Important Notice

UNIVERSITY OF CALGARY

The application of multivariate statistics and neural networks to the prediction of reservoir parameters using seismic attributes

by

Brian Henderson Russell

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF GEOLOGY AND GEOPHYSICS

CALGARY, ALBERTA

SEPTEMBER, 2004

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "The application of multivariate statistics and neural networks to the prediction of reservoir parameters using seismic attributes" submitted by Brian Henderson Russell in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____

Supervisor, Dr. Laurence R. Lines, Geology and Geophysics

_____

Dr. R. James Brown, Geology and Geophysics

_____

Dr. Gary F. Margrave, Geology and Geophysics

_____

Dr. Robert R. Stewart, Geology and Geophysics

_____

Dr. Michael P. Lamoureux, Mathematics and Statistics

_____

Dr. Matthew J. Yedlin, University of British Columbia

_____

Date

ABSTRACT

In this dissertation, I develop a number of new ideas for the statistical determination of reservoir parameters using seismic attributes. These ideas combine the classical techniques of multivariate statistics and the more recent methods of neural network analysis. I apply these techniques to both full seismic volumes and to maps derived from intervals averaged through these volumes, largely using the Blackfoot dataset from central Alberta. I show that multilinear regression often provides too simple a solution to the parameter estimation problem, and that the traditional feedforward neural network often provides a solution that is overly complex. My proposed solution is to use radial basis function neural networks for the prediction of reservoir parameters, since this approach combines the power of the multilinear regression technique with the nonlinearity of neural networks. I also show how the radial basis function neural network can be considered as a generalization of the generalized regression neural network, which has been previously used in this type of parameter estimation. My conclusions are illustrated using both an AVO classification problem and the Blackfoot seismic and well log dataset.

In addition to the application of the radial basis function neural network to the prediction of reservoir parameters, several new ideas are presented for the analysis of well log and seismic data. First, I derive an improved regression formula for the prediction of S-wave sonic logs from combinations of other logs. Second, I apply a new approach to data clustering, which I call Mahalanobis clustering, to the interpretation of AVO crossplots and to the delineation of optimal clusters for the radial basis function neural network with centres method. Finally, I develop a new approach to map analysis that combines geostatistics with multiattribute transforms. This technique uses multivariate statistics and neural networks to improve the secondary dataset used in the collocated cokriging technique.

PREFACE

Although this dissertation was technically written over the last four years (2000-2004), it is really the product of my thirty years in geophysics, both as a student and as a practitioner of the science. During this time I have worked as a field geophysicist, a seismic interpreter, a seismic processor and, most recently, as a seismic software developer. The thread that ties all this work together is my interest in seismic inversion. Initially this meant deterministic inversion using the convolutional model, applied to both poststack and prestack seismic data. These methods form the basis of Chapter 2 of this study. Thanks to the work of Ronen et al. (1994) and Jim Schuelke and his colleagues at Mobil (now ExxonMobil) I was introduced to the idea of statistical inversion. The latter approach was implemented by Hampson-Russell Software as the EMERGE program (Hampson et al., 2001) and I was closely involved with testing and training on this product. The most important motivation for me in developing the ideas that form the basis of this dissertation was in trying to reconcile the generalized regression neural network proposed by Hampson et al. (2001) with the radial basis function approach of Ronen et al. (1994). This is the basis of Chapters 6 through 8 in this study. However, to understand the building blocks upon which these methods are based, I was further motivated to study the areas of multivariate statistics and the single and multi-layer perceptron, which form the basis of Chapters 3 through 5 of the study. The result has been a period of intense, but satisfying, work on this area of exploration geophysics, which I hope will motivate others to use these tools.

ACKNOWLEDGEMENTS

First and foremost I want to acknowledge my partner in life, Elaine, who not only suggested that it was not too late for me to go back to university to fulfill a lifelong dream, but also gave me emotional support through four arduous years and acted as my literary editor (as she said: "Some days I wish I could turn your thesis upside down and shake out all those superfluous commas like a bunch of annoying insects".)

Second, thanks go to my longstanding business partner and colleague, Dan Hampson, who has been a daily source of inspiration and ideas for the last twenty years. Dan also acted as my tutor for the C++ programming skills that allowed me to write the source code for the RBFN method.

Third, I want to acknowledge my thesis advisor, Dr. Larry Lines, who has been instrumental in guiding me towards the completion of this dissertation. Larry has been easy going enough to let me largely go my own way, firm enough to push me when necessary, and always totally free with his time, even when he took on the onerous task of Department Head.

Finally, I want to thank all my colleagues at both Hampson-Russell and the CREWES project who have provided a great intellectual environment for me over the last four years. In particular, I want to acknowledge two individuals. Jon Downton, Larry's other "mature" Ph.D. student, has been a constant source of inspiration to me through our discussions of both his research and my own. And Chris Ross at Hampson-Russell gave me an idea that led to many of the examples in this dissertation when he pointed out that my analysis of the XOR function and how it could be solved was virtually identical to the analysis of an AVO class 3 anomaly.

DEDICATION

This dissertation is dedicated to my amazing wife, Elaine; to my wonderful children, Michelle and David; and to my magical grandchildren, Tyson and Avery.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

1.1 The fundamental objective

The fundamental objective of this dissertation is to study the relationship between reservoir parameters and seismic attributes. The reservoir parameters will include such measurements as P-wave velocity, porosity, and water saturation. The seismic attributes will include instantaneous attributes, windowed attributes, AVO attributes, and seismically derived impedance, all of which will be described in Chapter 2. Specifically, I will determine a relationship between a set of seismic attributes and a reservoir parameter at a number of specific well locations, and then use this relationship to compute reservoir parameters from sets of seismic attributes throughout a seismic volume. To address this problem, I will make use of a wide variety of tools, ranging from deterministic seismic techniques such as impedance inversion and amplitude variations with offset (AVO), to multivariate statistics and neural networks. This is a problem that has been discussed by other authors (Ronen et al., 1994; Hampson et al. 2001) and I will utilize many of the techniques that they introduced, as well as introduce several new approaches.

The overall structure of this study is as follows. In this chapter I will qualitatively describe the inter-relationships between seismic attributes and target reservoir parameters. I will also introduce the bias-variance dilemma, a crucial problem that we face. In Chapter 2, I will discuss the theory behind the attributes used in the subsequent chapters. Chapter 3 will deal with the linear regression problem and Chapter 4 will discuss the linear classification problem. In chapters 5 through 7 I will then apply neural network techniques to reservoir parameter computation. Specifically, Chapter 5 will

utilize the traditional neural network approach, called the feedforward neural network, or multilayer perceptron, Chapter 6 will utilize the kernel regression technique and neural networks based on this technique and Chapter 7 will introduce the radial basis function neural network with cluster centres. Chapter 8 will then discuss the combination of linear and neural network computation methods with geostatistics. Finally, Chapter 9 will give a summary of the complete study.

1.2 A qualitative example

To illustrate the goals of this study more graphically, let us start with a straightforward example. Figure 1.1 shows a target reservoir log on the left, and three seismic attributes on the right.

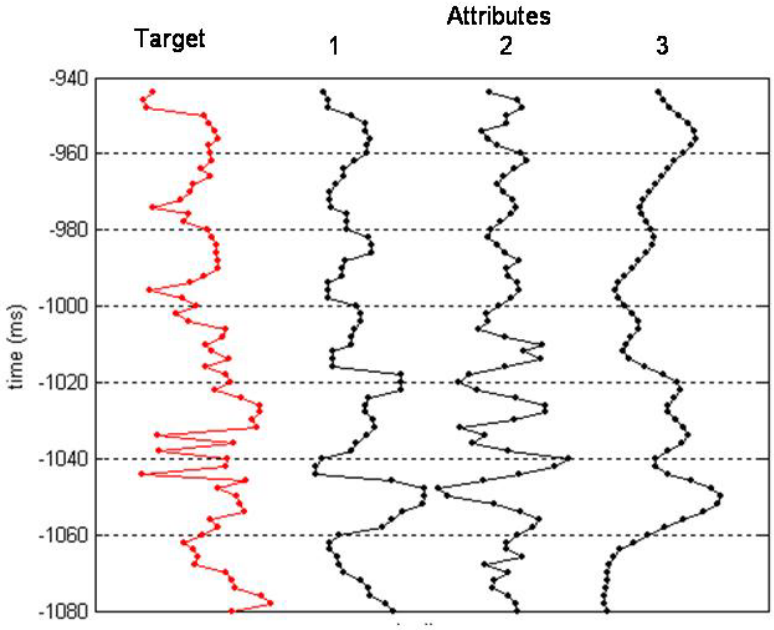


Figure 1.1: A target sonic log is shown on the left and three seismic attributes, extracted at the same location, are shown on the right. The dots represent time samples.

The P-wave sonic log shown in Figure 1.1 has been integrated to time and correlated with the seismic data so that there is a one-to-one correspondence between the samples on the well log and the samples on the seismic attributes (shown as dots on the

display). The log and each attribute contain 69 samples. In the general case (see Appendix 1) we will have *M* attributes and *N* time samples. At the moment, I will not be concerned with how these particular seismic attributes were chosen, or even which attributes they are (this will be addressed in the next two chapters), but will instead consider their inter-relationships. This can be done by cross-plotting the target and attributes against each other, as shown in Figure 1.2. This type of plot is called a *P*-dimensional scatter plot (Johnson and Wichern, 1998) where, in this case, $P = 4$. In both Figure 1.1 and Figure 1.2, the target log and attributes have been normalized by subtracting the mean and setting the largest absolute amplitude to either 1 or -1. This means that each crossplot has the same scale and is centered on zero.

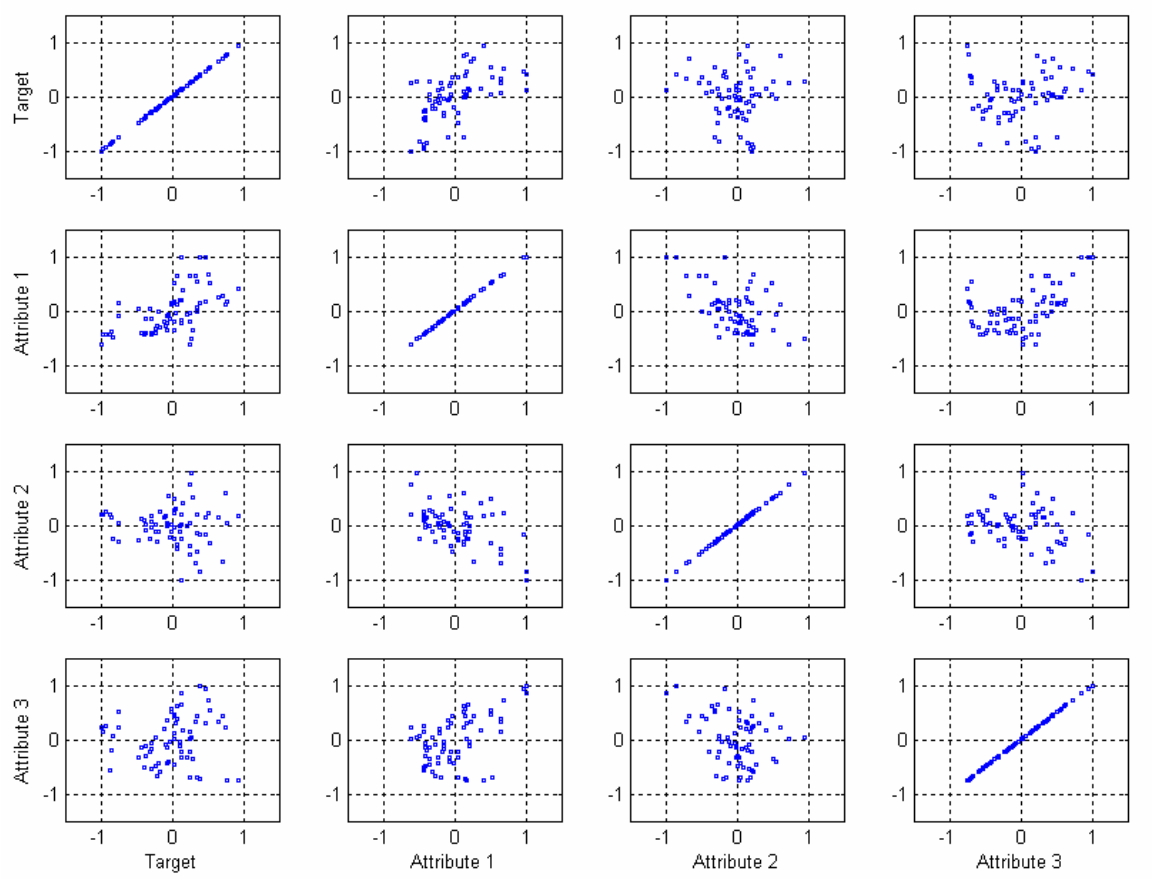
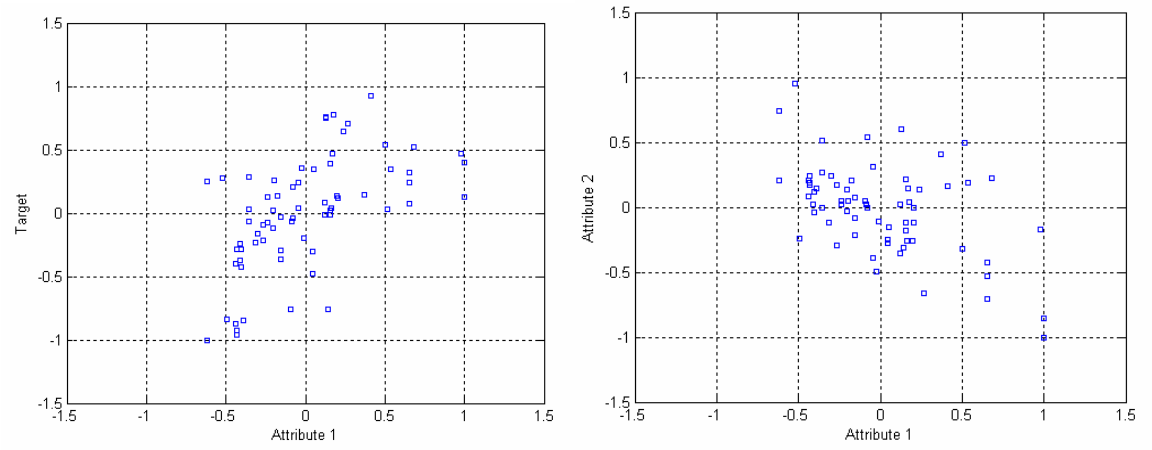


Figure 1.2: A 4 x 4 matrix of crossplots, where the target log and each of the attributes shown in Figure 1.1 have been crossplotted against each other.

