho_{q-1}^{(i)\,2}(k) = h_q^{(i,i)}(k) - \mathbf{h}_q^{(i)T}(k)\,{}^{i}\overline{\mathbf{H}}_{q-1}(k)\mathbf{h}_q^{(i)}(k)$$

$$err_{q-1}^{(i)}(k) = \rho_{q-1}^{(i)\,-2}(k)\phi_{q-1}^{(i)\,2}(k)$$

end

Stopping Rules:

For Order-Increase-Update

Pick $err_{q+1}(k) = max[err_{q+1}^{(i)}(k)]$ *for* $1 \le i \le q$

$$SSE_{q+1}(k) = SSE_q(k) - err_{q+1}(k)$$

If $\dfrac{err_{q+1}(k)}{SSE_{q+1}(k)/(k-q-2)} > F_e$, *order-increase-update parameters.*

For Order-Decrease-Update

Pick $err_{q-1}(k) = min[err_{q-1}^{(i)}(k)]$ *for* $1 \le i \le q$

$$SSE_{q-1}(k) = SSE_q(k) + err_{q-1}(k)$$

If $\dfrac{err_{q-1}(k)}{SSE_q(k)/(k-q-1)} < F_d$ *order-decrease-update parameters.*

Update Parameters:

For Order-Increase-Update

Keep $SSE_{q+1}(k)$. *Let* $\tilde{i} = index[err_q^{(i)}(k)]$ *then update*

$$\rho_{q+1}^{-2}(k) = \rho_{q+1}^{(\tilde{i})\,2}(k),\ \phi_{q+1}(k) = \phi_{q+1}^{(\tilde{i})}(k),\ \mathbf{z}_q(k) = \mathbf{z}_q^{(\tilde{i})}(k),$$

$$\overline{\mathbf{H}}_{q+1}(k) = \left[\begin{array}{c|c} \overline{\mathbf{H}}_q(k) + \mathbf{z}_q(k)\rho_{q+1}^{-2}(k)\mathbf{z}_q^T(k) & -\mathbf{z}_q(k)\rho_{q+1}^{-2}(k) \\ \hline -\rho_{q+1}^{-2}(k)\mathbf{z}_q^T(k) & \rho_{q+1}^{-2}(k) \end{array}\right]$$

$$\mathbf{x}_{q+1}(k) = \begin{bmatrix} \mathbf{x}_q(k) \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{z}_q(k)\rho_{q+1}^{-2}(k)\phi_{q+1}(k) \\ \rho_{q+1}^{-2}(k)\phi_{q+1}(k) \end{bmatrix}$$

For Order-Decrease-Update

Keep $SSE_{q-1}(k)$. *Let* $\tilde{i} = index[min[err_{q-1}^{(i)}(k)]]$, *then*

calculate ${}^{\tilde{i}}\mathbf{x}_{q-1}(k) = {}^{\tilde{i}}\overline{\mathbf{H}}_{q-1}(k)\mathbf{v}_q(k)$, *and extract* $\mathbf{x}_{q-1}(k)$

Extract $\mathbf{H}_{q-1}(k)$ *from* ${}^{\tilde{i}}\overline{\mathbf{H}}_{q-1}(k)$

8.6 Implementation Consideration

In this section, we will consider applying the recursive Efroymson algorithm to the special RBF network described in chapter 2. Also, we will discuss a method to reduce the computational time and incorporate exponential windowing.

8.6.1 Exponential Windowing

In recursive implementation, we often use an exponential window with the least squares algorithm. Since the time- and order- update algorithms are derived using least squares, we can easily incorporate an exponential window to the new algorithms as well. Since many books and journal (Haykin, 1996; Sayed & Kailath 1994) have in-depth coverage of the derivation and usage of exponential windows in recursive least squares, we will use these their results to derive a version of exponential windowing for the recursive algorithms discussed here.

As discussed in Haykin (1996 pp.565), when an exponential window is used, the correlation matrix and the cross correlation vector can be rewritten as follows:

$$\mathbf{H}_q(k) = \lambda\mathbf{H}_q(k-1) + {}_q\mathbf{a}(k)\,{}_q\mathbf{a}^T(k) \qquad (8 - 72)$$

$$\mathbf{v}_q(k) = \lambda\mathbf{v}_q(k-1) + {}_q\mathbf{a}(k)d(k) \qquad (8 - 73)$$

where λ is the exponential window weighting factor. Typically, λ is a constant less than 1 but greater than 0. A typical value is 0.95. By combining these equations with the derivation of the time-update correlation matrix, we can obtain the time-update correlation matrix with exponential window

$$\begin{bmatrix} \mathbf{H}_q(k) & \mathbf{U}_q^{(q)}(k) & \mathbf{v}_q(k) \\ \mathbf{U}_q^{(q)^T}(k) & \underline{\mathbf{H}}_{\underline{q}}(k) & \underline{\mathbf{v}}_{\underline{q}}(k) \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{H}_q(k-1) & \mathbf{U}_q^{(q)}(k-1) & \mathbf{v}_q(k-1) \\ \mathbf{U}_q^{(q)^T}(k-1) & \underline{\mathbf{H}}_{\underline{q}}(k-1) & \underline{\mathbf{v}}_{\underline{q}}(k-1) \end{bmatrix} + \begin{bmatrix} {}_q\mathbf{a}(k) \\ {}_{\underline{q}}\underline{\mathbf{a}}(k) \end{bmatrix} \begin{bmatrix} {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & d(k) \end{bmatrix} . \qquad (8 - 74)$$

By incorporating the exponential window into the time-update correlation matrix, we will also need to use exponential windowing in the time-update algorithm (RLS algorithm). This is also discussed in Haykin (1996), and we will only provide the result here.

Time-Update: RLS Algorithm with Exponential Windowing

$$\kappa_q(k) = 1 + \lambda^{-1}\,{}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)$$

$$\mathbf{k}_q(k) = \frac{\lambda^{-1}\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)}{1 + \lambda^{-1}\,{}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)} = \lambda^{-1}\kappa_q^{-1}(k)\overline{\mathbf{H}}_q(k-1)\,{}_q\mathbf{a}(k)$$

$$\xi_q(k) = d(k) - {}_q\mathbf{a}^T(k)\mathbf{x}_q(k-1)$$

$$\mathbf{x}_q(k) = \mathbf{x}_q(k-1) + \mathbf{k}_q(k)\xi_q(k)$$

$$\overline{\mathbf{H}}_q(k) = \lambda^{-1}\overline{\mathbf{H}}_q(k-1) - \lambda^{-1}\mathbf{k}_q(k)\,{}_q\mathbf{a}^T(k)\overline{\mathbf{H}}_q(k-1)$$

$$SSE_q(k) = SSE_q(k-1) + \xi_q^2(k)\kappa_q^{-1}(k)$$

8.6.2 Reduce Computational Time

Because the RBF network is localized in space, the output of ${}_q\mathbf{a}(k)$ and ${}_{\underline{q}-}\mathbf{a}(k)$ will contain many elements near zero. These elements are near zero because their centers are far fro the current input. For instance, Figure 8 - 2 illustrates a system trajectory. The circles (filled and unfilled) represent the potential and selected nodes. We can see that near the point X, the majority of the nodes (open circles) have output near zero, and only a small number of nodes (filled circles) have non-zero output.

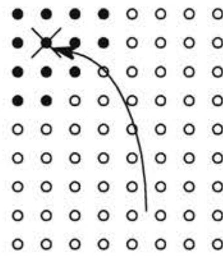


Figure 8 - 2 System Trajectory Travels over the RBF Nodes Planted in 2-D Spaces

If we assume that nodes that are far away from the current input are not contributing much to the output, then we can assume those nodes are not candidates for addition or deletion. In other words, we will only consider potential nodes that have non-zero output as candidates for order-increase-update and will only consider selected nodes that have non-zero output as candidates for order-decrease-update.

To implement this scheme, all we have to do is to find those ${}_{q}\mathbf{a}(k)$ and ${}_{\underline{q}-}\mathbf{a}(k)$ outputs that are near zero and eliminate them from consideration in the order-update. With this implementation, we can save a tremendous amount of computation.

8.7 Summary

In this chapter, we have explored several improvements to the RLS-AWS algorithm. These improvements have resulted in reduced storage, better subset selection (using recursive Efroymson) and reduced computational time for the RLS-AWS algorithm. We will test these improvements on several system identification problems in chapter 10. In the next chapter, we will look for improvements in the QR-RLS-AWS algorithm.

Chapter 9

QR-RLS-AWS Algorithm Improvement

9.1 Introduction 172
9.2 Square Root Error Reduction Term 172
9.3 The New QR-RLS Structure 176
9.4 Recursive QR-Order-Update Algorithms 179
9.4.1 Recursive QR-Order-Increase-Update 180
9.4.2 Recursive QR-Order-Decrease-Update 182
9.5 Recursive Subset Selection Algorithms 186
9.5.1 Recursive QR Forward Selection Method 186
9.5.2 Recursive QR Backward Elimination Method 188
9.5.3 Recursive QR Efroymson Algorithm 190
9.6 Implementation Considerations 194
9.6.1 Exponential Windowing 195
9.6.2 Reduce Computational Time 195
9.7 Summary 196

In this chapter, we improve the QR-RLS-AWS method that we developed in chapter 7. This improvement has lead to a better implementation framework that utilizes the Givens QR algorithm. Using this technique, we have developed a QR version of the recursive Efroymson algorithm. Also, we propose several improvements which reduce the computation and include exponential windowing.

9.1 Introduction

In this chapter, we will discuss a new way of implementing the QR-RLS-AWS algorithm. We will start by discussing an interesting property that we obtain after we have performed the QR time-update algorithm. Then, this key property is linked to the explanation of the square root error reduction term. With this discovery, we can calculate the QR-order-increase-update (discussed in section 9.4) and QR-order-decrease-update (discussed in section 9.4.2) without recomputing the whole orthogonal least squares solution. These order-update technique can be incorporated into the forward selection method and the backward elimination method. Later, by combining both subset selection methods, we form the recursive QR-Efroymson algorithm.

9.2 Square Root Error Reduction Term

Before we explain the square root error reduction term, let us recall the post-array matrix of the QR-RLS algorithm from chapter 5. From Figure 5 - 1, the post-array matrix is given by

$$\begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix} . \tag{9 - 1}$$

This post-array is the direct result of the QR-time-update on the pre-array as explained in chapter 5. To ease the explanation, we will concentrate on the top two terms of the matrix in Eq. (9 - 1)

$$\left[\mathbf{R}(k)\ \mathbf{g}(k)\right] \tag{9 - 2}$$

where $\mathbf{R}(k)$ is the upper triangular squared matrix and $\mathbf{g}(k)$ is the gain vector (Haykin, 1996). These $\mathbf{R}(k)$ and $\mathbf{g}(k)$ provide the batch orthogonal least squares solution to the following linear model

$$\mathbf{d}(k) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{e}(k) \tag{9 - 3}$$

where we consider

$$\mathbf{A}(k) = \mathbf{Q}(k)\mathbf{R}(k)\text{, and} \tag{9 - 4}$$

$$\mathbf{g}(k) = \mathbf{Q}^T(k)\mathbf{d}(k) = \mathbf{R}(k)\mathbf{x}(k)\,. \tag{9 - 5}$$

With this solution, the sum of squared error can be rewritten as

$$\begin{aligned} \mathbf{e}^T(k)\mathbf{e}(k) &= ((\mathbf{d}(k) - \mathbf{A}(k)\mathbf{x}(k))^T(\mathbf{d}(k) - \mathbf{A}(k)\mathbf{x}(k))) \\ &= (\mathbf{d}(k) - \mathbf{Q}(k)\mathbf{g}(k))^T(\mathbf{d}(k) - \mathbf{Q}(k)\mathbf{g}(k)) \\ &= \mathbf{d}^T(k)\mathbf{d}(k) - \underbrace{\mathbf{d}^T(k)\mathbf{Q}(k)}_{\mathbf{g}^T(k)}\mathbf{g}(k) - \mathbf{g}^T(k)\underbrace{\mathbf{Q}^T(k)\mathbf{d}(k)}_{\mathbf{g}(k)} + \mathbf{g}^T(k)\underbrace{\mathbf{Q}^T(k)\mathbf{Q}(k)}_{\mathbf{I}}\mathbf{g}(k) \end{aligned} \tag{9 - 6}$$

and we obtain

$$\mathbf{e}^T(k)\mathbf{e}(k) = \mathbf{d}^T(k)\mathbf{d}(k) - \mathbf{g}^T(k)\mathbf{g}(k)\,. \tag{9 - 7}$$

Assume that we have $q+1$ nodes in the model, then

$$\mathbf{e}^T_{q+1}(k)\mathbf{e}_{q+1}(k) = \mathbf{d}^T(k)\mathbf{d}(k) - g_1^2(k) - g_2^2(k)\ldots - g_q^2(k) - g_{q+1}^2(k)\,. \tag{9 - 8}$$

Similarly, if we assume that we have q nodes in the model, then the sum of squared error for the model that excludes the $q+1$ node is

$$\mathbf{e}^T_q(k)\mathbf{e}_q(k) = \mathbf{d}^T(k)\mathbf{d}(k) - g_1^2(k) - g_2^2(k)\ldots - g_q^2(k)\,. \tag{9 - 9}$$

Therefore, by substituting Eq. (9 - 9) into Eq. (9 - 8), we get

$$\underbrace{\mathbf{e}_{q+1}^T(k)\mathbf{e}_{q+1}(k)}_{SSE_{q+1}(k)} = \underbrace{\mathbf{d}^T(k)\mathbf{d}(k) - g_1^2(k) - g_2^2(k) \ldots - g_q^2(k)}_{SSE_q(k)} - g_{q+1}^2(k) \quad . \qquad (9 - 10)$$

If we compare Eq. (9 - 10) to the recursive sum of squared errors formula in Eq. (6 - 30), we can conclude that $g_{q+1}^2(k)$ is the error reduction term; in other words, $g_{q+1}(k)$ is the square root error reduction term. From Eq. (9 - 10), we can conclude two important observations.

1. Eq. (9 - 10) is a recursive equation, and it applies to any i-th order for i between 1 and q.
2. The i-th error reduction term $g_i^2(k)$ measures how much the sum of squared error has been increased when the i-th node is added into the model that contained node 1 to node i-1; or it measures how much the sum of squared error has been decreased when the i-th node is deleted from the model that contained node 1 to node i.

These observations indicate that if we compute the post-array matrix for a model that contains i nodes, then we can easily obtain the sum of squared errors information for the other $i - 1$ models. These $i - 1$ models are subsets of the model that contains i nodes. For example, if we have a model that has four nodes (shown in Figure 9 - 1), and we have calculated the post-array matrix using Eq. (9 - 2), then, we can obtain the sum of squared errors for the other three models as shown in Figure 9 - 1.

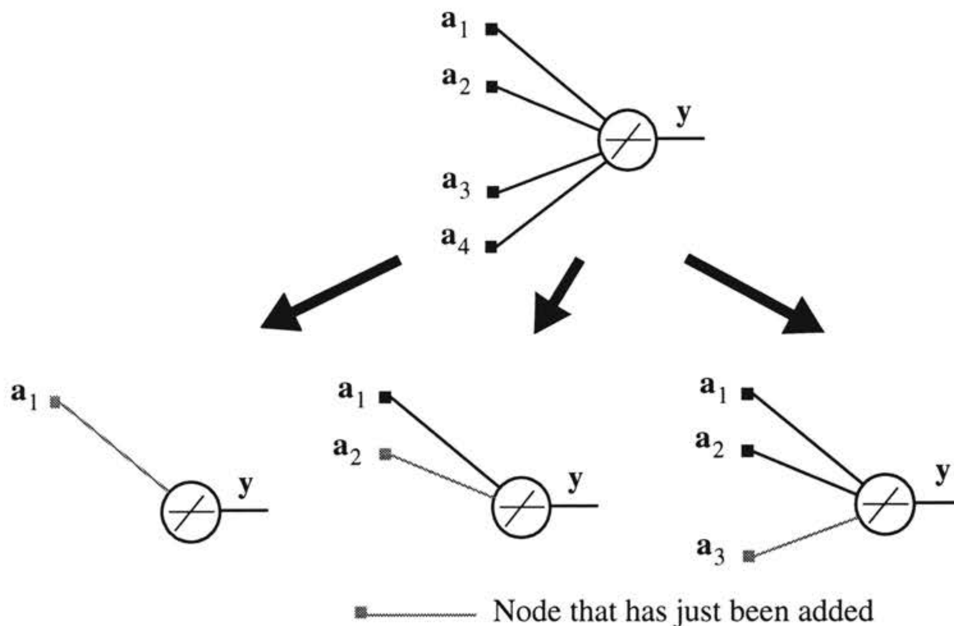


Figure 9 - 1 Obtainable subsets given a four-node linear model

Also, these observations suggest that the ordering of the nodes in a model affects the meaning of $g_i(k)$. For example, if we switch the order of nodes $\mathbf{a}_3$ and $\mathbf{a}_4$ in the four-node linear model shown in Figure 9 - 1, then $g_3(k)$ measures the increase in the sum of squared error when the $\mathbf{a}_4$ node is added into the model that contains node $\mathbf{a}_1$ and $\mathbf{a}_2$; or it measures the decrease in the sum of squared error when the $\mathbf{a}_4$ node is removed from the model that contains node $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_4$.

Now, given the two facts above, suppose we want to find the value of $g_3(k)$ in the example above (the order of nodes $\mathbf{a}_3$ and $\mathbf{a}_4$ in the four-node linear model is switched), how can we proceed to find it without recomputing Eq. (9 - 2)? If we examine Eq. (9 - 2), we will see that each column of $\mathbf{R}(k)$ belongs to a designated node. All we need to do is

swap the columns corresponding to the desired nodes and re-triangulate them by applying the re-orthogonalization process. We will explain these techniques in detail in sections 9.4 and 9.5.

9.3 The New QR-RLS Structure

In section 9.2, we have discovered that we can find the sum of squared errors for several subsets in a model. To use this technique for subset selection, we need to change the QR-RLS algorithm structure so that it contains both selected and potential nodes. Specifically, instead of computing the $\mathbf{R}(k)$ and $\mathbf{g}(k)$ for the selected nodes only, we compute the $\mathbf{R}(k)$ and $\mathbf{g}(k)$ for both selected and potential nodes. The reason we compute the $\mathbf{R}(k)$ and $\mathbf{g}(k)$ for both selected and potential nodes is so that we can think of it as the complete linear model and we would like to find out the sum of squared errors for a particular subset in this model (the model that contains only the selected nodes). These changes are as follows. First, we form the pre-array as shown on the left-hand-side of Eq. (9 - 11).

$$\begin{bmatrix} \mathbf{R}_q(k-1) & \mathbf{S}_q^{(\underline{q})}(k-1) & \mathbf{g}_q(k-1) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k-1) & \underline{\mathbf{g}}_{\underline{q}}(k-1) \\ {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & d(k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(\underline{q})}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k-1) & \underline{\mathbf{g}}_{\underline{q}}(k-1) \\ \mathbf{0}_q^T & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}. \quad (9 - 11)$$

Then, we annihilate the current input data ${}_q\mathbf{a}^T(k)$ one by one from left to right. The Givens QR algorithm described in Chapter 4.5 is used, except that we skip all zero elements contained in $\mathbf{R}_q(k-1)$ and $\mathbf{0}_{\underline{q}\times q}$. The Givens rotation operates by annihilating the input

data one by one until all elements become zero entries, as shown on the right-hand-side of Eq. (9 - 11). As soon as the all the current input data elements ${}_q\mathbf{a}^T(k)$ have been annihilated, we obtain the post-array elements, as in Eq. (9 - 1). At this instance, we will use Eq. (5 - 41) to compute time-update increase in the sum of squared error

$$SSE_q(k) = SSE_q(k-1) + \xi^2(k)\kappa^{-1}(k)\,. \tag{9 - 12}$$

Once we have computed this term, we can continue to annihilate the current potential data ${}_{\underline{q}}\mathbf{a}^T(k)$ until all entries become zero, as shown on the right-hand-side of Eq. (9 - 13).

$$\begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(q)}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k-1) & \underline{\mathbf{g}}_{\underline{q}}(k-1) \\ \mathbf{0}_q^T & {}_{\underline{q}}\mathbf{a}^T(k) & \kappa^{-1/2}(k)\xi(k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(q)}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k) & \underline{\mathbf{g}}_{\underline{q}}(k) \\ \mathbf{0}_q^T & \mathbf{0}_{\underline{q}}^T & \times \end{bmatrix} \tag{9 - 13}$$

The $\times$ term represents the increase in the sum of squared errors from both selected and potential nodes. Typically, we will ignore this value. However, it gives us a good indication of how much error reduction we can have if we were to use all the nodes (selected and potential) in our model. The right-hand-side matrix is the post-array matrix for our new QR-RLS algorithm. In particular, we are interested in the matrix elements excluded in the last row, as shown below.

$$\left[\begin{array}{c|c|c} \mathbf{R}_q(k) & \mathbf{S}_q^{(\underline{q})}(k) & \mathbf{g}_q(k) \\ \hline \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k) & \underline{\mathbf{g}}_{\underline{q}}(k) \end{array}\right] = \left[\begin{array}{cccc|cccc|c} r_1^1 & r_2^1 & \dots & r_q^1 & s_1^1 & \dots & s_{\underline{q}-1}^1 & s_{\underline{q}}^1 & g_1 \\ 0 & r_2^2 & \dots & r_q^2 & s_1^2 & \dots & s_{\underline{q}-1}^2 & s_{\underline{q}}^2 & g_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & r_q^q & s_1^q & \dots & s_{\underline{q}-1}^q & s_{\underline{q}}^q & g_q \\ \hline 0 & 0 & 0 & 0 & r_{\underline{1}}^1 & \dots & r_{\underline{q}-1}^1 & r_{\underline{q}}^1 & \underline{g}_1 \\ \vdots & \vdots & & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \ddots & r_{\underline{q}-1}^{\underline{q}-1} & r_{\underline{q}}^{\underline{q}-1} & \underline{g}_{\underline{q}-1} \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & r_{\underline{q}}^{\underline{q}} & \underline{g}_{\underline{q}} \end{array}\right] \qquad (9 - 14)$$

The $\mathbf{R}_q(k)$ and the $\underline{\mathbf{R}}_{\underline{q}}(k)$ are the R-factors (triangular matrices) for selected nodes and potential nodes. They contain $q(q+1)/2$ and $\underline{q}(\underline{q}+1)/2$ elements, respectively. $\mathbf{S}_q^{(\underline{q})}(k)$ contains the R-factor cross terms between the selected nodes and the potential nodes. It contains $q \times \underline{q}$ elements. Meanwhile, $\mathbf{g}_q(k)$ and $\underline{\mathbf{g}}_{\underline{q}}(k)$ are the gain vectors (or the square root error reduction vectors) for the selected and potential nodes. The total number of elements is $(q+\underline{q})(q+\underline{q}+3)/2$. In terms of floating operation, we will require ~ $O(3n^2+9n)$ flops to obtain the post-array in Eq. (9 - 13).

In the following, we will introduce an example of the post-array matrix that contains four selected nodes $q = 4$ and five potential nodes $\underline{q} = 5$. This example will be used to explain the QR-order-increase-update and QR-order-decrease-update.

$$\begin{bmatrix} \mathbf{R}_4(k) & \mathbf{S}_4^{(5)}(k) & \mathbf{g}_4(k) \\ \mathbf{0}_{5\times 4} & \underline{\mathbf{R}}_5(k) & \underline{\mathbf{g}}_5(k) \end{bmatrix} = \begin{bmatrix} r_1^1 & r_2^1 & r_3^1 & r_4^1 & s_1^1 & s_2^1 & s_3^1 & s_4^1 & s_5^1 & g_1 \\ 0 & r_2^2 & r_3^2 & r_4^2 & s_1^2 & s_2^2 & s_3^2 & s_4^2 & s_5^2 & g_2 \\ 0 & 0 & r_3^3 & r_4^3 & s_1^3 & s_2^3 & s_3^3 & s_4^3 & s_5^3 & g_3 \\ 0 & 0 & 0 & r_4^4 & s_1^4 & s_2^4 & s_3^4 & s_4^4 & s_5^4 & g_4 \\ 0 & 0 & 0 & 0 & \underline{r}_1^1 & \underline{r}_2^1 & \underline{r}_3^1 & \underline{r}_4^1 & \underline{r}_5^1 & \underline{g}_1 \\ 0 & 0 & 0 & 0 & 0 & \underline{r}_2^2 & \underline{r}_3^2 & \underline{r}_4^2 & \underline{r}_5^2 & \underline{g}_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_3^3 & \underline{r}_4^3 & \underline{r}_5^3 & \underline{g}_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_4^4 & \underline{r}_5^4 & \underline{g}_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_5^5 & \underline{g}_5 \end{bmatrix} \quad (9 - 15)$$

In the next section, we will use this new result to form the recursive QR-order-increase-update and the recursive QR-order-decrease-update algorithms.

9.4 Recursive QR-Order-Update Algorithms

In this section, the result of the new QR-RLS algorithm, the post-array matrix given by Eq. (9 - 14), is computed and is readily available. By applying the results in section 9.2 to this new QR-RLS algorithm, we obtain two recursive order-update algorithms called the recursive QR-order-increase-update and the recursive QR-order-decrease-update algorithms. The recursive QR-order-increase-update allows the addition of a potential node into the model and the recursive QR-order-decrease-update allows the removal of a selected node from the model. Both algorithms are obtained without recomputing the whole orthogonal least squares solution.

9.4.1 Recursive QR-Order-Increase-Update

In the recursive QR-order-increase-update, the objective is to add an ith potential node to the model containing q nodes, without recomputing the whole orthogonal least squares solution. By using the result discussed in section 9.2, we can achieve this objective by doing the following.

1. In Eq. (9 - 14), move the ith column of potential nodes to the $q+1$ column of the selected nodes.
2. Re-triangulate the matrix obtain in (1) so that it becomes an upper triangular matrix. This is done by applying the Givens rotation successively. This process is also called re-orthogonalization. Due to the structure of the matrix obtain in (1), we will only need to apply the Givens rotation to the following matrix

$$\begin{bmatrix} \underline{r}_{i}^{1} & \underline{r}_{1}^{1} & \underline{r}_{2}^{1} & \cdots & \underline{r}_{i-1}^{1} & \underline{r}_{i+1}^{1} & \cdots & \underline{r}_{q}^{1} & \underline{g}_{1} \\ \underline{r}_{i}^{2} & 0 & \underline{r}_{2}^{2} & \cdots & \underline{r}_{i-1}^{2} & \underline{r}_{i+1}^{2} & \cdots & \underline{r}_{q}^{2} & \underline{g}_{2} \\ & & & & & & & & \\ \underline{r}_{i}^{i} & 0 & \cdots & 0 & \underline{r}_{i-1}^{i} & \underline{r}_{i+1}^{i} & \cdots & \underline{r}_{q}^{i} & \underline{g}_{i} \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \cdots & \times & \cdots & \times \\ 0 & \times & \cdots & \times & \cdots & \times \\ & & & & & \\ 0 & 0 & \cdots & \times & \cdots & \times \end{bmatrix} \leftarrow \underline{g}_{q+1} \quad . \qquad (9 - 16)$$

Once we finish the re-orthogonalization process, the upper right-hand term contains g_{q+1}. By squaring this term, we can find out the reduction in the sum of squared error.

$$SSE_{q+1}(k) = SSE_{q}(k) - \underline{g}_{q+1}^{2}(k) \qquad (9 - 17)$$

Keep in mind that the above re-orthogonalization process calculates the error reduction and also generates the next iteration. If we only want to find the square root error reduction term

g_{q+1}, then we only need to apply the Givens rotation to the first and last column of Eq. (9 - 16).

$$\begin{bmatrix} \underline{r}_i^1 & \underline{g}_1 \\ \underline{r}_i^2 & \underline{g}_2 \\ \vdots & \vdots \\ \underline{r}_i^i & \underline{g}_i \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times \\ 0 & \times \\ \vdots & \vdots \\ 0 & \times \end{bmatrix} \leftarrow \underline{g}_{q+1} \tag{9 - 18}$$

The above computation only requires $6i$ flops.

Example 9 - 1

Consider a model that consists of four selected nodes $q = 4$ and five potential nodes $\underline{q} = 5$, where the post-array matrix is given by Eq. (9 - 15). Suppose we want to add the third node of the potential nodes to the model, then the post-array matrix will be

$$\begin{bmatrix} r_1^1 & r_2^1 & r_3^1 & r_4^1 & s_3^1 & s_1^1 & s_2^1 & s_4^1 & s_5^1 & g_1 \\ 0 & r_2^2 & r_3^2 & r_4^2 & s_3^2 & s_1^2 & s_2^2 & s_4^2 & s_5^2 & g_2 \\ 0 & 0 & r_3^3 & r_4^3 & s_3^3 & s_1^3 & s_2^3 & s_4^3 & s_5^3 & g_3 \\ 0 & 0 & 0 & r_4^4 & s_3^4 & s_1^4 & s_2^4 & s_4^4 & s_5^4 & g_4 \\ 0 & 0 & 0 & 0 & \underline{r}_3^1 & \underline{r}_1^1 & \underline{r}_2^1 & \underline{r}_4^1 & \underline{r}_5^1 & \underline{g}_1 \\ 0 & 0 & 0 & 0 & \underline{r}_3^2 & 0 & \underline{r}_2^2 & \underline{r}_4^2 & \underline{r}_5^2 & \underline{g}_2 \\ 0 & 0 & 0 & 0 & \underline{r}_3^3 & 0 & 0 & \underline{r}_4^3 & \underline{r}_5^3 & \underline{g}_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_4^4 & \underline{r}_5^4 & \underline{g}_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_5^5 & \underline{g}_5 \end{bmatrix} . \tag{9 - 19}$$

Next, we will apply the Givens rotation to the highlighted post-array matrix elements in Eq. (9 - 19). By applying the Givens rotation successively, we transform Eq. (9 - 19) to an upper

triangular matrix. Note that the highlighted terms are the terms involved in the Givens rotations.

$$\begin{bmatrix} r_{-3}^1 & r_{-1}^1 & r_{-2}^1 & r_{-4}^1 & r_{-5}^1 & g_1 \\ r_{-3}^2 & 0 & r_{-2}^2 & r_{-4}^2 & r_{-5}^2 & g_2 \\ r_{-3}^3 & 0 & 0 & r_{-4}^3 & r_{-5}^3 & g_3 \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & 0 & \times & \times & \times & \times \\ \times & 0 & 0 & \times & \times & \times \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \end{bmatrix} \qquad (9 - 20)$$

As explained in Eq. (9 - 18), we can perform Givens rotations on the first and last columns of Eq. (9 - 20).

$$\begin{bmatrix} r_{-3}^1 & g_1 \\ r_{-3}^2 & g_2 \\ r_{-3}^3 & g_3 \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times \\ \times & \times \\ 0 & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times \\ 0 & \times \\ 0 & \times \end{bmatrix} \leftarrow g_5 \quad . \qquad (9 - 21)$$

9.4.2 Recursive QR-Order-Decrease-Update

In the recursive QR-order-decrease-update, the objective is to remove the jth node from a model that contains q nodes, without recomputing the whole orthogonal least squares solution. By using the results discussed in section 9.2, we can achieve this objective by doing the following.