As will be discussed in Chapter 3, Figure 1.2 contains quantitative information about the multivariate statistics of our dataset, and can be thought of as the visual equivalent of the covariance matrix. Before getting to this quantitative analysis, let us make several qualitative observations about these plots. First, notice that there is a lot of redundancy in the crossplots shown in Figure 1.2, since the six crossplots in the upper right triangular part of the matrix use the same pair of variables as the six crossplots in the lower left triangular part. The only difference between the upper and lower sets of plots is that the order of the dependent and independent variables are reversed. But there is quite a different look to the plots and, as I will discuss in Chapter 3, the statistics of the relationship will change when the order of the dependent and independent attributes is reversed.

Notice also that there are three types of crossplots shown in Figure 1.2. The first type of crossplot, shown on the main diagonal of the figure, represents the autocorrelation of the target or a particular attribute against itself. This is perfect linear correlation, and the other crossplots fall short of this ideal to a greater or lesser extent. Although this crossplot will always give a straight line, the distribution of the points on the line, which can be continuous or grouped into clusters, will give us information about the distribution of the values in the dataset being correlated.

The second type of plot is shown along the top row and left column of the matrix. These are the crossplots of the target log with each attribute. As discussed in the next section, these plots introduce the idea of linear regression. Figure 1.3(a) shows a blowup of the second crossplot on the top row, in which the Target is on the vertical axis, and Attribute 1 is on the horizontal axis. Note that there appears to be a positive linear correlation of the target with Attribute 1, suggesting that as Attribute 1 increases, so does the target value. But there is also a lot of scatter this crossplot, and the linear fit is far from perfect.



(a) (b)

Figure 1.3: "Blowup" of (a) the Target vs Attribute 1, and (b) Attribute 2 vs Attribute 1 crossplots from Figure 1.2.

Finally, the third type of crossplot is the crossplot of each attribute against each other attribute. This shows the amount of correlation between attributes and will be used as the basis of classification. Figure 1.3(b) shows a blowup of the crossplot with Attribute 2 on the vertical axis and Attribute 1 on the horizontal axis. In this figure there appears to be a negative correlation between the two attributes, and again there is scatter is similar to that in Figure 1.3(a). By looking again at Figure 1.2, notice that there is scatter in all of the crossplots. We certainly do not get a good fit between any of the individual attributes and the target log. This sets the stage for all that follows in this dissertation, since I will be looking for optimum methods of combining the attributes to predict the target log.

1.3 Classification and regression

As mentioned in section 1.1, we can divide the approaches used in this study into two separate types: classification and regression. The term *classification* refers to the subdivision of the input samples into a set of output classes. An example of this would be the classification of the samples of a 3D seismic volume into a set of lithology classes (e.g. sand, shale, or carbonate) or fluid classes (e.g. gas, oil, or water). Such a classification scheme can be either *supervised*, where we use known classes on the target

log to guide the classification of the seismic attributes, or *unsupervised*, where we look for natural clusters within the crossplot of seismic attributes. The classification techniques used in this study will largely be supervised classification techniques. (The one exception is the *K*-means clustering technique that will be used in Chapter 7, which is an unsupervised method.)

The term regression refers to the mapping of *N* input vectors (again of dimension *M*, where *M* represents the total number of input attributes) into *N* output scalar values, where these scalars represent the values of the reservoir parameter that we wish to compute. An example would be the computation of a continuous parameter such as water saturation, density, or resistivity from the same set of *N* seismic attribute vectors described in the classification problem. The difference between clustering and regression, when applied to the data shown in Figure 1.1, is shown in Figures 1.4 and 1.5.

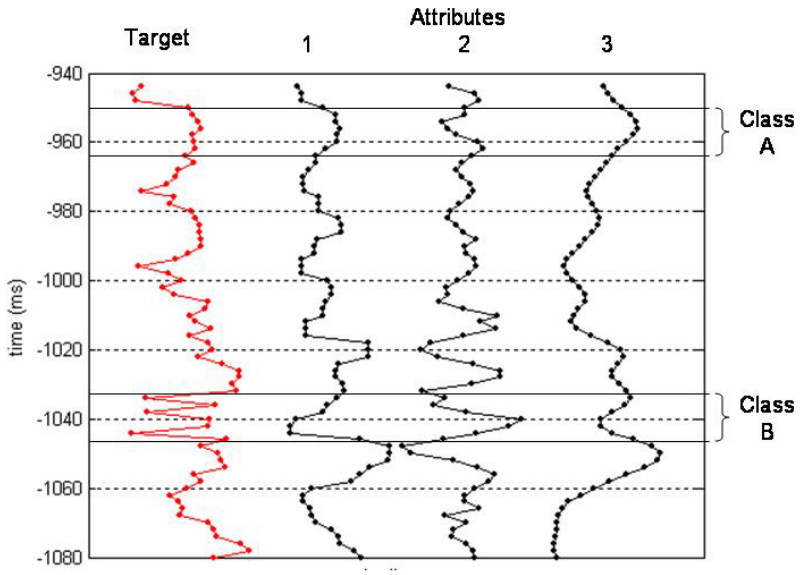


Figure 1.4: The classification problem, where classes A and B have been defined based on the target log.

Figure 1.4 gives us a qualitative idea of supervised clustering. In this figure, I have labelled two zones on the log as Class A and Class B, as determined by their different characteristics. Our objective in classification is to determine whether the other

samples fall into either Class A or Class B (or neither Class), and then apply this relationship to locations at which the log is not present.

In Figure 1.5, we see the basic regression problem, again applied to the example of Figure 1.1. A single sample has been shown just above a time of 1000 ms. The points in the box enclosing the seismic attributes are an M-dimensional vector $\boldsymbol{x}_j$, where $M = 3$. (Figure 1.5 also shows the difference between the vector of attributes, $\boldsymbol{x}_j$, and the attribute vector $\boldsymbol{a}_i$, which is more fully explained in Appendix 1). The point in the box enclosing the target log is a scalar value t_j. Mathematically, we want to compute the target value from the attribute vectors for all N samples in the window, where $N = 69$, since the sample rate is 2 ms.

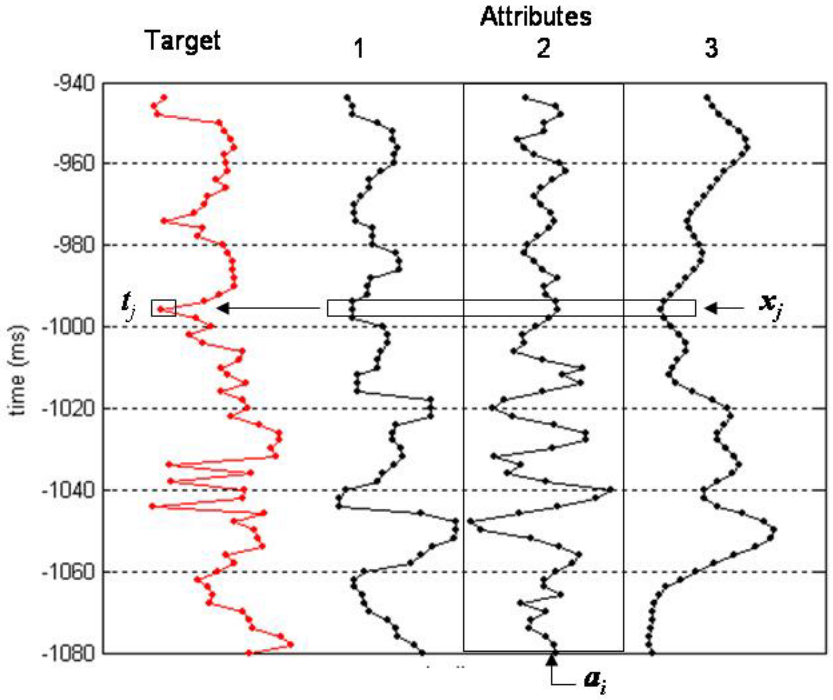


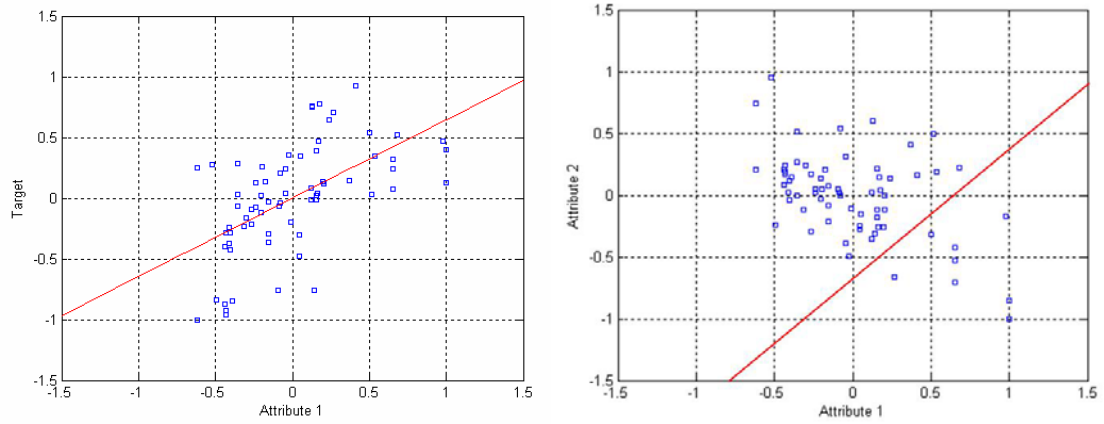
Figure 1.5: The basic multilinear regression problem, where we want to predict t_j from $\boldsymbol{x}_j$.

Comparing Figures 1.4 and 1.5, note that the key difference is that the regression problem assigns a value to each of the N samples, whereas the classification problem assigns each of the N samples to one of K classes. Although I will be utilizing both linear and nonlinear approaches to the solution of the regression and classification problems, the concept is best understood using the linear methods of Chapters 3 and 4.

As an introduction to these concepts, Figure 1.6 illustrates the difference between linear regression and classification utilizing the crossplots of Figure 1.3. Figure 1.6(a) shows the target vector crossplotted against Attribute 1 from Figure 1.1, as shown in Figure 1.3(a). However, this time I have plotted the regression line of t against a, where a is a particular attribute. That is, I have computed the best least-squares coefficients to the regression equation

$$t(a) = w_0 + w_1 a \,. \tag{1.1}$$

Notice that the fit is not very good in this case, due to the scatter of the points. We can quantify this scatter using the correlation coefficient, which will also be discussed in Chapter 3.



(a) (b)

Figure 1.6 : The concepts behind (a) linear regression, and (b) linear classification, applied to the crossplots of Figure 1.3

Figure 1.6(b) shows Attribute 1 crossplotted against Attribute 2 from Figure 1.1, as shown in Figure 1.3(b). Superimposed on the points is the line that separates two classes, let's say C_1 and C_2. That is, we have computed the coefficients to the equation

$$y(\boldsymbol{a}) = w_0 + w_1 a_1 + w_2 a_2 \,, \tag{1.2}$$

where $\boldsymbol{a} = (a_1, a_2)^T$, and y can be interpreted from the equation:

$$y(a) < 0 \Rightarrow a \in C_1$$
$$y(a) \geq 0 \Rightarrow a \in C_2$$

As we will see in subsequent chapters, equation (1.2) can be interpreted either as a linear discriminant function or as a single-layer perceptron. (The perceptron is the fundamental building block of the multilayer feed forward neural network and will be described in detail in Chapters 4 and 5.) The line in Figure 1.6(b) therefore represents the boundary between two sets of points, not the regression of the target against the attribute, as shown in Figure 1.6(a). Thus, although Figures 1.6(a) and (b) look very similar, there is a big difference in their interpretation, as indicated by equations (1.1) and (1.2). Equation (1.1) is a one-dimensional regression equation and indicates that the target values can be computed from a single attribute value. The fact that the fit is not very good, and that very few of the computed values will correspond to the correct values, is due to the poor underlying model assumption of the linear fit (more about this in section 1.5). Equation (1.2) is actually a two dimensional fit, and the fact that the line separates two sets of points implies a third axis pointing away from the two-dimensional plot. The values on this axis don't really matter, as long as they are distinct for the two classes. In many two-class problems, these values are given as 0 and 1, and in other cases they are given as -1 and +1. Chapter 4 uses an example in which the latter values are used.

1.4 Function approximation

The distinction between classification and regression is somewhat arbitrary since classification becomes regression as $K \to N$. (For example, predicting water saturation with arbitrary accuracy by regression is simply a more precise way of classifying a sample as being water or gas.) In fact, both can be seen as particular cases of the general problem of function approximation (Bishop, 1995), in which we try to discover the underlying function, or model, that relates our observed data to the earth parameters. That is, we would like to find the function

$$y = f(\boldsymbol{x}),$$

where y is our reservoir parameter and $\boldsymbol{x}$ is a vector of seismic attribute values. In the ideal case, f would be a known function. For example, if $\boldsymbol{x}$ were a two-dimensional vector, we could find a fit using a function such as

$$y = A_1 \sin(x_1) + A_2 \cos(x_2).$$

Unfortunately, such analytical functions rarely describe real data, so a more practical solution is to find an approximate relationship given by

$$y = f(\mathbf{x}; \mathbf{w}),$$

where $\mathbf{w}$ is a set of weights, and the function f can represent either membership in a class (that is, y takes on a set of discrete values such as, in the simplest case, *0* or *1*), or some linear or nonlinear regression function. As can be seen in the previous section, equations (1.1) and (1.2) represented linear regression and classification functions, respectively.

1.5 Bias and variance

As just discussed, our goal is to find a relationship that best relates a set of observed data values to the physical parameters that describe the earth. That is, we are searching for a model that relates seismic measurements to reservoir parameters. Because we do not assume a deterministic model, but rather will search for an underlying statistical relationship, the fundamental question is: how can we judge the validity of the derived model? On one hand, we do not want a model that is too simple and does not have the flexibility to ever fit our data points. This is called a model with high bias. On the other hand, we do not want a model that is too complex and has such a high degree of flexibility that it overfits the data. This is called a model with high variance. The optimal model would be complex enough to fit the data values reasonably well, but not so complex that it fits the noise in the data. This is a common problem in all statistical methods of data fitting, and is called the bias versus variance dilemma.

To understand the bias-variance problem, let us consider the problem of trying to estimate a single period of a sine wave of unit amplitude which has been contaminated with random noise and sampled N times. That is

$$f(x_i) = \sin(\pi x_i) + \eta_i, \tag{1.3}$$

where $x_i = 0, 1/N, 2/N \ldots, 1.0$ and η represents the random noise component. Figure 1.7 shows the sine wave and three different noise contaminated versions, each a different noise realization with a variance of 0.2. The sine wave has been sampled with $N = 21$.

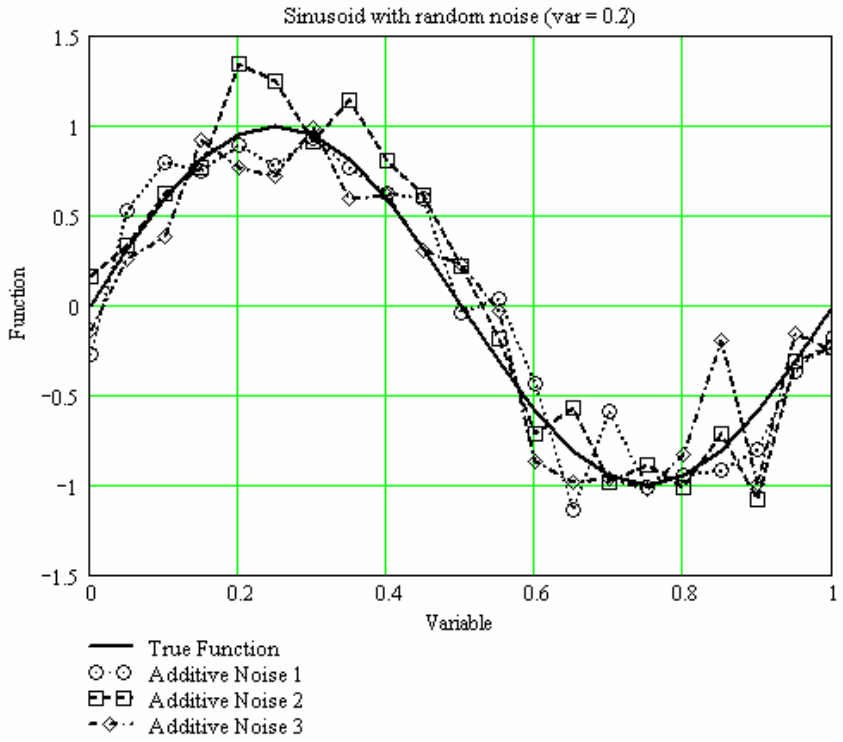


Figure 1.7: A sinusoid (heavy black line) corrupted with three realizations of random noise of variance 0.2.

To estimate the sine wave from the noise corrupted signals, we will use an Mth order polynomial fit, given by

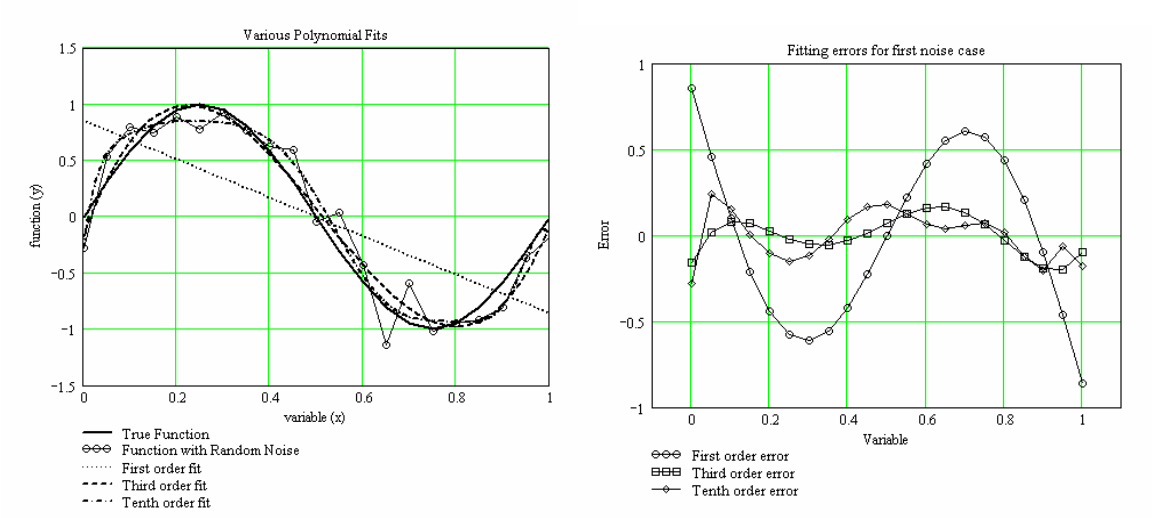
$$y(x) = w_0 + w_1 x + \ldots + w_M x^M . \tag{1.4}$$

For each noise-corrupted signal, I will use polynomial fits of orders 1, 3, and 10.

Figure 1.8(a) shows the three polynomial fits to the first noisy trace from Figure 1.7, with the original sinusoid and the noisy points plotted for reference. Figure 1.8(b) then shows the errors between the three fitted polynomials and the original sinusoid. The first-order fit has the largest error and is therefore not a good model for the sinusoid.

This is obvious since a first-order fit is a straight line and can not be expected to fit a sinusoid. In other words, the first-order fit has a high bias.

The third- and tenth-order fits show a reasonably good fit to the original sinusoid, with roughly similar error functions. That is, the third- and tenth-order fits have low bias. Since the error for the tenth-order fit appears to be slightly lower than the error for the third-order fit, we may therefore decide to use a tenth order fit to model our data.



(a) (b)

Figure 1.8: Polynomial fits to the first noise corrupted sinusoid of Figure 1.7, where (a) shows the three polynomial fits, and (b) displays the three error plots.

But which of the two fits has the lowest variance, or lack of change between models? To judge this, we need to look at all three realizations of the data from Figure 1.7. Figure 1.9 therefore shows the three polynomial fits to all three noisy signals, with the original sinusoid plotted for reference on each set of polynomial fits.

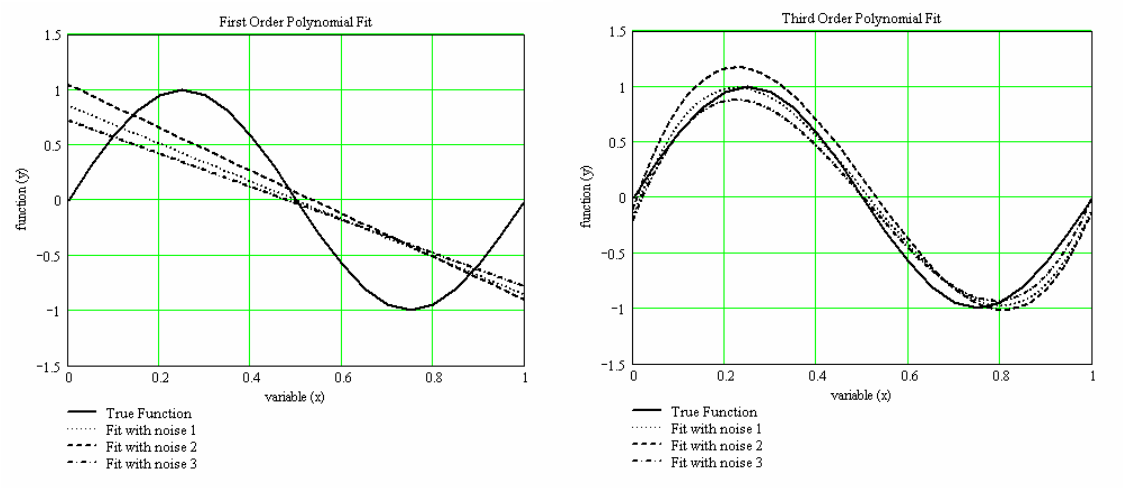


Figure 1.9: Polynomial fits to the three noise corrupted sine waves of Figure 1.7, where (a) is a first-order fit, (b) is a third-order fit, and (c) is a tenth-order fit. In each case, the true curve is shown as a heavy line.

From Figure 1.9(a), it is clear that the first-order fit is too simple a model (i.e. has high bias) to fit the sinusoid. However, notice that the tenth-order fit (Fig. 1.9(c)) produces a significantly different result for each data realization. This means that the tenth-order fit is too complex a model, since it tends to honour the noise too well. That is, the tenth-order fit has too high a variance. To see this even more clearly, look at Figure 1.10, which shows the errors from the three polynomial fits.