1. In Eq. (9 - 14), move the jth column of selected nodes to the first column of the potential nodes.

2. Apply the re-orthogonalization process to the matrix obtained in (1) until it becomes an upper triangular matrix. Due to the structure of the matrix obtain in (1), we will only need to apply the Givens rotation to the following matrix

$$\begin{bmatrix} r^j_{j+1} & r^j_{j+2} & \dots & r^j_{q-1} & r^j_q & r^j_j & s^j_1 & \dots & s^j_q & g_j \\ r^{j+1}_{j+1} & r^{j+1}_{j+2} & \dots & r^{j+1}_{q-1} & r^{j+1}_q & 0 & s^{j+1}_1 & \dots & s^{j+1}_q & g_{j+1} \\ 0 & r^{j+2}_{j+2} & \dots & r^{j+2}_{q-1} & r^{j+2}_q & 0 & s^{j+2}_1 & \dots & s^{j+2}_q & g_{j+2} \\ & & & & & & & & & \\ 0 & \dots & 0 & r^{q-1}_{q-1} & r^{q-1}_q & 0 & s^{q-1}_1 & \dots & s^{q-1}_q & g_{q-1} \\ 0 & \dots & 0 & 0 & r^q_q & 0 & s^q_1 & \dots & s^q_q & g_q \end{bmatrix}. \quad (9 - 22)$$

$$\rightarrow \begin{bmatrix} \times & \times & \dots & \times & \dots & \times \\ 0 & \times & \dots & \times & \dots & \times \\ & & & & & \\ 0 & 0 & \dots & \times & \dots & \times \end{bmatrix} \leftarrow g_q$$

Once we finish the re-orthogonalization process, the lower right-hand term contains g_q. By squaring this term, we can find out the reduction in the sum of squared error.

$$SSE_{q-1}(k) = SSE_q(k) + g_q^2(k) \quad (9 - 23)$$

The above re-orthogonalization process finds the error reduction terms and also calculates the extra cross terms so that the algorithm can restart for the next iteration. If we only need to find the square root error reduction term g_q, then we only need to apply the Givens rotation to the following subset of the matrix in Eq. (9 - 22)

$$
\begin{bmatrix}
r_{j+1}^{j} & r_{j+2}^{j} & \cdots & r_{q-1}^{j} & r_{q}^{j} & g_j \\
r_{j+1}^{j+1} & r_{j+2}^{j+1} & \cdots & r_{q-1}^{j+1} & r_{q}^{j+1} & g_{j+1} \\
0 & r_{j+2}^{j+2} & \cdots & r_{q-1}^{j+2} & r_{q}^{j+2} & g_{j+2} \\
 & & & & & \\
0 & \cdots & 0 & r_{q-1}^{q-1} & r_{q}^{q-1} & g_{q-1} \\
0 & \cdots & 0 & 0 & r_{q}^{q} & g_q
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\times & \times & \cdots & \times \\
0 & \times & \cdots & \times \\
 & & & \\
0 & 0 & \cdots & \times
\end{bmatrix}
\leftarrow g_q
\quad . \tag{9 - 24}
$$

The above computation only requires $3j(j+1)$ flops.

Example 9 - 2

Consider the model we used in Example 9 - 1. Suppose we want to delete the second node from the model, then the post-array matrix will be

$$
\begin{bmatrix}
r_1^1 & r_3^1 & r_4^1 & r_2^1 & s_1^1 & s_2^1 & s_3^1 & s_4^1 & s_5^1 & g_1 \\
0 & r_3^2 & r_4^2 & r_2^2 & s_1^2 & s_2^2 & s_3^2 & s_4^2 & s_5^2 & g_2 \\
0 & r_3^3 & r_4^3 & 0 & s_1^3 & s_2^3 & s_3^3 & s_4^3 & s_5^3 & g_3 \\
0 & 0 & r_4^4 & 0 & s_1^4 & s_2^4 & s_3^4 & s_4^4 & s_5^4 & g_4 \\
0 & 0 & 0 & 0 & \underline{r}_1^1 & \underline{r}_2^1 & \underline{r}_3^1 & \underline{r}_4^1 & \underline{r}_5^1 & \underline{g}_1 \\
0 & 0 & 0 & 0 & 0 & \underline{r}_2^2 & \underline{r}_3^2 & \underline{r}_4^2 & \underline{r}_5^2 & \underline{g}_2 \\
0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_3^3 & \underline{r}_4^3 & \underline{r}_5^3 & \underline{g}_3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_4^4 & \underline{r}_5^4 & \underline{g}_4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{r}_5^5 & \underline{g}_5
\end{bmatrix} . \tag{9 - 25}
$$

Successive Givens rotations are applied to the highlighted post-array matrix elements in Eq. (9 - 25). A step by step Givens rotation operation is shown in the following.

$$\begin{bmatrix} r_3^2 & r_4^2 & r_2^2 & s_1^2 & s_2^2 & s_3^2 & s_4^2 & s_5^2 & g_2 \\ r_3^3 & r_4^3 & 0 & s_1^3 & s_2^3 & s_3^3 & s_4^3 & s_5^3 & g_3 \\ 0 & r_4^4 & 0 & s_1^4 & s_2^4 & s_3^4 & s_4^4 & s_5^4 & g_4 \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & 0 & \times & \times & \times & \times & \times & \times \\ 0 & \times & 0 & \times & \times & \times & \times & \times & \times \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & 0 & \times & \times & \times & \times & \times & \times \end{bmatrix} \rightarrow \tag{9 - 26}$$

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times & \times & \times & \times \end{bmatrix}$$

Again, if we only want the error reduction term g_q, then we only need to apply the Givens rotation to Eq. (9 - 24) as follows

$$\begin{bmatrix} r_3^2 & r_4^2 & g_2 \\ r_3^3 & r_4^3 & g_3 \\ 0 & r_4^4 & g_4 \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix} \tag{9 - 27}$$

$g_q(k)$

9.5 Recursive Subset Selection Algorithms

In this section, we will use the QR-order-update techniques to develop several recursive subset selection algorithms. Specifically, we will extend the results of the QR-order-increase-update to develop the recursive QR forward selection method; and we will extend the results of the QR-order-decrease-update to develop the recursive QR backward elimination method. Then, we will combine both methods to develop the recursive Efroymson algorithm.

9.5.1 Recursive QR Forward Selection Method

In the recursive QR forward selection method, the recursive order-increase-update is used to compute the square root error reduction term for all the potential nodes. Since we are only interested in the square root error reduction terms, Eq. (9 - 18) is used. Once we calculate all the square root error reduction terms, the best potential node, which yields the largest sum of squared error reduction, is picked. The computation required to calculate all of the square root error reduction terms is ~ $6\underline{q}^2$. Keep in mind that Eq. (9 - 18) cannot be used to restart the next iteration, we will need to update the post-array matrix using Eq. (9 - 16) once we find out which potential node we need to add. Once this post-array matrix is updated, we can also use it to compute the parameters by solving a triangular system using back-substitution. The complete algorithm is given below.

Recursive QR Forward Selection Algorithm

Initialized $\underline{g}_1 = \underline{g}_{q+1}^{(1)}$.

For $i = 2 \ldots \underline{q}$, *compute*

$$\begin{bmatrix} \underline{r}_{\tilde{i}}^1 & \underline{g}_1 \\ \underline{r}_{\tilde{i}}^2 & \underline{g}_2 \\ \vdots & \vdots \\ \underline{r}_{\tilde{i}}^i & \underline{g}_i \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times \\ 0 & \times \\ \vdots & \vdots \\ 0 & \times \end{bmatrix} \leftarrow \underline{g}_{q+1}^{(i)} \qquad (9 - 28)$$

end

$$g_{q+1}(k) = max[\underline{g}_{q+1}^{(i)}] \quad , \tilde{i} = index(max[\underline{g}_{q+1}^{(i)}]),$$

$$SSE_{q+1}(k) = SSE_q(k) - g_{q+1}^2(k)$$

Update Parameters

$$\begin{bmatrix} \underline{r}_{\tilde{i}}^1 & \underline{r}_1^1 & \underline{r}_2^1 & \ldots & \underline{r}_{\tilde{i}-1}^1 & \underline{r}_{\tilde{i}+1}^1 & \ldots & \underline{r}_{\underline{q}}^1 & \underline{g}_1 \\ \underline{r}_{\tilde{i}}^2 & 0 & \underline{r}_2^2 & \ldots & \underline{r}_{\tilde{i}-1}^2 & \underline{r}_{\tilde{i}+1}^2 & \ldots & \underline{r}_{\underline{q}}^2 & \underline{g}_2 \\ \\ \underline{r}_{\tilde{i}}^{\tilde{i}} & 0 & \ldots & 0 & \underline{r}_{\tilde{i}-1}^{\tilde{i}} & \underline{r}_{\tilde{i}+1}^{\tilde{i}} & \ldots & \underline{r}_{\underline{q}}^i & \underline{g}_i \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \ldots & \times & \ldots & \times \\ 0 & \times & \ldots & \times & \ldots & \times \\ \\ 0 & 0 & \ldots & \times & \ldots & \times \end{bmatrix} \leftarrow \underline{g}_{q+1} \qquad (9 - 29)$$

9.5.2 Recursive QR Backward Elimination Method

As with recursive backward elimination in chapter 8, the recursive QR backward elimination proposed here is used in conjunction with recursive forward selection only. In recursive QR backward elimination, the recursive order-decrease-update is used to compute the square root error reduction term for all the nodes in the model. The idea is to examine the nodes one by one and to remove the least contributing node. The least contributing node produces the smallest increase in the sum of squared error. Since we are only interested in obtaining the square root error reduction terms, Eq. (9 - 24) is used. Once we calculate all the square root error reduction terms, the node that yields the smallest sum of squared error increase is picked. The computation required to compute all the square root error reduction terms is $q(q+1)^2 \sim O(q^3)$. Keep in mind that Eq. (9 - 24) cannot be used to generate the next iteration because we did not orthogonalize the cross terms. So, we will need to update the post-array matrix using Eq. (9 - 22). Again, once this post-array matrix is updated, we can also use it to compute the parameters by solving a triangular system using back-substitution method. The complete algorithm is given below.

Recursive Backward Elimination Algorithm

For $j = 1 \ldots \underline{q} - 1$, *compute*

$$\begin{bmatrix} r_{j+1}^{j} & r_{j+2}^{j} & \cdots & r_{q-1}^{j} & r_{q}^{j} & g_j \\ r_{j+1}^{j+1} & r_{j+2}^{j+1} & \cdots & r_{q-1}^{j+1} & r_{q}^{j+1} & g_{j+1} \\ 0 & r_{j+2}^{j+2} & \cdots & r_{q-1}^{j+2} & r_{q}^{j+2} & g_{j+2} \\ & & & & & \\ 0 & \cdots & 0 & r_{q-1}^{q-1} & r_{q}^{q-1} & g_{q-1} \\ 0 & \cdots & 0 & 0 & r_{q}^{q} & g_q \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ & & & \\ 0 & 0 & \cdots & \times \end{bmatrix} \leftarrow g_q^{(j)}$$

$$g_q(k) = min[\underline{g}_q^{(j)}] \quad , \quad \tilde{j} = index(min[\underline{g}_q^{(j)}])$$

$$SSE_{q-1}(k) = SSE_q(k) + g_q^2(k)$$

Update Parameters

$$\begin{bmatrix} r_{j+1}^{\tilde{j}} & r_{j+2}^{\tilde{j}} & \cdots & r_{q-1}^{\tilde{j}} & r_{q}^{\tilde{j}} & r_{\tilde{j}}^{j} & s_1^{\tilde{j}} & \cdots & s_{\underline{q}}^{\tilde{j}} & g_{\tilde{j}} \\ r_{j+1}^{\tilde{j}+1} & r_{j+2}^{\tilde{j}+1} & \cdots & r_{q-1}^{\tilde{j}+1} & r_{q}^{\tilde{j}+1} & 0 & s_1^{\tilde{j}+1} & \cdots & s_{\underline{q}}^{\tilde{j}+1} & g_{\tilde{j}+1} \\ 0 & r_{j+2}^{\tilde{j}+2} & \cdots & r_{q-1}^{\tilde{j}+2} & r_{q}^{\tilde{j}+2} & 0 & s_1^{\tilde{j}+2} & \cdots & s_{\underline{q}}^{\tilde{j}+2} & g_{\tilde{j}+2} \\ & & & & & & & & & \\ 0 & \cdots & 0 & r_{q-1}^{q-1} & r_{q}^{q-1} & 0 & s_1^{q-1} & \cdots & s_{\underline{q}}^{q-1} & g_{q-1} \\ 0 & \cdots & 0 & 0 & r_{q}^{q} & 0 & s_1^{q} & \cdots & s_{\underline{q}}^{q} & g_q \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \cdots & \times & \cdots & \times \\ 0 & \times & \cdots & \times & \cdots & \times \\ & & & & & \\ 0 & 0 & \cdots & \times & \cdots & \times \end{bmatrix}$$

9.5.3 Recursive QR Efroymson Algorithm

As with the recursive Efroymson algorithm discussed in chapter 8, the recursive QR Efroymson algorithm works by combining the recursive QR forward selection method and the recursive QR backward elimination method. The same stopping rule used by the Recursive Efroymson algorithm discussed in chapter 8 are used here. Thus, by combining the recursive QR forward selection and the recursive QR backward elimination, we obtain the recursive QR Efroymson algorithm. The complete algorithm is summarized below.