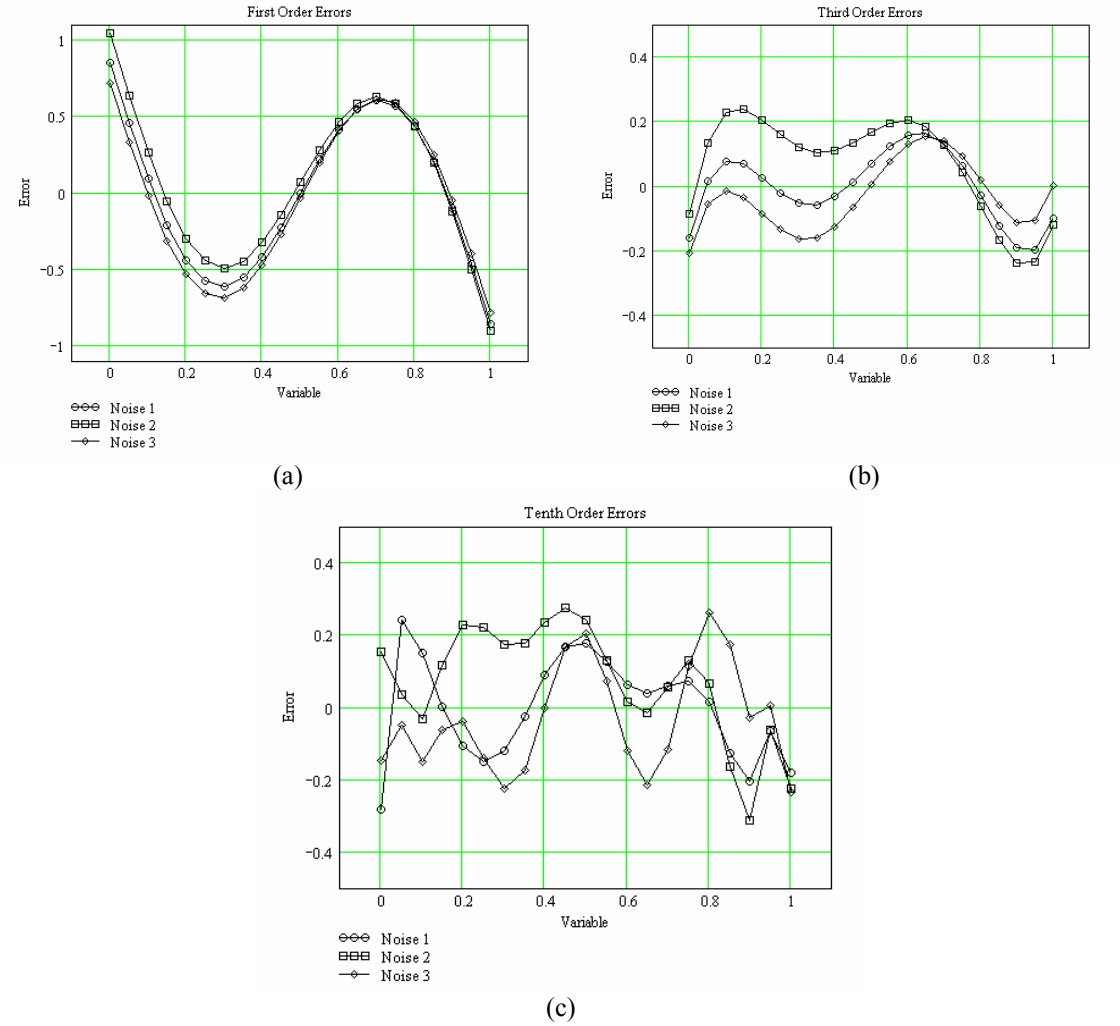


Figure 1.10: Errors between the actual sinusoid and the polynomial fits to the three noise corrupted sine waves of Figure 1.9, where (a) is the first order, (b) is the third order, and (c) is the tenth order error.

It is clear in Figure 1.10 that the errors are consistent for the first-order and third-order fits, as shown in Figures 1.10(a) and (b). However, the errors are highly variable for the tenth-order fit shown in Figure 1.10(c). Unfortunately, in the real data cases that I will be using in this dissertation we do not know the underlying model. This means that we need to find an alternate way to solve the bias-variance dilemma. One of the most accepted ways of doing this is to use some type of cross-validation procedure, in which we leave out part of the data, and compare our computed result with the data that has been left out. This is also often called “blind prediction”.

1.6 Introduction to artificial neural networks

In much of this dissertation, I will be using various types of artificial neural networks (ANNs) to find the relationship between the target log and the attributes. Thus, we first need to answer the question: what is a neural network? The simplest answer is that a neural network is a mathematical algorithm that can be trained to solve a problem that would normally require human intervention. Although there are many different types of neural networks, there are two ways in which they are categorized: by the type of problem that they can solve and by their type of learning.

Neural network applications in seismic data analysis generally fall into one of two categories: the classification problem, or the regression problem. In the classification problem, as discussed in section 1.3, we assign an input sample to one of several output classes, such as sand, shale, limestone etc. In the prediction problem we assign a specific value to the output sample, such as a porosity value.

Neural networks can also be classified by the way they are trained, using either supervised or unsupervised learning. In supervised learning the neural network starts with a training dataset for which we know both the input and output values. The neural network algorithm then "learns" the relationship between the input and output from this training dataset, and finally applies the "learned" relationship to a larger dataset for which we do not know the output values. The most common example of the supervised learning neural network is the multi-layer perceptron, or MLP, which has become almost synonymous with the term ANN. This method also goes by several other names, such as the multi-layer feed-forward neural network (MLFN) and the backpropagation neural network (BPNN). In Chapter 5 I will describe the MLP neural network and apply this approach to both a simple model, in which we can intuitively derive the weights for the network, and a real data example. Although the MLP is the most common artificial neural network, there are many other approaches described in the literature. In this dissertation,

I will describe and apply three other neural network approaches: the probabilistic neural network, or PNN, the generalized regression neural network, or GRNN, and the radial basis function neural network, or RBFN. The PNN will be used for classification, and the GRNN and RBFN will be used for nonlinear regression.

In unsupervised learning, we present the neural network with a series of inputs and let the neural network look for patterns itself. That is, the specific outputs are not required. The advantage of this approach is that we do not need to know the answer in advance. This disadvantage is that it is often difficult to interpret the output. An example of this type of unsupervised technique is the Kohonen self organizing map (KSOM) (Kohonen, 2001). In this dissertation, I will be exclusively concerned with neural networks that use supervised learning.

CHAPTER 2 : SEISMIC ATTRIBUTES

2.1 Introduction

In this chapter, I will discuss the various seismic attributes that will be used to compute reservoir parameters. A seismic attribute can be defined as a transform, either linear or nonlinear, of the seismic trace. Seismic attributes can be classified into the following seven basic types:

(1) instantaneous attributes, which are derived from a combination of the input seismic trace and the Hilbert transform of the trace,

(2) windowed frequency attributes, in which the amplitude spectrum of seismic trace is computed over a running window,

(3) recursive attributes, which are derived by applying a recursive operator along the seismic trace,

(4) bandpass attributes, which are narrowband filter slices of the seismic traces,

(5) multi-trace attributes, which are found by applying an operator to a local collection of seismic traces in a 3D volume,

(6) AVO attributes, which are derived from prestack seismic data, and

(7) model-based attributes, in which an a priori model is included as a component of the final solution.

It should be pointed out that some authors (Chen and Sidney, 1997) do not include model-based attributes in their definition of seismic attributes. I have chosen to do so in this dissertation based on the observation that model-based attributes often give the best overall fit to the reservoir parameter that we are predicting. From this viewpoint, we can consider the statistical methods to be discussed in the subsequent chapters to be

improvements on classical deterministic methods of parameter estimation such as seismic inversion (Russell, 1988).

A second distinction when discussing seismic attributes is between sample-based attributes and horizon-based attributes. Sample-based attributes are computed on a sample-by-sample basis over the complete seismic volume, whereas horizon-based attributes are computed as the average over a time window around a seismic horizon (either at constant time or picked on an event) throughout a 3D volume. The key difference between sample-based attributes and horizon-based attributes is that there is a one-to-one correspondence between the samples in a seismic volume and the samples in a sample-based attribute, whereas there is no one-to-one correspondence in a horizon-based attribute, which is used in spatial, or map, form. In most of this dissertation I will deal with sample-based attributes. In Chapter 9, I will discuss horizon-based attributes.

Examples of all seven types of attributes have been taken from the Blackfoot dataset, which will be used in several case studies in subsequent chapters (Dufour et al., 2002). A 3C-3D seismic survey was recorded in this area in October 1995, with the primary target being the Glauconitic member of the Mannville group. The reservoir occurs at a depth of around 1550 m, where Glauconitic sand and shale fill valleys incised into the regional Mannville stratigraphy. The objectives of the survey were to delineate the channel and distinguish between sand-fill and shale-fill.

Figure 2.1 shows a map view of the seismic survey and available well coverage, with the inline and cross-line numbers annotated on the four corners of the map. Seismic inline 27 is shown as a vertical red line on the map, intersected by well 08-08.

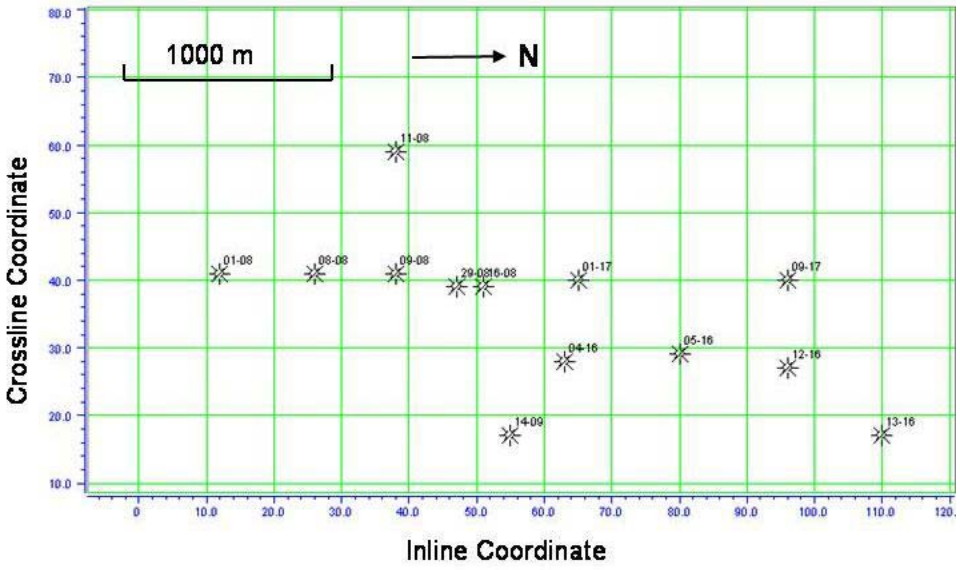


Figure 2.1. A map view of the Blackfoot dataset, with wells annotated.

Figure 2.2 shows the seismic section from inline 27 on the map shown in Figure 1.1. The synthetic seismogram from well 08-08 has been spliced into the section at crossline 42.

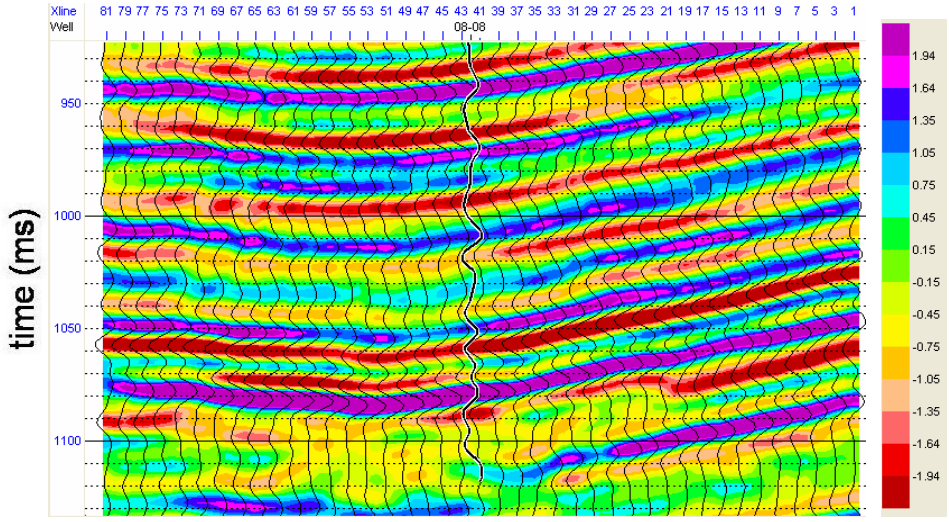


Figure 2.2. The input seismic line used to display the various attributes in this section. The synthetic seismogram from a nearby well (08-08) is shown at Xline 42.

In following sections, I will use the seismic section shown in Figure 2.2 to illustrate the different types of seismic attributes listed above.

2.2 Instantaneous attributes

Instantaneous attributes were first computed from seismic data by Taner et al (1979), using theory that had been developed for electrical engineering applications (Bracewell, 1965). The basis of the instantaneous attributes is the complex seismic trace, which can be written:

$$C(t) = s(t) + i\, h(t) \tag{2.1}$$

where $s(t)$ = the seismic trace, $h(t)$ = the Hilbert transform of the seismic trace, often called the quadrature trace, and $i = \sqrt{-1}$. Rewriting equation (2.1) in polar form, we get

$$C(t) = A(t)\exp(i\phi(t)), \tag{2.2}$$

where $A(t) = \sqrt{s(t)^2 + h(t)^2}$ is referred to as the amplitude envelope or instantaneous amplitude, and $\phi(t) = \tan^{-1}(h(t)/s(t))$ is referred to as the instantaneous phase. Equation (2.2) can also be expressed as:

$$C(t) = A(t)\cos(\phi(t)) + iA(t)\sin(\phi(t)), \tag{2.3}$$

which tells us that the seismic trace is equal to the product of the amplitude envelope and the cosine of instantaneous phase, and the quadrature trace is the product of the amplitude envelope and the sine of the instantaneous phase.

Figure 2.3 shows the Hilbert transform of the seismic section shown in Figure 2.2. Since both the seismic section and its Hilbert transform look similar, one way to observe the difference between the two is to compare their wiggle traces at the well intersection with the synthetic seismogram from well 08-08. Notice that the synthetic seismogram differs by a 90 degree phase shift on the Hilbert transformed section, when compared to the original seismic section of Figure 2.2.

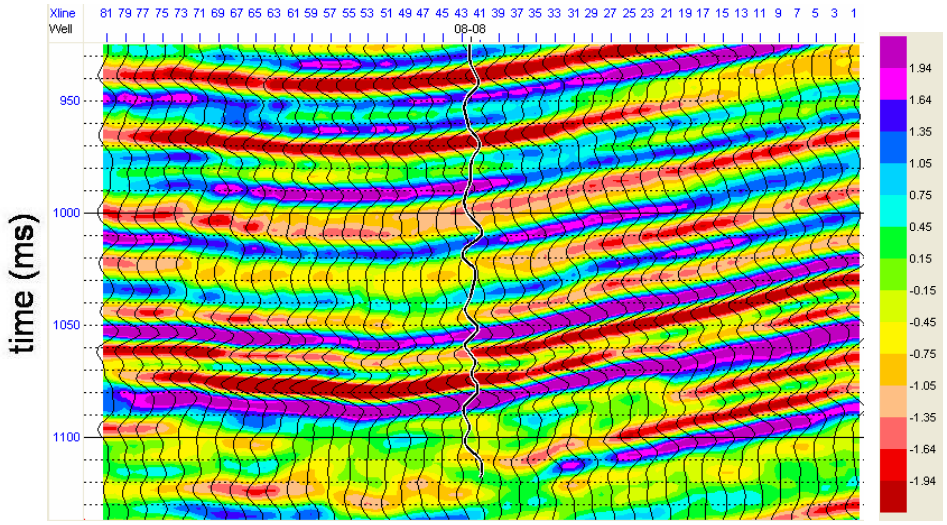


Figure 2.3. The Hilbert transform of the seismic section shown in Figure 2.1.

Figure 2.4 shows the amplitude envelope and Figure 2.5 shows the instantaneous phase of the same section shown in Figure 2.1. As expected, the amplitude envelope section of Figure 2.4 emphasizes the amplitude changes from the original seismic section, and the instantaneous phase section of Figure 2.5 shows the subtle stratigraphic detail.

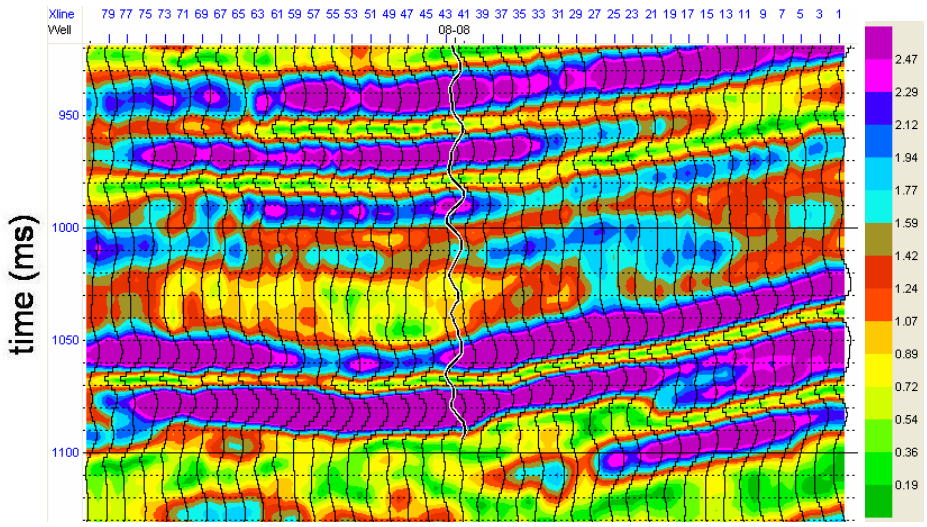


Figure 2.4. The amplitude envelope of the seismic section shown in Figure 2.1. The integrated P-wave sonic log from a nearby well (08-08) is shown at Xline 42 on both seismic displays.

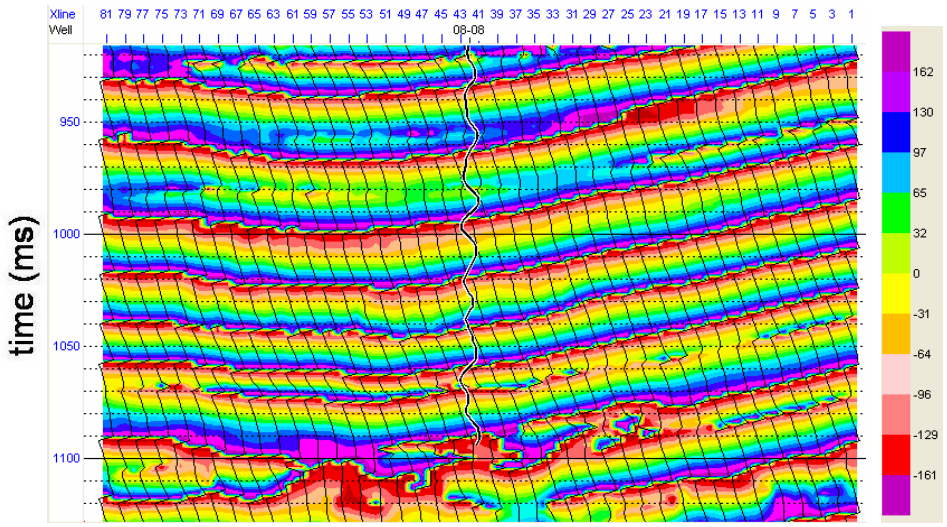


Figure 2.5. The instantaneous phase of the seismic section shown in Figure 2.1.

The traces on the instantaneous phase section (Fig. 2.5) show sharp jumps at ±180 degrees, so the colour bar on the instantaneous phase has been designed to be identical at the two extremes. For a more "seismic-looking" trace than the instantaneous phase, Equation (2.3) suggests taking the cosine of the instantaneous phase, as shown in Figure 2.6.

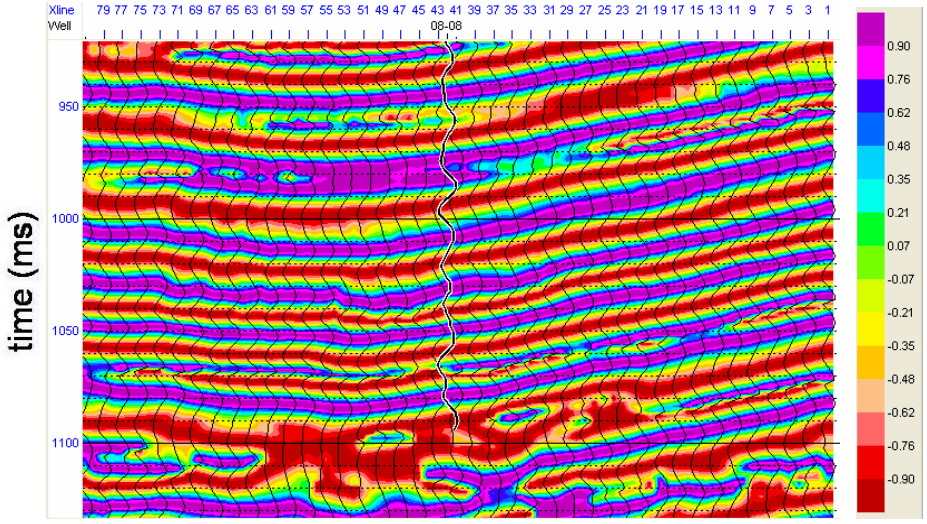


Figure 2.6. The cosine of the instantaneous phase section shown in Figure 2.5.

The cosine of instantaneous phase, shown in Figure 2.6, is a seismic section in which the amplitude changes have been removed, as if a strong gain control has been

applied. Note from Equation 2.3 that the product of the sections in Figures 2.4 and 2.6 gives us the original seismic section of Figure 2.2.

A third instantaneous attribute can be found by differentiating the instantaneous phase to get:

$$\omega(t) = \frac{d\phi(t)}{dt} \tag{2.4}$$

where $\omega(t)$ is termed the instantaneous frequency. Figure 2.7 shows the instantaneous frequency of the same section as in Figure 2.2. Notice the alternately low and high instantaneous frequency bands between 1050 and 1100 ms.

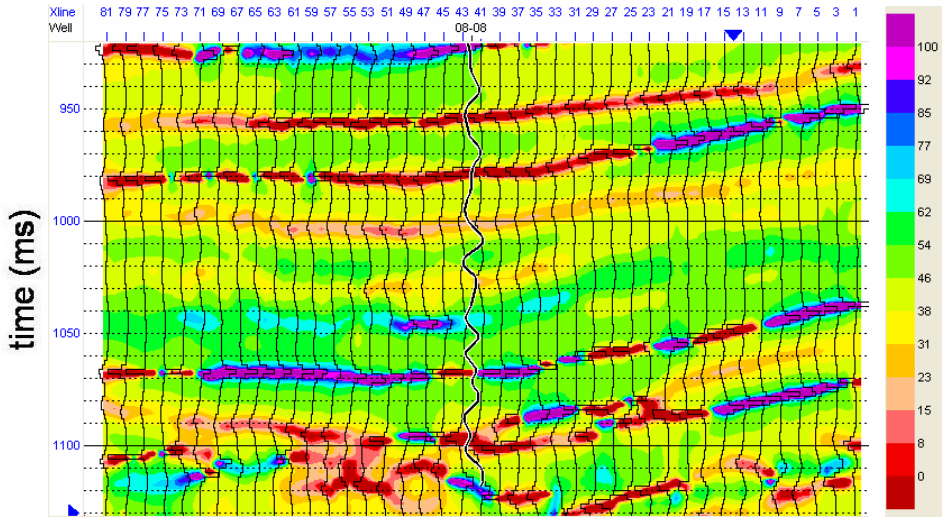


Figure 2.7. The instantaneous frequency section derived from the seismic section shown in Figure 2.2.

The amplitude envelope, instantaneous phase and instantaneous frequency are the three primary attributes, but many more can be derived from these basic three. These new attributes are usually products of the existing three attributes.

2.3 Windowed frequency attributes

To compute a windowed frequency attribute, the Fourier transform of the seismic trace is computed over a window of fixed size around a seismic sample and some property of the transform is extracted. To compute a sample-based attribute of this type,

either the average or dominant frequency is computed and placed at the sample location. This procedure is then repeated using a running window down the trace. Figure 2.8 shows the dominant frequency attribute from the section shown in Figure 2.2, using a window size of 64 ms and an increment between samples of 32 ms. The samples in between the 32 ms increment are then interpolated. As can be seen, this produces a long-period, smoothly varying estimate of the frequency content of the data.