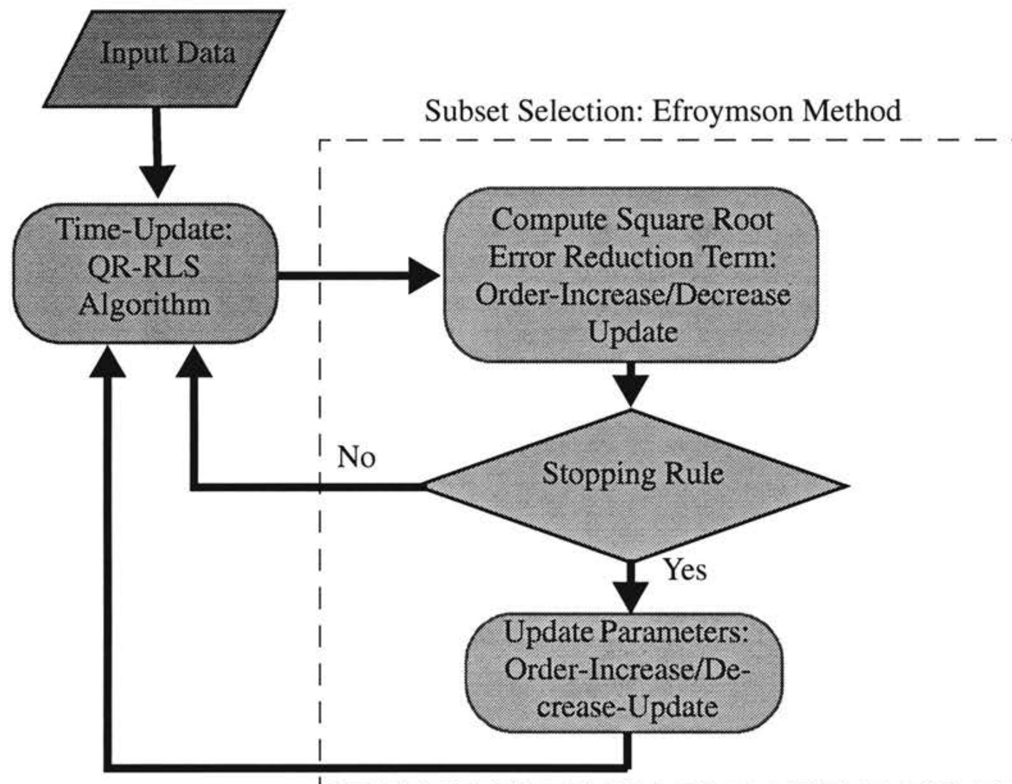


Figure 9 - 2 Flow Chart for QR-RLS-AWS with Efroymson Method

Recursive QR-Efroymson Algorithm

Initialization:

$q = 0,\ \mathbf{R}_q(0) = \mathbf{0},\ \mathbf{S}_q^{(q)}(0) = \mathbf{0},\ \underline{\mathbf{R}}_{\underline{q}}(0) = \mathbf{0},\ \underline{\mathbf{g}}_{\underline{q}}(0) = \mathbf{0}$ *and* $\mathbf{g}_q(0) = \mathbf{0}$

Input Data:

$\{{}_q\mathbf{a}(k), d(k), {}_{(q)}\underline{\mathbf{a}}(k)\}$

Time-Update: QR-RLS Algorithm

Solve using Givens Rotation

$$\begin{bmatrix} \mathbf{R}_q(k-1) & \mathbf{S}_q^{(q)}(k-1) & \mathbf{g}_q(k-1) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k-1) & \underline{\mathbf{g}}_{\underline{q}}(k-1) \\ {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & d(k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(q)}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k-1) & \underline{\mathbf{g}}_{\underline{q}}(k-1) \\ \mathbf{0}_q^T & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & \kappa^{-1/2}(k)\xi(k) \end{bmatrix}$$

$$SSE_q(k) = SSE_q(k-1) + \kappa^{-1}(k)\xi^2(k)$$

Continue to solve using Givens Rotation

$$\begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(q)}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k-1) & \underline{\mathbf{g}}_{\underline{q}}(k-1) \\ \mathbf{0}_q^T & {}_{\underline{q}}\underline{\mathbf{a}}^T(k) & \kappa^{-1/2}(k)\xi(k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(q)}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{\underline{q}\times q} & \underline{\mathbf{R}}_{\underline{q}}(k) & \underline{\mathbf{g}}_{\underline{q}}(k) \\ \mathbf{0}_q^T & \mathbf{0}_{\underline{q}}^T & \times \end{bmatrix}$$

Compute Square Root Error Reduction Terms

For QR-Order-Increase-Update:

For $1 \le i \le q$

$$\begin{bmatrix} \underline{r}_i^1 & \underline{g}_1 \\ \underline{r}_i^2 & \underline{g}_2 \\ \vdots & \vdots \\ \underline{r}_i^i & \underline{g}_i \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times \\ 0 & \times \\ \vdots & \vdots \\ 0 & \times \end{bmatrix} \leftarrow g_{q+1}^{(i)}$$

end

For QR-Order-Decrease-Update:

For $1 \le j \le q$

$$\begin{bmatrix} r_{j+1}^j & r_{j+2}^j & \dots & r_{q-1}^j & r_q^j & g_j \\ r_{j+1}^{j+1} & r_{j+2}^{j+1} & \dots & r_{q-1}^{j+1} & r_q^{j+1} & g_{j+1} \\ 0 & r_{j+2}^{j+2} & \dots & r_{q-1}^{j+2} & r_q^{j+2} & g_{j+2} \\ & & & & & \\ 0 & \dots & 0 & r_{q-1}^{q-1} & r_q^{q-1} & g_{q-1} \\ 0 & \dots & 0 & 0 & r_q^q & g_q \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \dots & \times \\ 0 & \times & \dots & \times \\ & & & \\ 0 & 0 & \dots & \times \end{bmatrix} \leftarrow g_q^{(j)}(k)$$

end

Stopping Rules:

For Order-Increase-Update

$$g_{q+1}(k) = max[g_{q+1}^{(i)}] \quad , \tilde{i} = index(max[g_{q+1}^{(i)}])$$

$$SSE_{q+1}(k) = SSE_q(k) - g_{q+1}^2(k) \quad , \quad W_e = \frac{g_{q+1}^2(k)}{SSE_{q+1}^{(i)}(k)/(k-q-2)}$$

If $W_e > F_e$, *update parameters*

For Order-Decrease-Update

$$g_q(k) = min[g_q^{(j)}] \qquad , \quad \tilde{j} = index(min[g_q^{(j)}])$$

$$SSE_{q-1}(k) = SSE_q(k) + g_q^2(k), \quad W_d = \frac{g_q^2(k)}{SSE_q(k)/(k-q-1)} \quad .$$

If $W_d < F_d$, *update parameters*

Update Parameter:

For Order-Increase-Update

$$\begin{bmatrix} \underline{r}_i^1 & \underline{r}_1^1 & \underline{r}_2^1 & \cdots & \underline{r}_{i-1}^1 & \underline{r}_{i+1}^1 & \cdots & \underline{r}_q^1 & \underline{g}_1 \\ \underline{r}_i^2 & 0 & \underline{r}_2^2 & \cdots & \underline{r}_{i-1}^2 & \underline{r}_{i+1}^2 & \cdots & \underline{r}_q^2 & \underline{g}_2 \\ & & & & & & & & \\ \underline{r}_i^i & 0 & \cdots & 0 & \underline{r}_{i-1}^i & \underline{r}_{i+1}^i & \cdots & \underline{r}_q^i & \underline{g}_i \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \cdots & \times & \cdots & \times \\ 0 & \times & \cdots & \times & \cdots & \times \\ & & & & & \\ 0 & 0 & \cdots & \times & \cdots & \times \end{bmatrix} \leftarrow g_{q+1}$$

For Order-Decrease-Update

$$
\begin{bmatrix}
r^j_{j+1} & r^j_{j+2} & \dots & r^j_{q-1} & r^j_q & r^j_j & s^j_1 & \dots & s^j_{\underline{q}} & g_j \\
r^{j+1}_{j+1} & r^{j+1}_{j+2} & \dots & r^{j+1}_{q-1} & r^{j+1}_q & 0 & s^{j+1}_1 & \dots & s^{j+1}_{\underline{q}} & g_{j+1} \\
0 & r^{j+2}_{j+2} & \dots & r^{j+2}_{q-1} & r^{j+2}_q & 0 & s^{j+2}_1 & \dots & s^{j+2}_{\underline{q}} & g_{j+2} \\
 & & & & & & & & & \\
0 & \dots & 0 & r^{q-1}_{q-1} & r^{q-1}_q & 0 & s^{q-1}_1 & \dots & s^{q-1}_{\underline{q}} & g_{q-1} \\
0 & \dots & 0 & 0 & r^q_q & 0 & s^q_1 & \dots & s^q_{\underline{q}} & g_q
\end{bmatrix}
$$

$$
\rightarrow \begin{bmatrix}
\times & \times & \dots & \times & \dots & \times \\
0 & \times & \dots & \times & \dots & \times \\
 & & & & & \\
0 & 0 & \dots & \times & \dots & \times
\end{bmatrix} \leftarrow g_q
$$

9.6 Implementation Considerations

In this section, we apply the QR recursive Efroymson algorithm to the special RBF network described in chapter 2. We will first discuss how we can incorporate an exponential window into the recursive QR Efroymson algorithm. Then, we will discuss a technique to reduce the computational time.

9.6.1 Exponential Windowing

Because the QR-time-update algorithm is the QR Recursive Least Squares algorithm discussed in chapter 5, an exponential window can be easily incorporated. According to Haykin (1996), an exponential window can be incorporated into the pre-array of the QR-RLS algorithm, Figure 5 - 1, in the following fashion

$$\begin{bmatrix} \lambda\mathbf{R}(k-1) & \lambda\mathbf{g}(k-1) \\ \mathbf{a}^T(k) & d(k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{R}(k) & \mathbf{g}(k) \\ \mathbf{0}^T & \kappa^{-1/2}(k)\xi(k) \end{bmatrix} \qquad (9 - 30)$$

where λ is the exponential window weighting factor $0 \le \lambda \le 1$.

If we apply these changes to the pre-array of the QR-time-update, we get the following modification for Eq. (9 - 11).

$$\begin{bmatrix} \lambda\mathbf{R}_q(k-1) & \lambda\mathbf{S}_q^{(q)}(k-1) & \lambda\mathbf{g}_q(k-1) \\ \mathbf{0}_{q\times q} & \lambda\mathbf{R}_{\underline{q}}(k-1) & \lambda\mathbf{g}_{\underline{q}}(k-1) \\ {}_q\mathbf{a}^T(k) & {}_{\underline{q}}\mathbf{a}^T(k) & d(k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{R}_q(k) & \mathbf{S}_q^{(q)}(k) & \mathbf{g}_q(k) \\ \mathbf{0}_{q\times q} & \mathbf{R}_{\underline{q}}(k-1) & \mathbf{g}_{\underline{q}}(k-1) \\ \mathbf{0}_q^T & {}_{\underline{q}}\mathbf{a}^T(k) & \kappa^{-1/2}(k)\xi(k) \end{bmatrix} \qquad (9 - 31)$$

9.6.2 Reduce Computational Time

As explained in chapter 8, because the RBF network is localized many elements in ${}_q\mathbf{a}(k)$ and ${}_{\underline{q}}\mathbf{a}(k)$ will be near zero. Therefore, we only need to consider selected/potential nodes that have non-zero output as candidates for order-decrease-update/order-increase-update. To implement this scheme, all we have to do is to find those ${}_q\mathbf{a}(k)$ and ${}_{\underline{q}}\mathbf{a}(k)$ that

are close to zero and eliminate them in the order-update. Thus, this implementation affects the FOR loop calculation in the recursive QR forward selection and recursive QR backward elimination. Specifically, we find the indexes of the non-zero elements by thresholding the ${}_{q}\mathbf{a}(k)$ and ${}_{\underline{q-}}\mathbf{a}(k)$ output. With this implementation, we can save a tremendous amount of computation. This will be illustrated in chapter 10.

9.7 Summary

In this chapter, we have abandoned the old QR-RLS-AWS methodology and have developed a completely new scheme based on the QR-RLS algorithm. This new QR-RLS-AWS algorithm is developed based on the Givens QR techniques. We also explored several possible improvements to this new QR-RLS-AWS algorithm. These improvements have resulted in reduced storage, better subset selection (using recursive QR Efroymson) and reduced computation. Theoretically, this algorithm should be numerically more stable than the RLS-AWS algorithm developed in chapter 7. In the next chapter, we will test the numerical stability of this algorithm. Then, we will test the algorithm on several system identification problems.

Chapter

10

Numerical Testing and Applications

10.1 Introduction 198
10.2 Numerical Stability of the Algorithms 199
10.3 Applications 202
10.3.1 Chaotic Time Series 202
10.3.1.1 Recursive Forward Selection Method and Recursive Efroymson Method 203
10.3.1.2 Effects of the Stopping Rules 206
10.3.1.3 Summary 208
10.3.2 1-D Function Approximation 209
10.3.3 2-D Function Approximation 212
10.3.3.1 Comparison of Batch Forward Selection Method and Recursive Efroymson Method 213
10.3.4 Magnetic Levitation System 216
10.3.4.1 On-line Adaptation Results 217
10.3.4.2 Comparison to Multilayer Feedforward Network 221
10.4 Summary 222

This final chapter consists of two parts. The first part tests and compares the algorithms numerical stability by using a test setup described by Trefethen (1997). The second part of the chapter applies the best algorithm (based on numerical testing) to function approximation and system identification problems. We will use the tests to analyze the effects of the parameter settings and also to compare the algorithm to the batch forward selection method and the multilayer feedforward network.

10.1 Introduction

One of the biggest questions regarding the algorithms proposed in this research is the numerical stability of the algorithms. To answer this question, we will conduct a numerical stability test to show the algorithms stability in section 10.2. As shown in the test, out of these four algorithms, the improved QR-RLS-AWS algorithm has shown excellent numerical stability. For this reason, we will only use the improved QR-RLS-AWS algorithm for further testing. We will apply the improved QR-RLS-AWS algorithm to the RBF network on four test problems. The first test is to identify the underlying dynamics of a chaotic time-series. We will use this test to compare two subset selection methods: the recursive forward selection method and the recursive Efroymson method. Both methods utilize the improved QR-RLS-AWS framework. Also, we will examine the effects of the Efroymson algorithm parameter settings. In the second test, we will use a 1-D function approximation problem and examine the effect of the smoothing factor. In the third test, we will compare the results of the batch forward selection method to the recursive Efroymson method. In the last test, we will apply the recursive Efroymson algorithm to identify a magnetic levitation system. We will also compare these results to a multilayer feedforward network.

10.2 Numerical Stability of the Algorithms

In this section, we will conduct a numerical stability test of the four algorithms, the RLS-AWS, the QR-RLS-AWS, the improved RLS-AWS, and the improved QR-RLS-AWS, proposed in previous chapters. To have a baseline for comparison, we use a least squares solving test illustrated by Trefethen (1997 pp.137). Since this test has a known solution, it will be easy for us to compare the numerical stability of various algorithms.

The test illustrated by Trefethen (1997) is to solve a least squares solution of a 100x15 Vandermonde matrix. The MATLAB setup is as follows:

```
m = 100; n = 15;
t = (0:m-1)'/(m-1); % Set t to a interval of [0,1]
A = [];
for i=1:n,
   A = [A t.^(i-1)];% Construct Vandermonde matrix
end
d = exp(sin(4*t));
d = d/2006.787453080206; % Normalization
```

The idea behind this test is to least squares fit the function $\exp(\sin(4t))$ on the interval $[0, 1]$ by a polynomial degree of 14. The last line of code is to normalize the least squares solution so that the parameter $x_{15} = 1$. Trefethen (1997) has shown QR, SVD, normal equation and other computation results in his test.

Since the numerical stability of the time-update algorithms, the RLS and the QR-RLS algorithms, have been thoroughly analyzed in Haykin (1996), we will only test the numerical stability of the order-update algorithms. In all four algorithms, we assume that

we have time-updated the algorithms and we will perform the order-update. Specifically, in both RLS-AWS and QR-RLS-AWS algorithms, we will add each column of $\mathbf{A}(:, i)$ one by one to update the order. In the improved RLS-AWS algorithm, we will first update the time-correlation matrix Eq. (8 - 7) and then perform the order-update. In the QR-RLS-AWS algorithm, we will use the post-array matrix Eq. (9 - 13) to perform the order-update.

Methods	x_{15}	Relative Error
RLS-AWS	0.01192845328609	0.99×10^{-1}
QR-RLS-AWS	-0.32387348994992	1.32×10^{-1}
improved RLS-AWS	0.01192845328609	0.99×10^{-1}
improved QR-RLS-AWS	0.99999994604243	5.40×10^{-8}
Normal Equation	-0.70348736838334	1.70×10^{-1}
QR Householder	0.99999937299332	6.27×10^{-7}
SVD	1.00000004860933	4.86×10^{-8}

Table 10 - 1 Numerical Test Results

Table 10 - 1 summaries the results. We also included three other batch techniques: the normal equation (NE), QR Householder (QR-H), and the Singular Value Decomposition (SVD) for comparison. As shown, the RLS-AWS, the QR-RLS-AWS and the Improved RLS-AWS all fail to compute the accurate value. Note that the RLS-AWS and the Improved RLS-AWS yield identical results because the Improved RLS-AWS uses the same computation but just computed in every update. All three algorithms yield poor results. This is not surprising, as all three algorithms are constructed based on the least

squares method. The improved QR-RLS algorithm yields the most accurate solution among the four algorithms. In fact, the relative error is comparable to the batch SVD algorithm and better than the QR Householder algorithm.

To further test the order-update part of the Improved QR-RLS-AWS algorithm, we perform the order-increase-update and the order-decrease-update many times randomly to see if the solution degrades with each order-update.

(X) # of times order-increase/ decrease-update is performed	x_{15}	Relative Error
50	0.99999994604033	5.40×10^{-8}
100	0.99999994603604	5.40×10^{-8}
200	0.99999994604022	5.40×10^{-8}
400	0.99999994603823	5.40×10^{-8}

Table 10 - 2 Solution of Improved QR-RLS-AWS Algorithm After X-times Order-Update

Table 10 - 2 shows the solution of the improved QR-RLS-AWS algorithm after 50, 100, 200 and 400 randomly chosen order-increase/decrease-updates. As shown, the relative error is kept fairly constant even after 400 random updates. This implies that the solution does not degrade after each order-update. We can conclude that the improved QR-RLS-AWS method has good numerical stability.

10.3 Applications

In this section, we will apply the algorithms developed in previous section to the four problems related to system identification and function approximation. We will only use the improved QR-RLS-AWS algorithm, since it is numerically more accurate than the other three methods Note that if numerical accuracy is not an issue, all methods should yield the same result. In all tests, the RBF network begins with a bias as a node that is not subject to subset selection. Subset selection is used to select the node centers from the potential nodes beginning with time point $k = 0$. We will compare the result of two different subset selection methods: the recursive forward selection and the recursive Efroymson method. In the following, we will describe the test setup and the result that we obtained.

10.3.1 Chaotic Time Series

The chaotic time series generated by the logistic map is a difference equation described by Eq. (10 - 1).

$$z(k+1) = 4z(k)(1-z(k)) \tag{10 - 1}$$

This is a first-order nonlinear process where the previous sample $z(k)$ determines the value of the present sample $z(k+1)$. This logistic map is known to be chaotic on the interval $[0, 1]$. The input-output relationship of this logistic map is plotted on the left of Figure 10 - 1. Before the training, the smoothing factor of the RBF network is selected as $\sigma^2 = 0.125$, and 51 potential RBF network nodes with centers equally spaced at 0.02 in

the interval $[0, 1]$. The output of these 51 potential nodes is plotted on the right hand side of Figure 10 - 1.

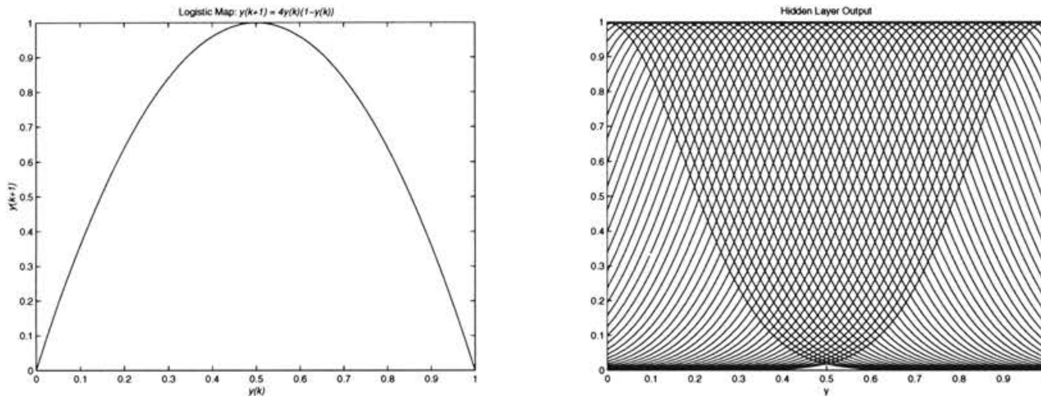


Figure 10 - 1 Input-Output of Logistic Map and the Potential Nodes of the RBF Network

10.3.1.1 Recursive Forward Selection Method and Recursive Efroymson Method

In the following, we will identify the logistic map using the recursive forward selection method and the recursive Efroymson method. Simultaneously, we will compare their performance. The threshold criterion for the forward selection method is set at $\gamma = 10^{-2}$. The F-to-enter and F-to-delete parameters for the Efroymson method are set at $F_e = 2$ and $F_d = 1.5$. For this test, an initial condition $z(0) = 0.39$ is used.

The top two plots of Figure 10 - 2 show the result of the logistic map constructed by the RBF network for 10, 25, and 75 samples using the recursive forward selection method and the recursive Efroymson method. Each sample is presented to the network only once and the training samples are indicated in the top two plots by the plus mark +. Both

techniques have managed to reconstruct the logistic map after 75 samples. (This is indicated by the convergence of the tracking error as shown in the bottom two plots of Figure 10 - 2.) However, the number of nodes used in the recursive forward selection method is greater than in the recursive Efroymson method (8 nodes versus 5 nodes). The placement of these node centers is indicated by the unfilled circles in the top two plots of Figure 10 - 2. The middle two plots show the number of selected nodes (bias included) during the training. As shown, the recursive forward selection algorithm continues to selecting nodes until the error criterion is reached. One difficulty with the recursive forward selection algorithm is that it is possible that the centers selected at an early stage of the training can become unimportant in a later stage. Since the recursive forward selection algorithm is not designed to take out insignificant nodes, more nodes are selected. Meanwhile, the recursive Efroymson algorithm has the ability to take out insignificant nodes during the course of training. This is shown in the middle left plot of Figure 10 - 2, where it took out one node when at $k = 8$. Hence, the recursive Efroymson method is able to keep RBF network size smaller than the forward selection method.

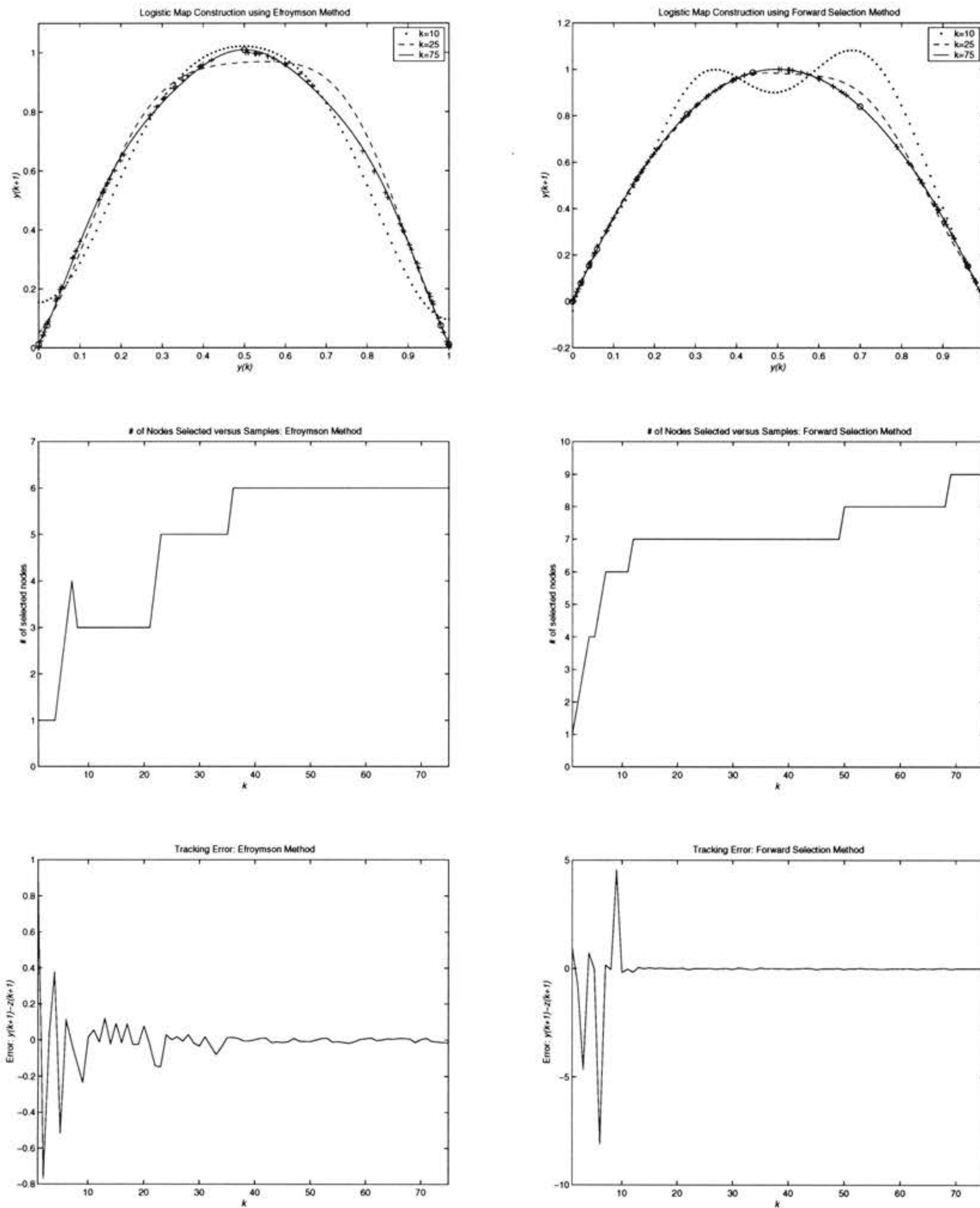
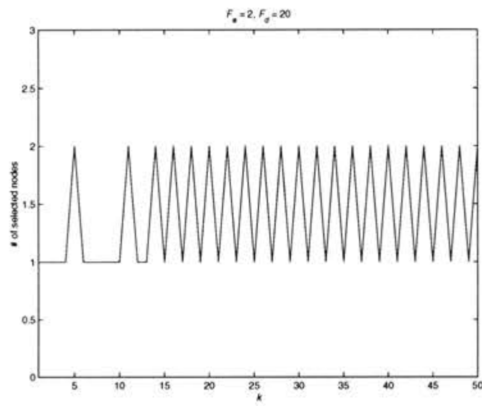


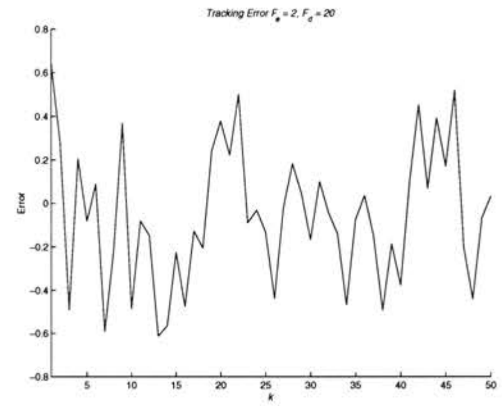
Figure 10 - 2   Comparison of Efroymson Method and Forward Selection Method

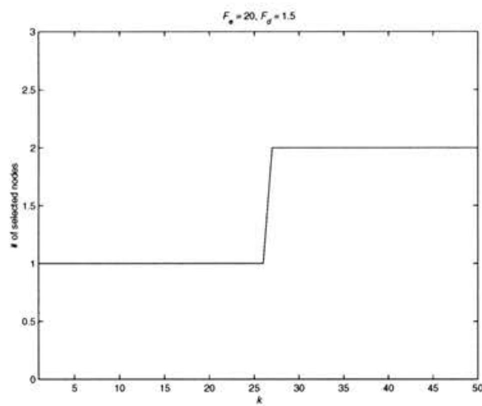
10.3.1.2 Effects of the Stopping Rules

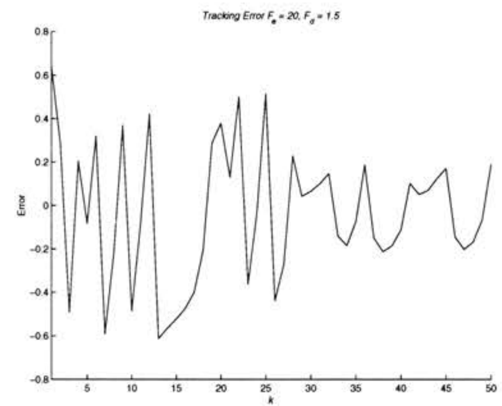
In this section, we will examine the effects of the stopping rules for the recursive Efroymson algorithm using the chaotic time-series as our test problem. We have used the same test setup described in the previous section but will change the values of F_e and F_d. As explained in Chapter 8, we have made an assumption that the recursive Efroymson algorithm is executed at each time-point to ensure fast real-time operations. Since this assumption is crucial for the error convergence, we will need to make sure that the error converges over time. Through numerous experiments, we have demonstrated that this method works well. This is illustrated in the convergence of the tracking error as shown in the bottom left plot of Figure 10 - 2.

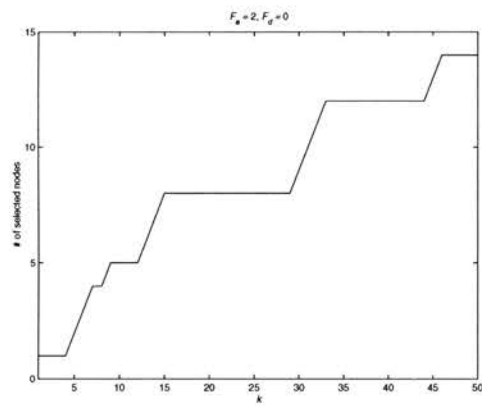
According to Miller (1990), the criteria $F_d < F_e$ has to be satisfied for the batch Efroymson algorithm to ensure the error convergence. We have shown that this is also true for the recursive Efroymson algorithm. As shown in top left plot of Figure 10 - 3, when $F_d > F_e$ ($F_e = 2$ and $F_d = 20$), the recursive Efroymson algorithm would repeatedly select a node in one iteration, then deselect the node in the next iteration. Hence, the tracking error would not convergence (top right plot of Figure 10 - 3).











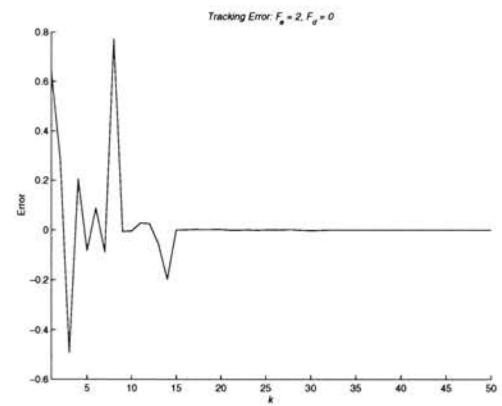


Figure 10 - 3 Effects of F-to-enter and F-to-delete

When the F_e is set to a higher value (e.g. $F_e = 20$), fewer nodes will be selected. This effect is illustrated in the middle left plot of Figure 10 - 3. As shown, only one node and one bias are selected. Consequently, higher tracking error is obtained, as indicated in the middle right plot of Figure 10 - 3.

Meanwhile, the value of F_d is used for deselecting the nodes. A small F_d value (e.g. $F_d = 0$) will cause less significant nodes to be removed from the RBF network. This is illustrated in the bottom left plot of Figure 10 - 3. As shown, not a single node has been removed during the on-line adaptation. Thus, this leads to small tracking error, as shown in the bottom right plot of Figure 10 - 3.