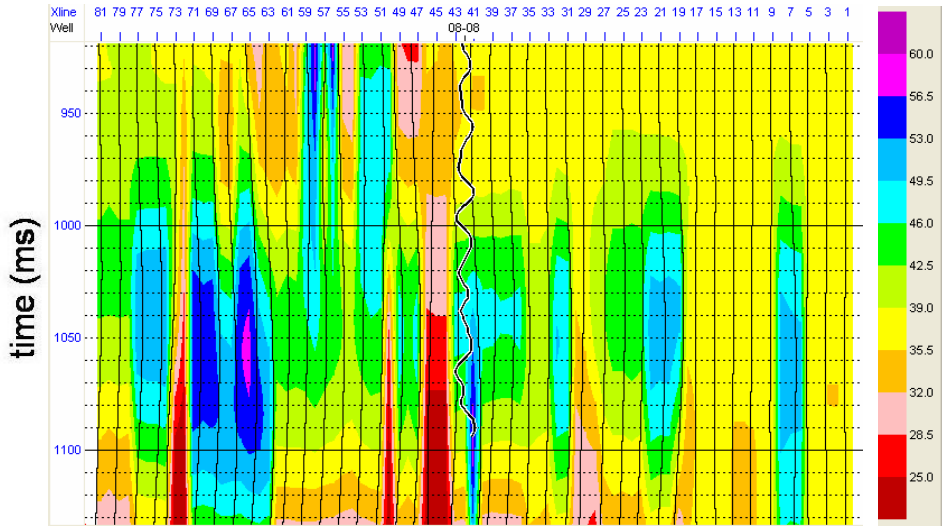
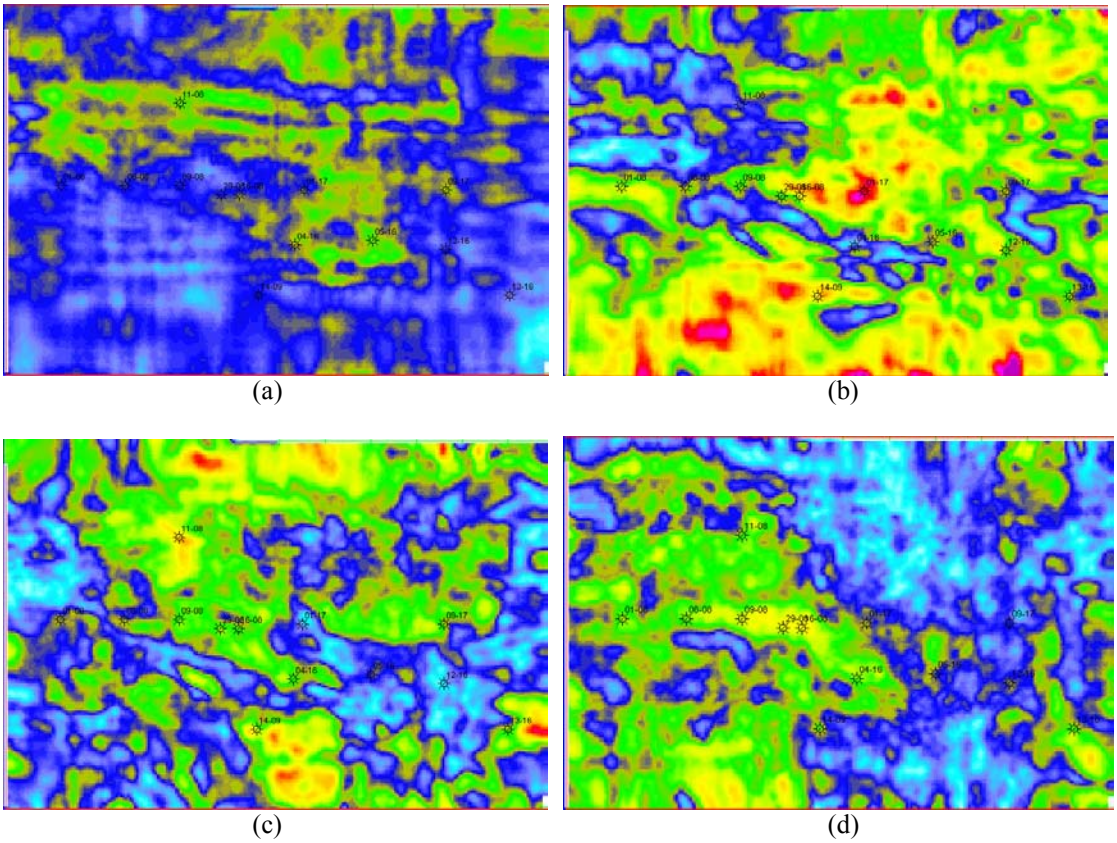


Figure 2.8. The dominant frequency along a sliding window of the seismic section shown in Figure 2.1.

To compute a horizon-based attribute based on windowing the data, Partyka et al. (1999) suggest computing the Fourier spectrum over a window from the complete seismic volume and then displaying the amplitude spectrum in frequency slices. This is referred to as the spectral decomposition method, and will show spatial stratigraphic detail at a zone of interest.

Figure 2.9 shows the spectral decomposition between the two events picked at times of 1030 and 1080 on the seismic section of Figure 2.2, over the complete seismic volume. In this figure I have shown slices at dominant frequencies of 16, 32, 48, and 64 Hz, respectively.



(a) (b)

(c) (d)

Figure 2.9. The spectral decomposition of the full seismic volume shown in Figure 2.1, where (a) shows the 16 Hz slice, (a) shows the 32 Hz slice, (a) shows the 48 Hz slice, and (a) shows the 64 Hz slice.

2.4 Recursive attributes

Recursive attributes are computed by applying an operator recursively along the trace. If we apply a recursive difference operator, the output is the first derivative of the seismic trace. In practice, this is estimated by taking the difference between adjacent samples, which can be written:

$$d_i = s_i - s_{i-1}, \tag{2.5}$$

where s_i, $i = 1,\ldots N,$ are the seismic samples. Figure 2.10 shows the first derivative estimate of the seismic section from Figure 2.2. Notice the extra detail shown in this figure when compared to Figure 2.2, and also the 90-degree phase shift.

To illustrate this, consider the effect of the derivative on a single frequency component, written as

$$\frac{de^{i\omega t}}{dt} = i\omega e^{i\omega t},\ \omega = 2\pi f\ . \tag{2.6}$$

In equation (2.6), multiplication by i implies a +90 degree phase shift to the original seismic trace, and multiplication by ω implies a "ramping-up" of the amplitude spectrum.

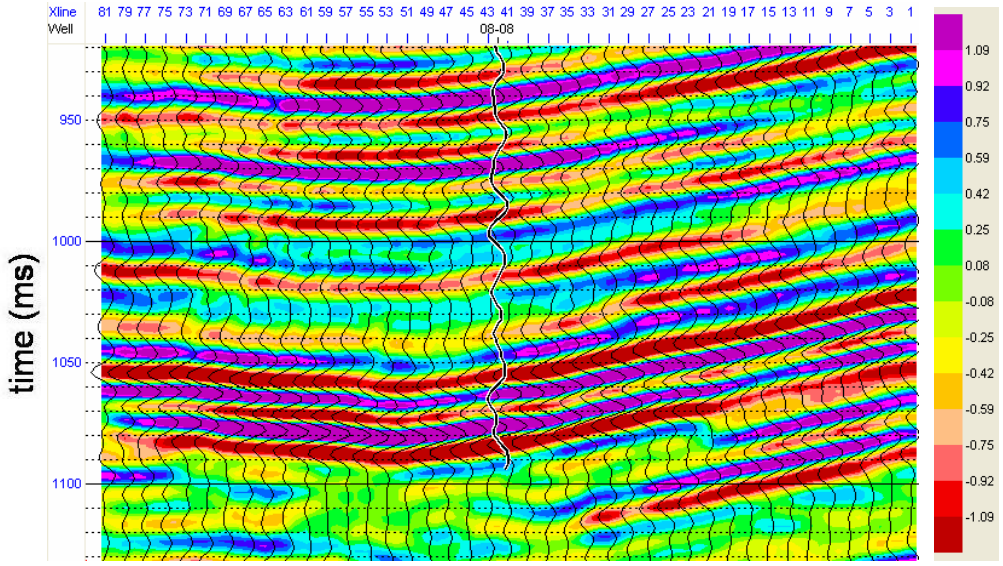


Figure 2.10. The first derivative of the seismic section shown in Figure 2.2.

If the difference operation is applied twice to the input, we estimate the second derivative of the seismic trace, given in equations as

$$d2_i = d_i - d_{i-1} = s_i - 2s_{i-1} + s_{i-2}\ . \tag{2.7}$$

Figure 2.11 shows the second derivative of the seismic section from Figure 2.2. Notice the extra detail shown in this figure when compared to Figure 2.9, and also the 180-degree phase shift. Again, this can be seen by applied the second derivative operation to a single frequency, or

$$\frac{d^2e^{i\omega t}}{dt^2} = i\omega(i\omega e^{i\omega t}) = -\omega^2 e^{i\omega t}\ . \tag{2.8}$$

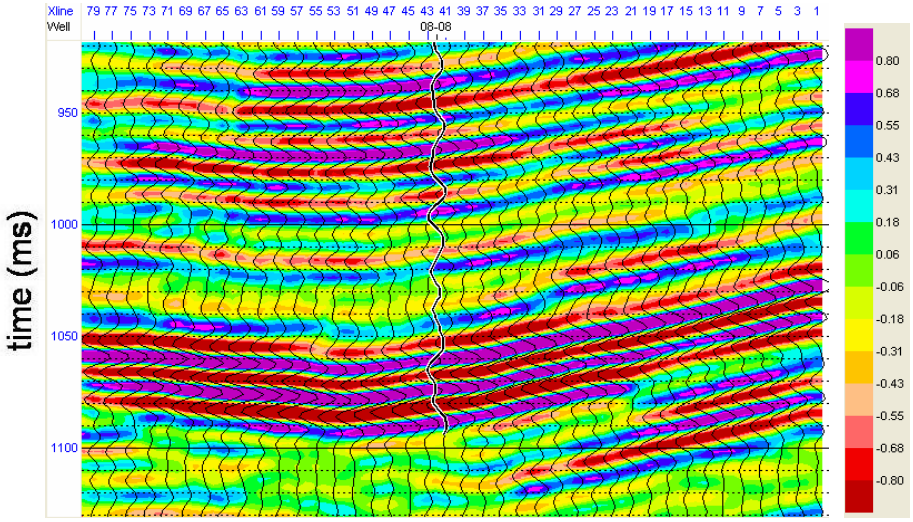


Figure 2.11. The second derivative of the seismic section shown in Figure 2.2.

To compute other attributes, we can also apply the first and second derivative operations to the amplitude envelope of the seismic trace.

If we apply a recursive sum to the seismic trace, the output is the integral of the trace, given as

$$I_n = \sum_{i=1}^{n} s_i \,, \tag{2.9}$$

where the n^{th} integrated value is the sum of all of the samples from the *first* to the n^{th}. To remove any bias, the smoothed integrated trace is subtracted from the integrated trace. Figure 2.12 shows the integrated seismic section of Figure 2.2, where I have used a 50 ms running smoother to remove the bias. Notice that there is less high-frequency content in this figure than in the seismic section of Figure 2.2, and also there is a -90-degree phase shift. To illustrate this, consider the effect of integration on a single frequency component, written as

$$\int e^{i\omega t} dt = \frac{1}{i\omega} e^{i\omega t} = \frac{-i}{\omega} e^{i\omega t} \,. \tag{2.10}$$

In equation (2.10), multiplication by $-i$ implies a -90 degree phase shift to the original seismic trace, and division by ω implies a "ramping-down" of the amplitude spectrum. Integration can also be thought of as a type of seismic inversion (Lindseth, 1979).

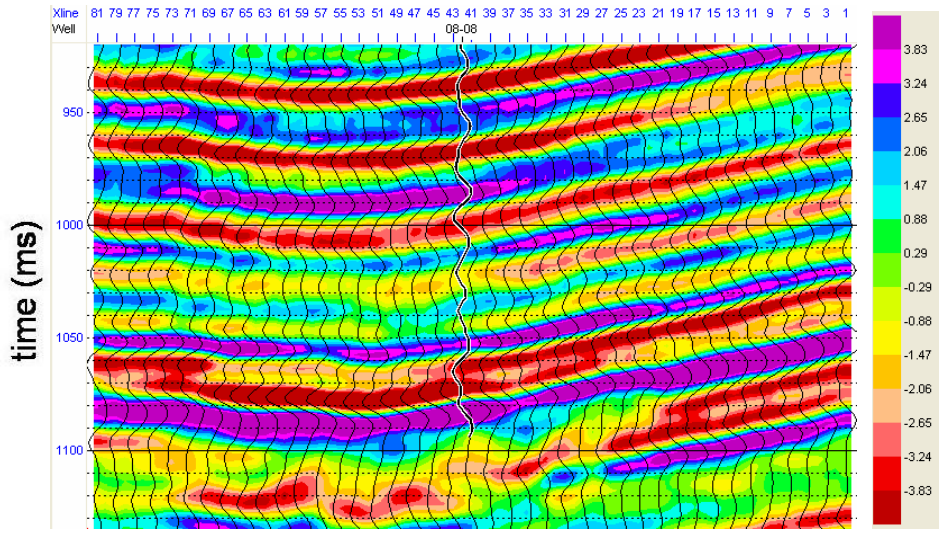


Figure 2.12. The integration of the seismic section shown in Figure 2.2.

As with the derivative operation, integration can also be applied to the amplitude envelope of the seismic trace.

2.5 Bandpass attributes

Bandpass attributes are simply narrow-band filter slices of the seismic data. By applying a narrow-band filter, we isolate signal components within particular frequency bands of the seismic data. To ensure that the filter slices can be added together to give the original data, trapezoidal filters are designed with overlapping 5-Hz slopes. For example, the first two filter slices could be 5/10-15/20 and 15/20-25/30, where the first number is the low-cut value, the second is the low-pass value, the third is the high-pass value and the fourth is the high-cut value, all given in Hertz.

Figures 2.13 and 2.14 show low frequency (5/10-15/20) and high frequency (55/60-65/70) bandpass filter slices, respectively, applied to the seismic data of Figure 2.2.