Unfortunately, there is no one set of rules that we can follow to effectively determine the value of F_e and F_d so that the recursive Efroymson algorithm can meet a certain network reconstruction error criteria, as in the forward selection method. This is still an ongoing research topic even for the batch Efroymson algorithm (Miller, 1990).

10.3.1.3 Summary

Due to the possibility that the nodes selected at an early stage of training can become unimportant in a later stage, the forward selection algorithm has to compensate by adding more nodes. Hence, the recursive forward selection method tends to yield a large RBF network. On the other hand, the recursive Efroymson algorithm has the capability of removing insignificant nodes; thus, it yields much smaller RBF networks. Due to this result, we will only use the recursive Efroymson method in the remaining tests.

10.3.2 1-D Function Approximation

In this test, we will perform a 1-dimensional function approximation on a function described by Eq. (10 - 2).

$$z(p) = 0.25 + e^{-p/20}\sin(0.08\pi p) - 0.1\cos(0.01\pi p - 0.5) \qquad (10 - 2)$$

This function is shown in the solid line in the left plot of Figure 10 - 4. We have randomly selected 300 input patterns (+ mark in the left plot of Figure 10 - 4) in the range of [0,50] as our training data. These input samples are fed one-by-one to the algorithm. The sequence of random inputs is shown in the right plot of Figure 10 - 4.

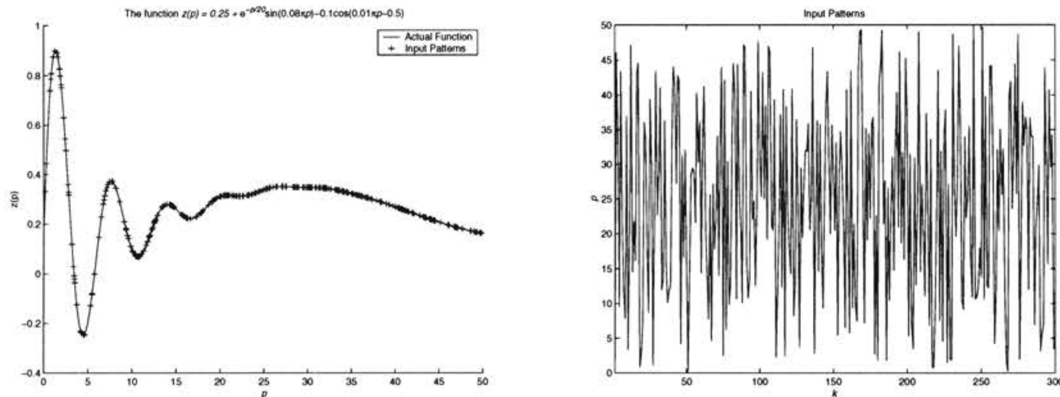


Figure 10 - 4 The Target Function and the Input Patterns

The F-to-enter and the F-to-delete parameters are set at $F_e = 0.1$ and $F_d = 0.07$. A total of 201 potential RBF network nodes with centers equally spaced at 0.25 in the interval $[0, 50]$ are chosen. Three separate tests were conducted, and each test uses a different RBF smoothing factor, specifically $\sigma^2 = 0.5, 2, 10$ are used.

According to Canon & Soltine (1995), this function has significant variation in its local spatial bandwidth, and a regularly spaced RBF network with a single smoothing factor

would not be ideal to fit such a function. Indeed, this phenomenon is reflected in the recursive Efroymson algorithm when we use it to approximate the function. Note that in Figure 10 - 5, the unfilled circles are the selected center locations, the solid line is the RBF network output after 300 iterations of on-line adaptation and the dotted line is the target. At the bottom of each plot are the selected nodes.

When a small smoothing factor $\sigma^2 = 0.5$ is used, many nodes (39 nodes) are used for approximating the flat surface. (This is illustrated in Figure 10 - 5a) Meanwhile, when a large smoothing factor $\sigma^2 = 10$ is used, the RBF network has difficulty in approximating the portion of the signal with large variations (This is illustrated in Figure 10 - 5c). Hence, a smoothing factor that is not too small or too big is required to approximate this function. Such a choice, $\sigma^2 = 2$, is illustrated in Figure 10 - 5b.

Now, since different smoothing factors perform better on different signals, we will include both smoothing factors in the potential nodes. In other words, we let the algorithm decide which centers and smoothing factors to use. Figure 10 - 5d illustrates this idea. We allow potential nodes that use $\sigma^2 = 4$ and $\sigma^2 = 2$ to be available for the algorithm. A total of 401 potential nodes are available. As shown, the recursive Efroymson algorithm is able to select larger smoothing factor nodes for the flat surface and smaller smoothing factor nodes for the regions of the signal with larger variation. In fact, this methodology yields a smaller RBF network (13 nodes) when compared to the single smoothing factor $\sigma^2 = 2$.

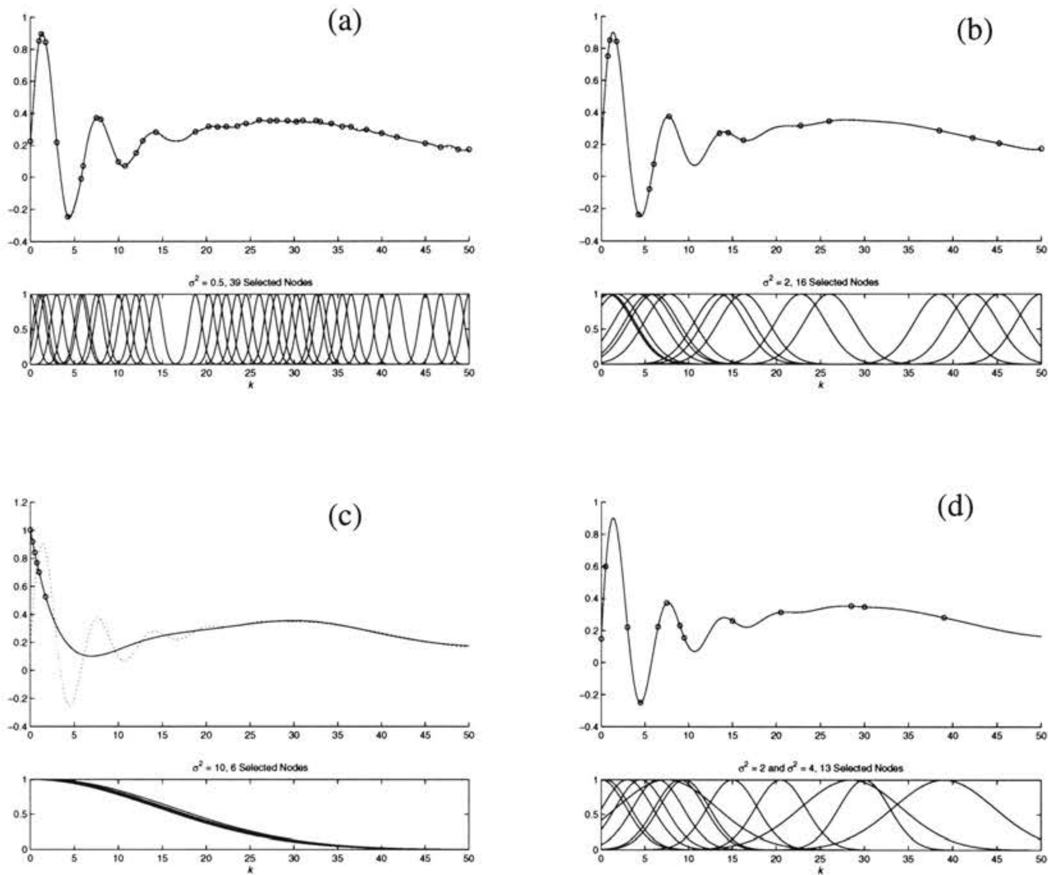


Figure 10 - 5 1-D Function Approximation Results for Different Smoothing Factor

Figure 10 - 6 shows the errors at the conclusion of the previous tests. As shown, the error for the two smoothing factors, $\sigma^2 = 2$ and $\sigma^2 = 4$ (13 nodes selected), is smaller than for the single smoothing factor $\sigma^2 = 2$ (16 nodes selected) and the single smoothing factor $\sigma^2 = 0.5$ (39 nodes).

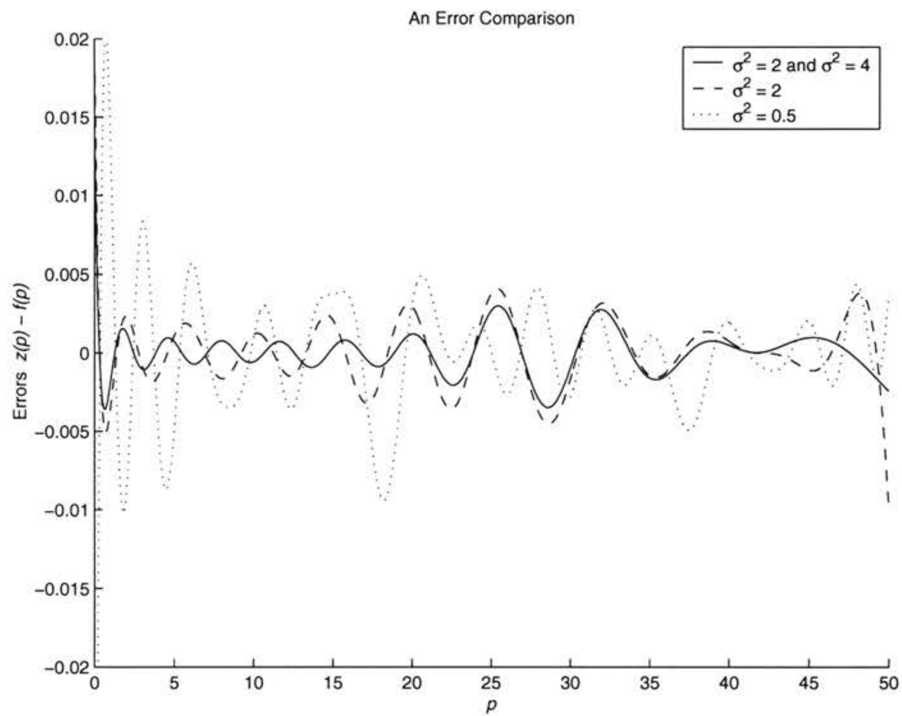


Figure 10 - 6 A Comparison of the Errors

10.3.3 2-D Function Approximation

In this test, we will approximate a surface function (Chang *et. al.*,1996)

$$z(p_1, p_2) = (p_1^2 - p_2^2)\sin(0.5p_1) \qquad (10 - 3)$$

for p_1 and p_2 range from -10 to 10. A 3-dimensional plot of this function is shown in Figure 10 - 7.

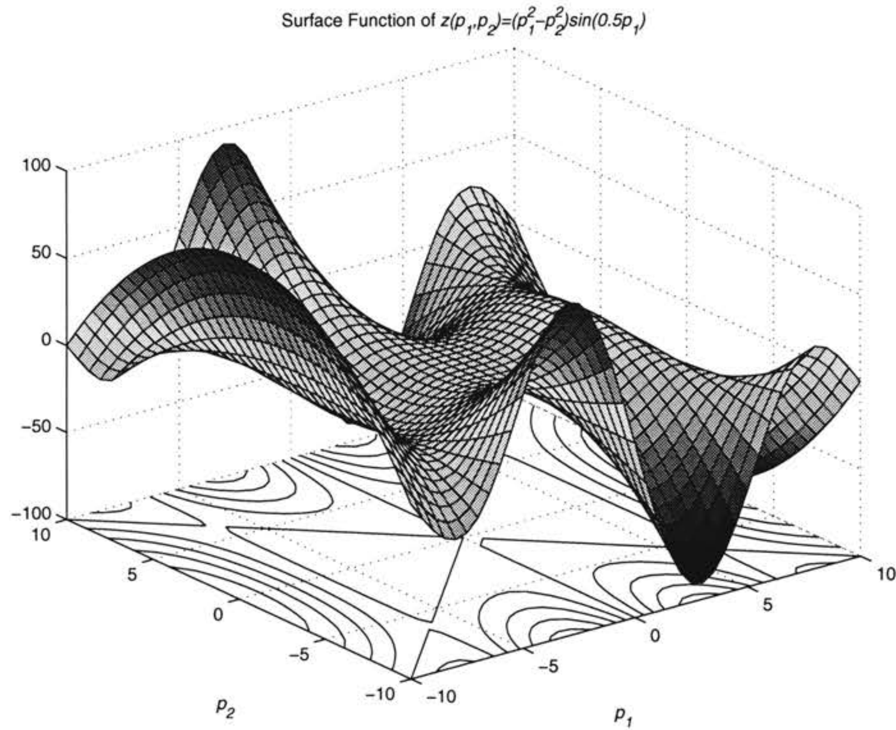


Figure 10 - 7 Surface Function

10.3.3.1 Comparison of Batch Forward Selection Method and Recursive Efroymson Method

In this test, we will compare the performance of the batch forward selection method versus the recursive Efroymson method. For the recursive Efroymson method, the potential RBF nodes are arranged in a two-dimensional grid at intervals of 1 on each axis and in the range of $[-10, 10] \times [-10, 10]$; a total of 441 potential nodes are used. To set up this comparison test, we let the input be the potential centers. We randomly permute these centers and apply them one by one until all 441 randomly permute centers are presented. Meanwhile, the inputs to the batch forward selection algorithm are the 441 randomly permute centers (applied at one time).

Keep in mind that because the recursive Efroymson method is an on-line adaptation method, the future patterns are not available to the algorithm. So, a fair comparison is to compare the network reconstruction error after all 441 input patterns has been fed into the network. Because the value of F_e and F_d determine the size of the RBF network, one way to compare the results is to take the number of nodes that the recursive Efroymson algorithm constructed after the on-line adaptation and compare it to the same size RBF network that the batch forward selection method constructed.

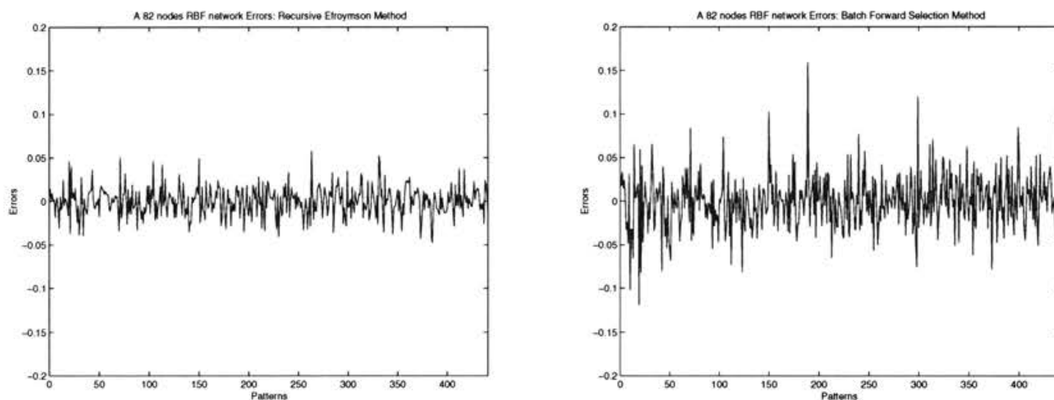


Figure 10 - 8 Errors Comparison for a 82 Nodes RBF Network Constructed by Recursive Efroymson Algorithm and Batch Forward Selection Method

Figure 10 - 8 shows the network reconstruction errors for an 82 node RBF network constructed by the recursive Efroymson method (left plot) and the batch forward selection method (right plot). As shown, the recursive Efroymson method yields a much smaller error. In fact, the sum of squared error is about 3.5 times lower, as shown in Table 10 - 3.