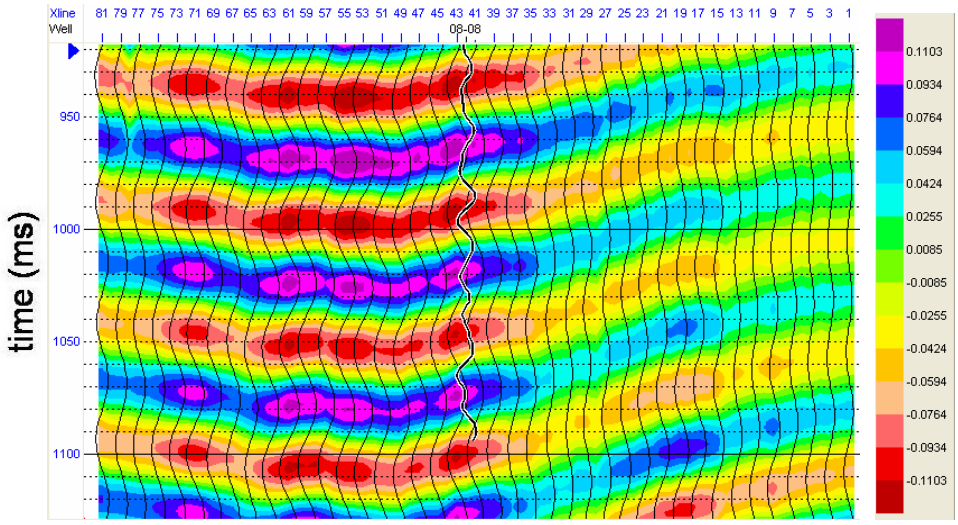


Figure 2.13: A low-frequency filter slice (5/10-15/20) of the seismic section shown in Figure 2.2.

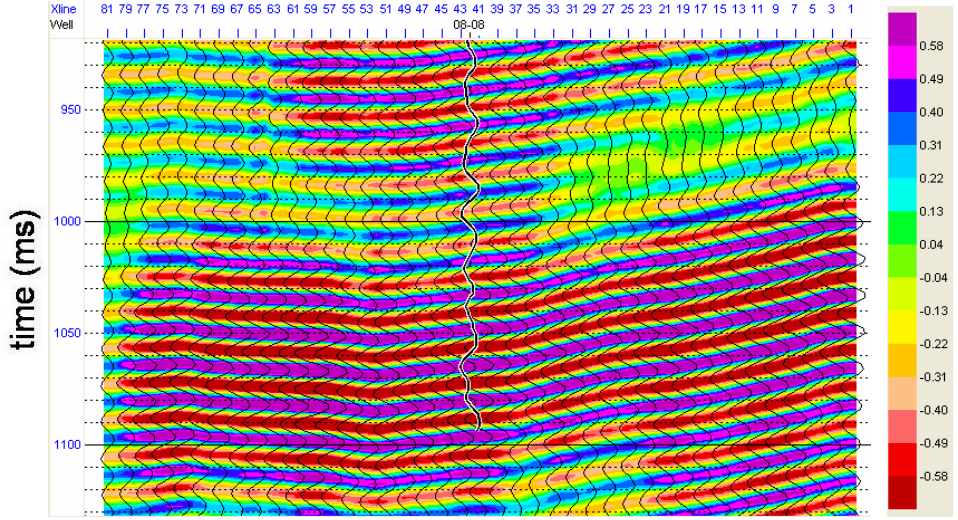


Figure 2.14: A high-frequency filter slice (55/60-65/70) of the seismic section shown in Figure 2.2.

As expected, Figure 2.13 and 2.14 show a lot of "ringing" at both the low and high ends of the signal spectrum. However, both sections contain valuable information about the seismic data that will be used by the statistical prediction techniques that will be explored in subsequent chapters.

2.6 Coherency attributes

Using a window of traces in a 3D seismic volume, information about discontinuities within the subsurface of the earth, such as faults and fractures, can be determined. The resulting attributes are called coherency attributes. There have been three generations of algorithms proposed for extracting the coherency attribute. Bahorich and Farmer (1995) published the first widely used algorithm, termed cross-correlation coherency. Their method involved cross-correlating each trace with its in-line and cross-line neighbour and then combining the two results after energy normalization. Marfurt et al. (1998) published the second-generation method, based on semblance. Most recently, Gersztenkorn and Marfurt (1999) published the third-generation approach based on eigenstructure methods. The latter two methods can be summarized as follows. Let D be a matrix containing a suite of J seismic traces, each of N samples, which can be written

$$D = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1J} \\ d_{21} & d_{22} & \cdots & d_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ d_{N1} & d_{N2} & \cdots & d_{NJ} \end{bmatrix}. \tag{2.11}$$

Using the values in D, the semblance coherency estimate can be defined as

$$S_C = \frac{\sum_{n=1}^{N}\left[\sum_{j=1}^{J} d_{nj}\right]^2}{J\sum_{n=1}^{N}\sum_{j=1}^{J}\left(d_{nj}\right)^2}. \tag{2.12}$$

.

To derive the eigenstructure estimate, we require the covariance of D, which can be written as

$$C = D^T D. \tag{2.13}$$

The eigenstructure coherence estimate is then found by the expression

$$E_C = \frac{\lambda_1}{\sum_{j=1}^{J} \lambda_j}, \tag{2.14}$$

where λ_j are the J eigenvalues of the matrix C, ordered from largest (λ_1) to smallest (λ_J). As shown by Gersztenkorn and Marfurt (1999), the semblance estimate can also be written in terms of the covariance matrix and its eigenvalues, as

$$S_C = \frac{u^T C u}{\sum_{j=1}^{J} \lambda_j}, \tag{2.15}$$

where $u = \frac{1}{\sqrt{J}} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$. The numerator of equation (2.15) is often called a variance probe, and is maximized when $u^T C u = \lambda_1$, the maximum eigenvalue of C. Thus, the eigenstructure approach to coherency can be seen to be the optimum implementation of the semblance coherency method.

Marfurt and Kirlin (2000) extended the concept of coherency to define other multi-trace attributes such as dip, amplitude gradient, and curvature. In this study I will not use coherence attributes. However, the concepts of the covariance matrix, the variance probe and eigenvalues and eigenvectors (and the related topic of principal components) will be addressed in later chapters.

2.7 AVO Attributes

If we consider an incident P-wave striking the boundary between two elastic half-spaces at an angle θ, as shown in Figure 2.15, mode conversion results in the creation of reflected and transmitted P and SV-waves (Aki and Richards, 2002).

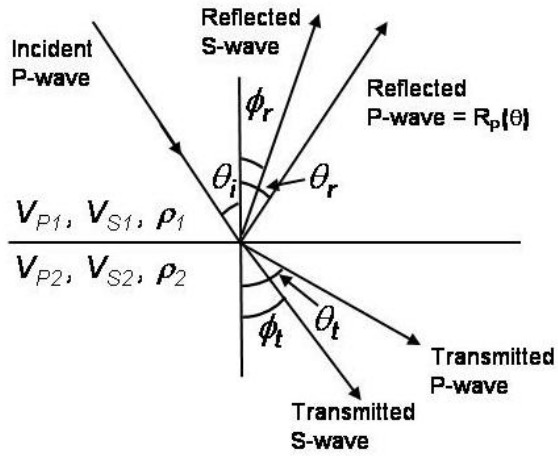


Figure 2.15. Mode conversion from an incident P-wave arrival at the interface between two elastic media.

To derive the amplitudes of the reflected and transmitted waves, note that the stresses and displacements across the boundary are continuous. This gives four equations in four unknowns, referred to as the Zoeppritz equations (Zoeppritz, 1919), and written as

$$\begin{bmatrix} R_P \\ R_S \\ T_P \\ T_S \end{bmatrix} = \begin{bmatrix} -\sin\theta_1 & -\cos\phi_1 & \sin\theta_2 & \cos\phi_2 \\ \cos\theta_1 & -\sin\phi_1 & \cos\theta_2 & -\sin\phi_2 \\ \sin 2\theta_1 & \frac{V_{P1}}{V_{S1}}\cos 2\phi_1 & \frac{\rho_2 V_{S2}^2 V_{P1}}{\rho_1 V_{S1}^2 V_{P2}}\cos 2\phi_1 & \frac{\rho_2 V_{S2} V_{P1}}{\rho_1 V_{S1}^2}\cos 2\phi_2 \\ -\cos 2\phi_1 & \frac{V_{S1}}{V_{P1}}\sin 2\phi_1 & \frac{\rho_2 V_{P2}}{\rho_1 V_{P1}}\cos 2\phi_2 & -\frac{\rho_2 V_{S2}}{\rho_1 V_{P1}}\sin 2\phi_2 \end{bmatrix}^{-1} \begin{bmatrix} \sin\theta_1 \\ \cos\theta_1 \\ \sin 2\theta_1 \\ \cos 2\phi_1 \end{bmatrix} \quad (2.16)$$

where R_P is the *P*-wave reflection coefficient as a function of *P*-wave reflection angle θ_1, R_S is the *SV*-wave reflection coefficient as a function of *S*-wave reflection angle ϕ_1, T_P is the *P*-wave transmission coefficient as a function of *P*-wave refraction angle θ_2, T_S is the *SV*-wave transmission coefficient as a function of *S*-wave refraction angle ϕ_2, and V_{Pi}, V_{Si}, and ρ_i are the *P*-wave velocity, *S*-wave velocity, and density in the first (i = 1) and

second (i = 2) layers, respectively. We can derive the reflected and transmitted angles from the input angle using Snell's law, which is written

$$p = \frac{\sin\theta_i}{V_{P1}} = \frac{\sin\theta_r}{V_{P1}} = \frac{\sin\theta_t}{V_{P2}} = \frac{\sin\phi_r}{V_{S1}} = \frac{\sin\phi_t}{V_{S2}}, \tag{2.17}$$

where p is called the ray parameter.

As shown by Bortfeld (1961), Richards and Frasier (1976), and Aki and Richards (2002), equation (2.16) can be linearized by expanding it as a Taylor series and keeping only first-order terms (i.e. neglecting all squares and products). The linearized P-wave reflection coefficient as a function of angle (or offset) is written

$$R_P(\theta) = \left[\frac{1}{2\cos^2\theta}\right]\frac{\Delta V_P}{V_P} - \left[4\gamma\sin^2\theta\right]\frac{\Delta V_S}{V_S} + \left[0.5 - 2\gamma\sin^2\theta\right]\frac{\Delta\rho}{\rho}, \tag{2.18}$$

where $\rho = \frac{\rho_2 + \rho_1}{2}$, $V_P = \frac{V_{P2} + V_{P1}}{2}$ and $V_S = \frac{V_{S2} + V_{S1}}{2}$ are the average density and velocity terms, $\Delta\rho = \rho_2 - \rho_1$, $\Delta V_P = V_{P2} - V_{P1}$ and $\Delta V_S = V_{S2} - V_{S1}$ are the density and velocity differences, $\gamma = \left(\frac{V_S}{V_P}\right)^2$, and $\theta = \frac{\theta_i + \theta_t}{2}$ is the average angle. Another equivalent form of equation (2-18), derived initially by Wiggins et al. (1982), and expanded by Shuey (1985), is written

$$R_P(\theta) \cong A + B\sin^2\theta + C\tan^2\theta\sin^2\theta, \tag{2.19}$$

where $A = R_{P0} = \frac{1}{2}\left[\frac{\Delta V_P}{V_p} + \frac{\Delta\rho}{\rho}\right]$, $B = \frac{1}{2}\frac{\Delta V_P}{V_p} - 4\gamma\frac{\Delta V_S}{V_S} - 2\gamma\frac{\Delta\rho}{\rho}$, and $C = \frac{1}{2}\frac{\Delta V_P}{V_p}$.

(It should be noted that Shuey (1985) added the third term C, although his second term, B, was a function of Vp, Poisson's ratio, and density).

A second equivalent version of equation (2.18) was derived by Fatti et al. (1994), based on an earlier equation by Smith and Gidlow (1987), and expresses R_P as a function of the zero-offset P and S-wave reflectivity. Their equation is written

$$R_P(\theta) \cong \left[1 + \tan^2\theta\right]R_{P0} - \left[8\gamma \sin^2\theta\right]R_{S0} - \left[\frac{1}{2}\tan^2\theta - 2\gamma\sin^2\theta\right]\frac{\Delta\rho}{\rho}, \qquad (2.20)$$

where $R_{S0} = \frac{1}{2}\left[\frac{\Delta V_S}{V_S} + \frac{\Delta\rho}{\rho}\right]$.