F_e	F_d	Nodes	SSE Recursive Efroymson Method	SSE Batch Forward Selection Method	Ratio
2	1.5	12	134864.4249	140706.8055	1.0433
0.1	0.08	30	4612.7726	7856.3306	1.7032
0.07	0.05	37	374.9623	909.4148	2.4253
0.01	0.008	42	136.2935	392.2697	2.8781
0.005	0.004	55	19.2074	53.2040	2.7700
0.0025	0.0021	61	4.2796	13.9645	3.2630
0.001	0.0008	82	0.1282	0.4493	3.5047

Table 10 - 3 Sum of Squared Errors Comparison between the Recursive Efroymson Method and the Batch Forward Selection Method

Table 10 - 3 summaries several RBF networks of different sizes that have constructed with different F_e and F_d (1st and 2nd column) using the recursive Efroymson method. At the end of the on-line adaptation, the number of nodes selected and the sum of squared errors are recorded in the 3rd and 4th columns. To compare these results, the sums of squared errors of the batch forward selection method, constructed with the same number of nodes, are tabulated in the 5th column. In addition, the ratio of the two sum of squared errors (SEE) is shown in the last column.

As shown in the table, when F_e and F_d are lowered, RBF networks with more nodes and lower SSE are obtained. When we compare the same size RBF network constructed using the batch forward selection method, the recursive Efroymson method yields lower SSE. As the number of nodes increases, the ratio of the SSE increases. This

ratio implies that as RBF network size increases, the recursive Efroymson method selects the same size RBF network with less network reconstruction error.

10.3.4 Magnetic Levitation System

In this last test, we will identify the magnetic levitated system, as described in the Neural Network Toolbox version 4.0 for MATLAB. A diagram of the system is shown in the following figure.

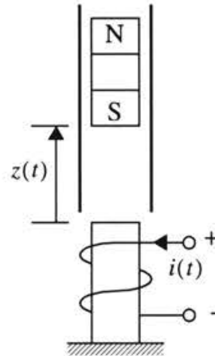


Figure 10 - 9 Magnetic Levitation System

The objective is to identify the magnet position. This magnet is suspended above an electromagnet whose field changes as the current flow changes. This magnet is constrained so that it can only move in the vertical direction. The equation of motion for this system is

$$\frac{\partial^2 z(t)}{\partial t^2} = -g + \frac{\alpha}{M}\mathrm{sgn}(i)\frac{i^2(t)}{z(t)} - \frac{\beta}{M}\frac{\partial z(t)}{\partial t} \quad (10 - 4)$$

where $z(t)$ is the distance of the magnet above the electromagnet, $i(t)$ is the current flow in the electromagnet, M is the mass of the magnet, and g is the gravitational constant. The parameter β is the viscous friction coefficient that is determined by the material in which

the magnet moves, and α is a field strength constant that is determined by the number of turns of wire on the electromagnet and the strength of the magnet. These parameters are set as follows: $M = 3$, $g = 9.8$, $\beta = 12$, and $\alpha = 15$.

10.3.4.1 On-line Adaptation Results

We will use the following NARMA model (Narendra & Parthasarathy, 1990; Ljung 1987) to identify the magnetic levitation system:

$$z(k) = f(i(k), i(k-1), \ldots, i(k-N_i), z(k-1), z(k-2), \ldots, z(k-N_z)) \qquad (10 - 5)$$

Since this is a discrete time model, we assume that the continuous system can be sampled at a specific sampling time and the input and output data are available for on-line adaptation. To identify the magnetic levitated system (the plant), an RBF network with two delayed plant inputs $N_i = 1$ and two delayed plant outputs $N_z = 2$ is used. Figure 10 - 10 is a diagram for this online adaptation scheme (D is the unit time delayed). $z(k)$ is used directly for the algorithm on-line adaptation.

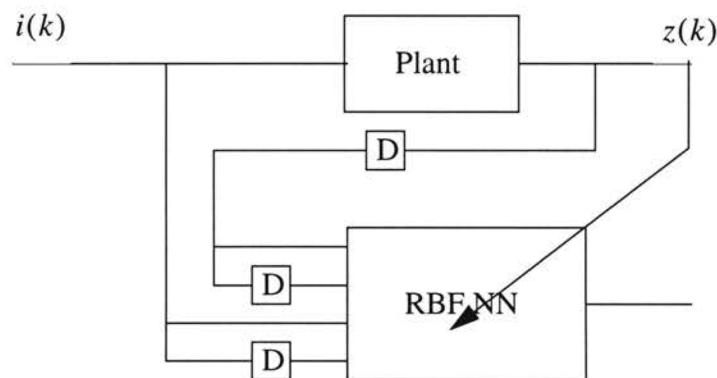


Figure 10 - 10 System Identification Scheme

The data for the on-line system identification is generated by applying a series of steps with random height, occurring at random intervals, to the input of the plant with

sampling time of 0.01 seconds. The left plot in Figure 10 - 11 shows the input sequence (current flow). The magnitudes of the current ranged from $0 < i(t) < 4$, and the widths of the intervals spanned $0.01 < t < 2$. The corresponding plant output is shown in the right of Figure 10 - 11. As shown, the output is stable in the range of $0 < z(t) < 6$.

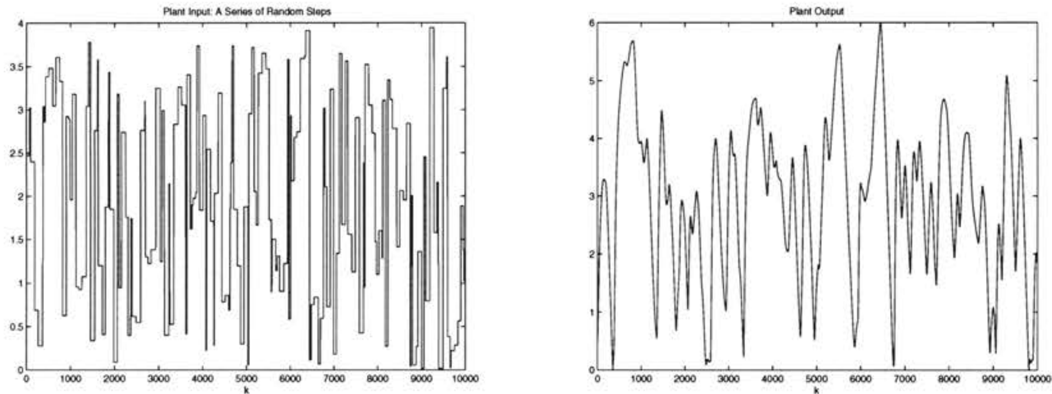


Figure 10 - 11 Input Sequence and Output of the Plant

These on-line data is fed one by one to the RBF network. A total of 1225 potential RBF nodes are used. These potential nodes are equally spaced at intervals of 1 in a 4 dimensional space with input range of $[0, 4] \times [0, 4] \times [0, 6] \times [0, 6]$ for $i(k)$, $i(k-1)$, $z(k-1)$, and $z(k-2)$ respectively. A single smoothing factor of $\sigma^2 = 4$ is used for the RBF network. Also, small $F_e = 0.0009$ and $F_d = 0.0008$ are used since we would like to obtain an accurate model. In addition, we utilize the reduce computational time method as we discussed in Chapter 8.

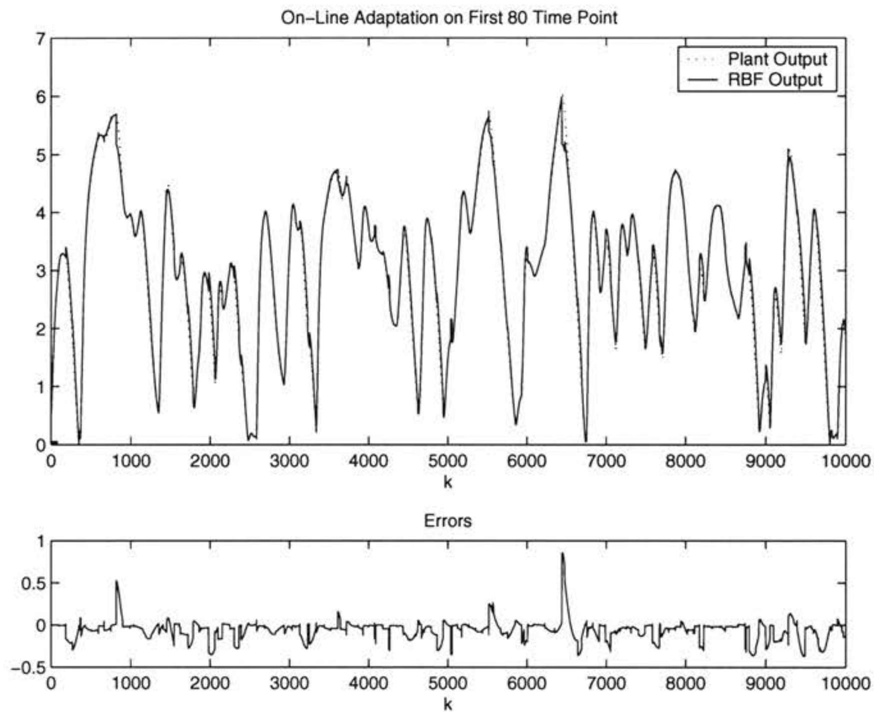


Figure 10 - 12 Result of On-Line Adaptation After 80 Time Point

Figure 10 - 12 shows the result of the on-line adaptation at work. As shown, the plant received the first 80 data points (very tiny portion at the beginning of the plot), but the algorithm has already started to learn the underlying dynamics of the magnetic levitation system (only 6 nodes have been selected at this instant). Training was turned off after the first 80 data points, but Figure 10 - 12 shows that the accuracy is reasonably good on the remaining 9920 untrained data points.

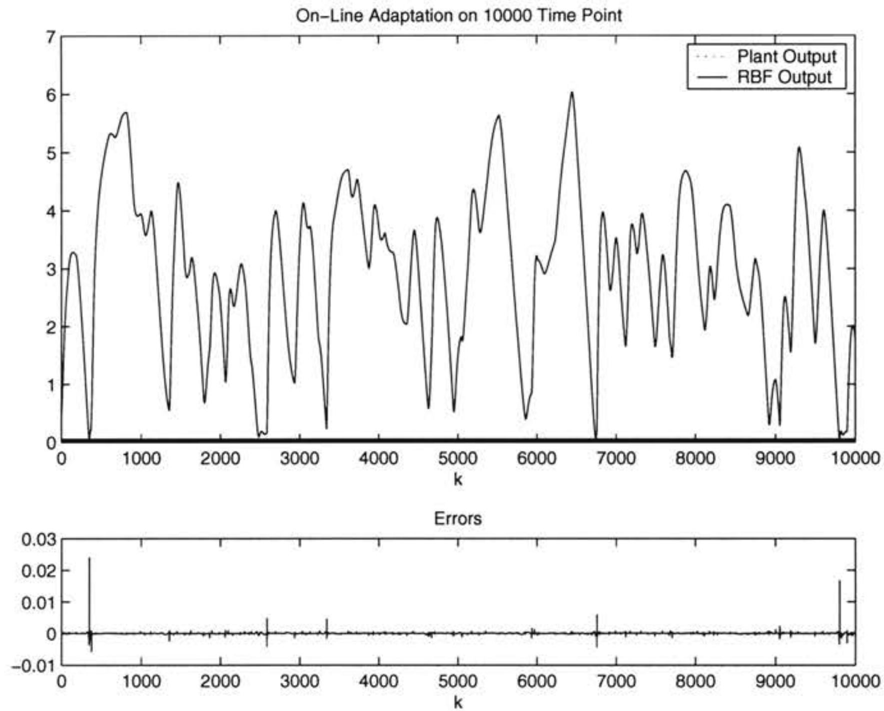


Figure 10 - 13 Results of On-Line Adaptation After 10000 Time Point

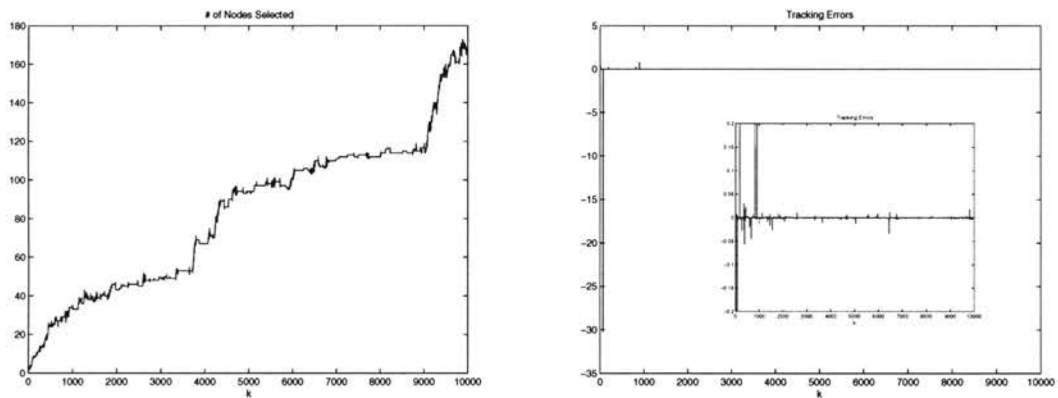


Figure 10 - 14 Number of Selected Nodes and Tracking Error

Figure 10 - 13 shows the results after 10000 data points have been fed into the algorithm. As shown, the errors have converged to ± 0.03. Overall, a total of 168 nodes have been selected by the end of the training (left plot in Figure 10 - 14). The right plot in

Figure 10 - 14 shows the tracking error convergence. We have enlarged the plot so that we can see the error convergence.

One advantage of this algorithm is its ability to adapt on-line. As shown in the tracking error plot, the algorithm seems to have slight difficulty near 9000-th data point. We can see that the algorithm steadily selected more RBF nodes to adapt to these changes.

10.3.4.2 Comparison to Multilayer Feedforward Network

In this section, we will compare the performance of the RBF network to a multilayer feedforward network. Specifically, we will compare the RBF network that has been adapted for 10000 iterations to a multilayer network that has been trained in batch mode. The plant identification of the multilayer neural network is trained using Bayesian regularization. The best multilayer network model is trained and used in this evaluation.

To achieve a fair comparison, we use another 10000 random step sequence to validate both models. Because the error is very small, we will compare the errors. As shown in Figure 10 - 15, the error for the RBF network (right plot) has slightly lower error than the multilayer neural network (left plot).

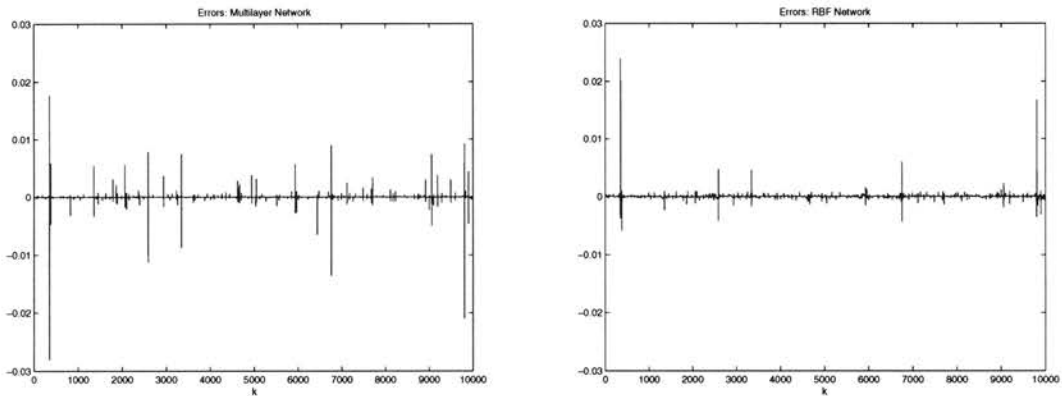


Figure 10 - 15 Error Comparison: The Trained RBF Network and Multilayer Network

10.4 Summary

In this chapter, we have evaluated the numerical stability of several algorithms that we have created in previous chapters. Among these algorithms, the improved QR-RLS-AWS method is numerically stable. We have tested this algorithm on several function approximation and system identification problems. Through these tests, we have found that the QR-RLS-AWS using the recursive Efroymson method has shown good results in creating small and parsimonious RBF networks.

Chapter 11

Summary and Conclusions

11.1 Research Summary 224
11.2 Conclusion 226

In this last chapter, we will summarize our research.

11.1 Research Summary

In this research, we developed a new on-line learning framework that can effectively construct small and parsimonious RBF networks on-line. This framework is adopted by combining three schemes: the time-update, the order-update and the subset selection method. The time-update scheme involves the RLS algorithm and the QR-RLS algorithm, which are readily documented in many books and journals (Haykin 1996, Sayed & Kailath 1992). Meanwhile, the least squares order-update scheme is mathematically derived based on the block matrix inversion lemma. These derivations are tailored to order-increase-update and/or order-decrease-update the parameters in the RLS algorithm. The is a new result. In addition, the orthogonal least squares order-update scheme is mathematically derived based on the QR Givens rotations. These derivations are tailored to order-increase-update and/or order-decrease-update the parameters in the QR-RLS algorithm. This result is also new.

Using this framework, two new algorithms, the Recursive Least Squares with Automatic Weight Selection (RLS-AWS) algorithm and the QR Recursive Least Squares with Automatic Weight Selection (QR-RLS-AWS), have been developed. Both algorithms are recursive in time and order. We first developed the subset selection mechanism of the algorithms based on the forward selection method. This technique allows useful RBF nodes to be added into the network sub-optimally and recursively. Later, we developed an improved subset selection mechanism based on the Efroymson method. This method has

the capability of removing insignificant RBF nodes in addition to adding useful RBF nodes. Both recursive subset selection methods are new.

In addition, we also improved the algorithms' storage requirements. In the RLS-AWS algorithm, we utilized the time-update correlation matrix to reduce storage requirements. In QR-RLS-AWS algorithm, the storage saving is built into the algorithm as shown in Chapter 9. Because the RBF network is localized in space, the outputs of RBF nodes contain many near zero elements. Hence, we can consider potential nodes that have non-zero output as candidates for order-increase-update and can consider selected nodes that have non-zero output as candidates for order-decrease-update. With this method, we can save tremendous amount of computation. Lastly, an exponential windowing scheme can be easily incorporated into the algorithms.

The comparison of these algorithms to the batch forward selection method and the multilayer network are documented in Chapter 10.

11.2 Conclusions

To conclude this research, we will highlight the key results:

- The QR-RLS-AWS algorithm is numerical more accurate than the RLS-AWS algorithm. However, if numerical ill conditioning is not a problem, both algorithms yield the same solution.
- Both subset selection schemes, the recursive forward selection method and the recursive Efroymson method, have been adopted successfully.
- The results have shown that the recursive Efroymson method can produce a smaller RBF network than the recursive forward selection method.
- In simulated results, the QR-RLS-AWS algorithm with Efroymson method has consistently constructed better performance RBF networks than the batch forward selection method.
- In addition, the simulated results also show that the constructed RBF network has performance comparable to the multilayer network.

In conclusion, this research has successfully designed and implemented the recursive time- and order- update algorithms for on-line learning. Although the work described in this dissertation has focused on small RBF networks, the algorithms can be applied to all linear models and all nonlinear models that have a linear-in-parameters structure, such as the fuzzy basis function network, functional-link network, polynomial network, and more.

References

Akaike, H. (1969). Fitting autoregressive models for prediction. Ann. Inst. Statis. Math. Vol.21, pp.221-227.

Ardalan S.H., Alexander S.T. (1987). Fixed-point roundoff error analysis of the exponentially windowed RLS algorithm for time varying systems, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-35, pp.770-783.

Ben-Israel A., Greville T.N.E. (1974). Generalized Inverses Theory and Applications, Wiley, New York.

Berk K.N. (1978). Comparing subset regression procedures, Technometrics, Vo.20, pp.1-6.

Biodini R., Simpson J., Woodley W. (1977). Empirical predictors for natural and seeded rainfalls in the Florida Area Cumulus experiment (FACE) Journal of Application in Meteorology, Vol.16, pp.585-594.

Bjorck A. (1996). Numerical Methods for Least Squares Problems, SIAM, Philadelphia.

Broomhead D.S., Lowe D. (1988). Multivariable functional interpolation and adaptive networks, Complex Systems, Vol.2, pp.281-355.

Cannon M., Slotine J.J.E. (1995). Space-frequency localized basis function networks for nonlinear system estimation and control, Neurocomputing, Vol.9, No.3.

Chang E.-S., Yang H., Bos S. (1996). Adaptive orthogonal least squares learning algorithm for the radial basis function network, Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop, pp.3 -12.

Chen S., Cowan C.F.N., Grant P.M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. IEEE Transactions on Neural Networks, Vol.2, No.2, pp.302-309.

Cohen J., Cohen P. (1983). Applied Multiple Regression/Correlation Analysis for the Behavioral Science, 2nd Edition, Hillsdale, NJ: Erlbaum.

Cun Y.L., Denker J.S., Solla S.A. (1990). Optimal brain damage, in Advances in Neural INformation Processing Systems 2, Touretzky D.S. ed., pp.598-605, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

Cybenko G. (1989). Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems, Vol.2, pp.303-314.

Derksen S., Keselman H.J. (1992). Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables, British Journal of Mathematical and Statistical Psychology, Vol.45, pp.265-282.

Dixon W.J., Brown M.B., Engelman L., Hill M., Jennrich R.I. (1988). BMDP Statistical Software Manual, Vol.1 Berkerly: University of California Press.

Drape N., Smith H. (1981). Applied Regression Analysis, 2nd Ed. New York: Wiley.

Duncan W.J. (1944). Some devices for the solution of large sets of simultaneous linear equations, The Philosophical Magazine Ser. 7 Vol. 35, pp.660-670.

Efroymson M.A. (1960). Multiple regression analysis. In Ralston A., Wilf H.S. Mathematical Methods for Digital Computers, New York: Wiley pp.191-203.

Fabri S., Kadirkamanathan V. (1996). Dynamic structure neural networks for stable adaptive control of nonlinear systems, IEEE Transactions on Neural Networks, Vol.7, No.5, pp.1151-1167.

Fahlman S.E., Lebiere C. (1990). The cascade-correlation learning architecture, in Advances in Neural Information Processing Systems 2, Touretzky D.S. ed., pp.524-532, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

Fletcher R. (1969). A technique for orthogonalization, Journal of Institute Mathematical Applications, Vol.5, pp.162-166.

Funahashi K. (1989). On the approximate realization of continuous mappings by neural networks, Neural Networks, Vol.2, pp.183-192.

Furnival G.M., Wilson R.B. (1974). Regression by leaps and bounds, Technometrics, Vol.16, pp.499-511.

Gentleman W.M., Kung H.T. (1981). Matrix triangularization by systolic arrays, Proceedings of SPIE, Vol.298, Real Time Signal Processing IV, pp.298-303.

Givens W. (1958). Computation of plane unitary rotations transforming a general matrix to triangular form, SIAM Journal of Applied Mathematics, Vol.6, pp.26-50.

Golub G.H., Van Loan C. (1989). Matrix Computations, 2nd Edition, John Hopkins University Press, Baltimore.

Golub G.H., VanLoan C. (1996). Matrix Computations, John Hopkins University Press, Baltimore.

Hagan M.T., Demuth H.B., Beale M. (1996). Neural Network Design, PWS Publishing Company, Boston MA.

Hager W.W. (1989). Updating the inverse of a matrix, SIAM Review, Vol.31, pp.221-239.

Hassibi B., Stork D.G. (1992). Second order derivatives for network pruning: optimal brain surgeon, in Advances in Neural Information Processing Systems 5, S.J. Hanson et al. eds., pp. 164-171, San Mateo, CA: Morgan Kaufmann Publishers, 1992.

Haykin S. (1991). Adaptive filter theory, 2nd Editon, Prentice-Hall, Englewood Cliffs, N.J.

Haykin S. (1994). Neural Networks A Comprehensive Foundation, Macmillan, N.Y.

Haykin S. (1996). Adaptive filter theory, 3rd Edition, Prentice-Hall, N.J.

Hocking R.R. (1976). The analysis and selection of variables in linear regression. Biometrics, Vol.32, pp.1-49.

Hoerl R.W., Schuenemeyer J.H., Hoerl A.E. (1986). A simulation of biased estimation and subset regression techniques. Technometrics, Vo.28, pp.369-380.

Hornik K., Stinchcombe M., White H. (1989). Multilayer feedforward network are universal approximators, Neural Networks, Vol.2, pp.359-366.

Hornik K. (1991). Approximation capabilities of multilayer feedforward networks, Neural Networks, Vol.4, pp.251-257.

Hubing N.E. Alexander S.T. (1990). Statistical analysis of the soft constrained initializationof recursive least squares algorithms, Proceedings of ICASSP, Albuquerque, New Mexico.

Jacobi C.G.J. (1846). Uber ein Leichtes Verfahren Die in der Theorie der Sacularstroungen Vorkommendern Gleichungen Numerisch Aufzulosen, Crelle's J. Vol.30, pp.51-95.

Karayiannis, N.B., Mi, G.W. (1997). Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques, IEEE Transactions on Neural Networks, Vol.8, No.6 pp.1492-1506.

Leshno M., Lin V.Y., Pinkus A., Schocken S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Networks, Vol.6, pp.861-867.

Liu G.P., Kadirkamanatham V., Billings S.A. (1996). Stable sequential identification of continuous nonlinear dynamical systems by growing radial basis function networks. International Journal of Control, Vol.65, No.1, pp.53-69.

Ljung L. (1987). System Identification - Theory for the User. Prentice-Hall, Englewood Cliffs, N.J.

Lowe D. (1989). Adaptive radial basis function nonlinearities, and the problem of generalization, 1st IEE International Conference on Artificial Neural Networks, pp. 171-175, London, UK.

MacKay D.J.C. (1992). Bayesian interpolation, Neural Computation, Vol.4, pp.415-447.

MacKay D.J.C. (1994). Bayesian methods for backpropagation networks, Models of Neural Networks III, Domany E., Van Hemmen J.L., Schulten K., eds., pp.211-254, New York: Springer-Verlag.

Mallows, C.L. (1973). Some comments on C_p. Technometrics, Vol.15, pp.661-675.

Miller A.J. (1984). Selection of subsets of regression variables, Journal of the Royal Statistical Soceity, Series A, Vol.147, pp.389-425.

Miller A.J. (1990). Subset Selection in Regression. Chapman and Hall, N.Y.

Narendra, K.S., Parthasarathy K. (1990). Identification and control of dynamical systems using neural networks, IEEE Transactions on Neural Networks, Vol.1, No.1, pp.4-7.

Ogata K. (1987). Discrete Time Control Systems, Prentice-Hall, N.J.

Park J., Sandberg I.W. (1991). Universal approximation using radial basis function networks, Neural Computation, Vol.3, pp.246-257.

Poggio T., Girosi F. (1990a). Network for approximation and learning, Proceedings of the IEEE, Vol.78, pp.1481-1497.

Poggio T., Girosi F. (1990b). Regularization algorithms for learning that are equivalent to multilayer networks, Science, Vol.247, pp.978-982.

Powell M.J.D., (1987a). Radial basis functions for multivariable interpolation: A review, Algorithms for the Approximation, Mason J.C. and Cox M.G., Oxford, England: Clarendon Press, pp.143-167.

Powell M.J.D. (1987b). Radial basis function approximations to polynomials, Proceedings of 12th Biennial Numerical Analysis Conference (Dundee) pp.223-241.

Ramon y Cajal S. (1911). Histologie du systeme nerveux de l'homme et des vertebres, Paris: Moloine; Edition Francaise Revue: Tome I, 1952;Tome II, 1955; Madrid: Consejo Superior de Investigaciones Cientificas.

Sanner R. (1993). Stable adaptive control and recursive identification of nonlinear systems using radial gaussian networks. Ph.D. thesis, Massachusetts Institute of Technology.

Sanner R., Slotine J.-J.E. (1992). Gaussian networks for direct adaptive control. IEEE Transactions on Neural Networks, Vol.3, No.6.

Sanner R., Slotine J.-J.E (1995). Stable adaptive control of robot manipulators using "neural" networks. Neural Computation, Vol.7, No.4.

SAS Institute (1985). SAS User's Guide: Statistics, 5th ed. Cary, NC: Author.

Sayed A.H., Kailath T. (1994). A state-space approach to adaptive RLS filtering, IEEE Signal Processing Magazine, Vol.11, pp.18-60.

Sherman J., Morrison W.J. (1949). Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix, The Annals of Mathematical Statistics, Vol.20, pp.621.

Slock D.T.M., Kailath T. (1991). Numerically stable fast transversal filters for recursive least squares adaptive filtering, IEEE Transactions on Signal Processing Vol.39, pp.92-114.

Stewart G.W. (1973). Introduction to Matrix Computations, Academic Press, New York.

Trefethen, L.N., Bau D. (1997). Numerical Linear Algebra. SIAM Philadelphia, PA.

Tzirkel-H.E., Fallside F. (June 1992). Stable control of nonlinear systems using neural networks, International Journal on Robust Nonlinear Control, Vol.2, No. pp.63-86.

Weisberg S. (1980). Applied Linear Regression. New York: Wiley.

Woodbury M. (1950). Inverting modified matrices, Memorandum Rept. 42, Statistical Research Group. Princeton University, Princeton, N.J.

Yang B. (1994). A note on the error propagation analysis of recursive least squares algorithms, IEEE Transactions on Signal Processing, Vol.42, pp.3523-3525.

2

VITA

Meng Hock Fun

Candidate for the Degree of

Doctor of Philosophy

Thesis: RECURSIVE TIME- AND ORDER- UPDATE ALGORITHMS FOR RADIAL BASIS FUNCTION NETWORKS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Johor Bahru, Malaysia on March 5, 1970, the son of Swee Chin, Fun and Choo Mui, Yap.

Education: Graduated from Taylor's College, Kuala Lumpur, Malaysia in 1989; received Bachelor of Science degree in Electrical Engineering and Master of Electrical Engineering degree from Oklahoma State University, Stillwater, Oklahoma, in December 1993 and May 1996, respectively. Completed the requirements for the Doctor of Philosophy degree in Electrical Engineering at Oklahoma State University in August 2001.

Experience: Employed by Oklahoma State University, Department of Electrical Engineering as a teaching assistant and as a graduate researcher from 1993 to present.

Professional Status and Memberships: Member of Phi Kappa Phi Society, Tau Beta Phi Society, Eta Kappa Nu Society, Institute of Electrical and Electronic Engineers Society, and United States Chess Federation.