Note that equations (2.18) through (2.20) can all be written in the form

$$R_P(\theta) \cong f_1(\theta)P_1 + f_2(\theta,\gamma)P_2 + f_3(\theta,\gamma)P_3, \qquad (2.21)$$

where f_1, f_2, and f_3 are functions of θ and sometimes γ (for example, f_3 is a function of γ in the first and third equations, but not the second), and P_1, P_2, and P_3 are parameters that are dependent on some combination of $\Delta V_P/V_P$, $\Delta V_S/V_S$, and $\Delta\rho/\rho$. Thus, the parameter terms can be extracted by performing a weighted stack of the prestack seismic amplitudes. Let us assume that there are N traces in a CMP gather. We can then extract N amplitudes at time t, as shown in Figure 2.16, and write any one of the previous equations as a set of N equations with three unknowns, or:

$$\begin{aligned}
R_P(\theta_1) &= f_1(\theta_1)P_1 + f_2(\theta_1,\gamma)P_2 + f_3(\theta_1,\gamma)P_3 \\
R_P(\theta_2) &= f_1(\theta_2)P_1 + f_2(\theta_2,\gamma)P_2 + f_3(\theta_2,\gamma)P_3 \\
&\vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \\
R_P(\theta_N) &= f_1(\theta_N)P_1 + f_2(\theta_N,\gamma)P_2 + f_3(\theta_N,\gamma)P_3
\end{aligned} \qquad (2.22)$$

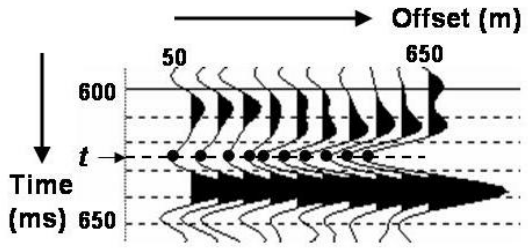


Figure 2.16. Extraction of AVO attributes using the amplitudes from a seismic gather picked at time t.

These equations can be written in matrix form as

$$\begin{bmatrix} R_P(\theta_1) \\ R_P(\theta_2) \\ \vdots \\ R_P(\theta_N) \end{bmatrix} = \begin{bmatrix} f_1(\theta_1) & f_2(\theta_1,\gamma) & f_3(\theta_1,\gamma) \\ f_1(\theta_2) & f_2(\theta_2,\gamma) & f_3(\theta_2,\gamma) \\ \vdots & \vdots & \vdots \\ f_1(\theta_N) & f_2(\theta_2,\gamma) & f_3(\theta_N,\gamma) \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_2 \end{bmatrix}, \qquad (2.23)$$

or $R = MP$, with solution

$$P = (M^T M + \lambda I)^{-1} M^T R, \tag{2.24}$$

where λ is a prewhitening factor and I is the *3 x 3* identity matrix.

Since we extract the parameters as a function of offset rather than angle, as shown in Figure 2.16, we need a relationship between offset and angle. As shown by Walden (1991) this relationship can be written

$$\sin\theta = \frac{XV_{INT}}{tV_{RMS}^2}, \tag{2.25}$$

where X = offset, t = two-way traveltime, V_{INT} = interval velocity, and V_{RMS} = RMS velocity.

As an example of computing the AVO attributes just discussed, Figure 2.17 shows a subset of the common-depth-point gathers used to create the stack in Figure 2.2.

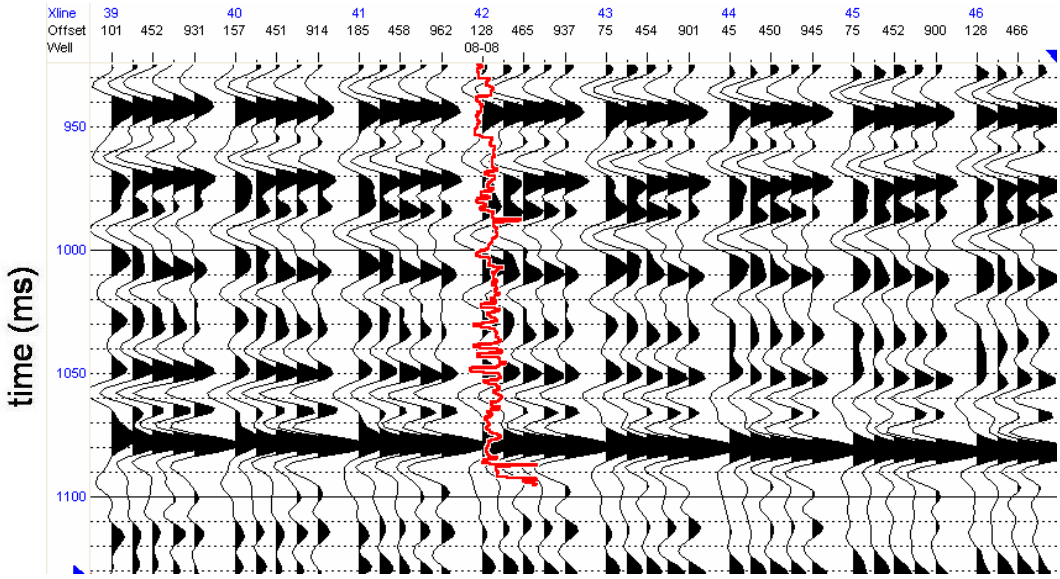


Figure 2.17. The seismic gathers that were used to create the stacked section in Figure 2.2, with the P-wave sonic log from well 08-08 spliced in at the intersecting location.

Using the techniques just described, estimates of the R_{P0} and R_{S0} AVO attributes are created. The R_{P0} estimate is in Figure 2.18 and the R_{S0} estimate is in Figure 2.19.

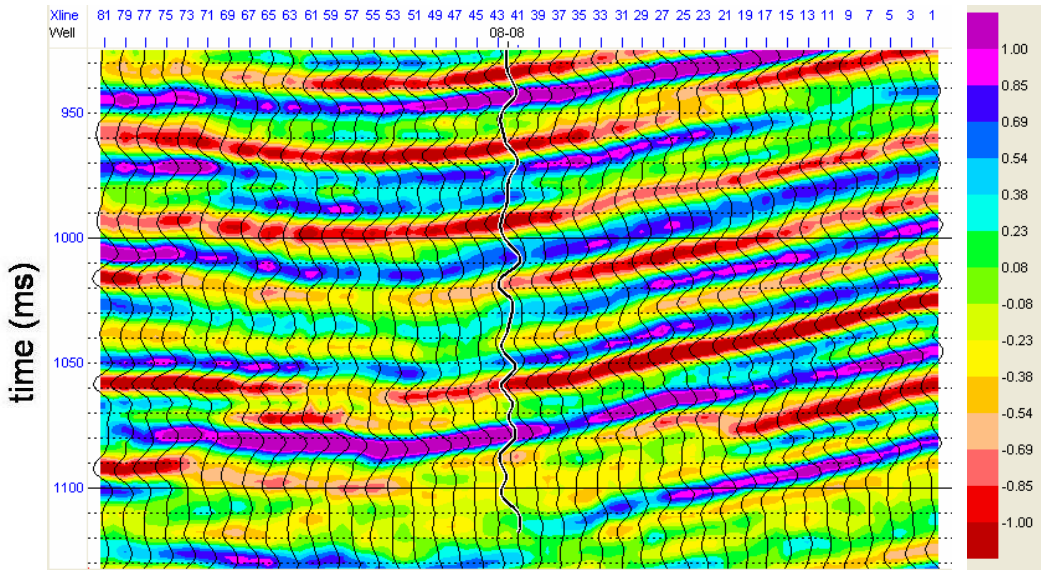


Figure 2.18. Extraction of the R_{P0} attribute from the gathers of Figure 2.17.

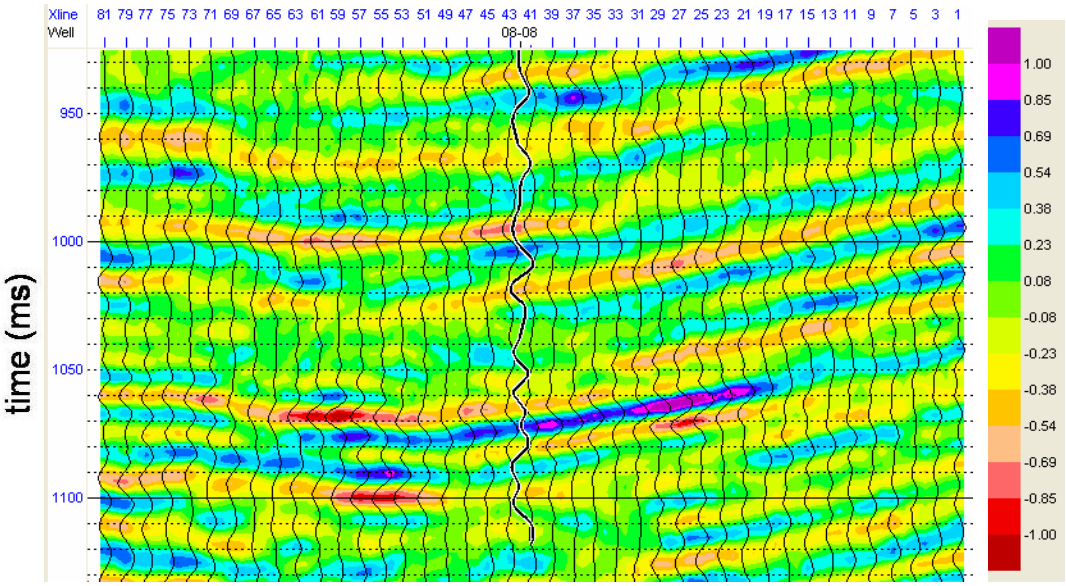


Figure 2.19. Extraction of the R_S attribute from the gathers of Figure 2.17.

These two reflectivity estimates will be used as the input to inversion in the last section in this chapter.

2.8 Model-based attributes

In this section, I will discuss model-based attributes, in which a geological model is included with some transform of the seismic trace to create an attribute. This is a deterministic approach to reservoir delineation, because we determine the subsurface of the earth by assuming that we know the underlying model that fits the seismic data. I will start by introducing the most common model, which is based on the convolution of a wavelet with the earth's reflectivity.

2.8.1 The convolutional model

The noise-free convolutional model for the seismic trace can be written

$$s(t) = w(t) * r(t) \tag{2.26}$$

where $s(t) =$ the seismic trace, $w(t) =$ the seismic wavelet, $r(t) =$ the earth's reflectivity, and * denotes convolution. The mathematics of convolution are discussed in Appendix 3, where it is shown that convolution can be written as the matrix multiplication

$$S = WR, \tag{2.27}$$

where $S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{N+M-1} \end{bmatrix}$, $R = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_M \end{bmatrix}$, *and* $W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ w_2 & w_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ w_N & \cdots & w_2 & w_1 \\ 0 & w_N & \cdots & w_2 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & w_N \end{bmatrix}$.

In equation (2.27), note that the reflectivity is of length *M* samples, the wavelet is of length *N* samples, and the output seismic trace is of length *N+M-1* samples. (This implies that *W* has *M* columns and *N+M-1* rows).

As a simple example in which the length of both the reflectivity and the wavelet is two samples, the result is a seismic trace of three samples given as

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} w_1 & 0 \\ w_2 & w_1 \\ 0 & w_2 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} w_1 r_1 \\ w_2 r_1 + w_1 r_2 \\ w_2 r_2 \end{bmatrix}. \tag{2.28}$$

The matrix formulation of convolution is similar to the Z-transform approach, which leads us to the frequency domain (Claerbout, 1976), where convolution is written:

$$S(f) = |S(f)| \exp(-i\phi_s) = W(f)R(f) = |W(f)|\,|R(f)| \exp(-i(\phi_w + \phi_r)), \tag{2.29}$$

where *S(f), W(f),* and *R(f)* are the Fourier transforms of the trace, wavelet, and reflectivity, respectively, | | denotes amplitude spectrum, and ϕ denotes phase spectrum. For the complex product, equation (2.29) tells us that the amplitude spectra are multiplied, and the phase spectra are added. Equations (2.26) through (2.29) are illustrated graphically in Figure 2.20, which shows the convolution of a ninety-degree phase rotated Ricker wavelet with a well-log-derived reflectivity.

In equation (2.26), the reflectivity can be written:

$$r_i = \frac{Z_{i+1} - Z_i}{Z_{i+1} + Z_i}, \tag{2.30}$$

where r_i is the reflection coefficient at the i^{th} geological interface, and Z_i is the impedance at the i^{th} geological layer, after integrating to time. As will be discussed shortly, there are three different forms that impedance can take: acoustic impedance $Z_P = \rho V_P$, shear impedance $Z_S = \rho V_S$, and elastic impedance $Z_E = V_P^{(1+\tan^2\theta)} V_S^{(-8K\sin^2\theta)} \rho^{(1-K\sin^2\theta)}$, where V_P is the *P*-wave velocity, V_S is the shear wave velocity, ρ is the density, θ is average angle of incidence and $K = V_S^2 / V_P^2$. By inverting equation (2.30), the impedance can be recursively derived from the reflectivity. The recursive inversion equation is written

$$Z_{i+1} = Z_i \frac{1 + r_i}{1 - r_i}, \tag{2.31}$$

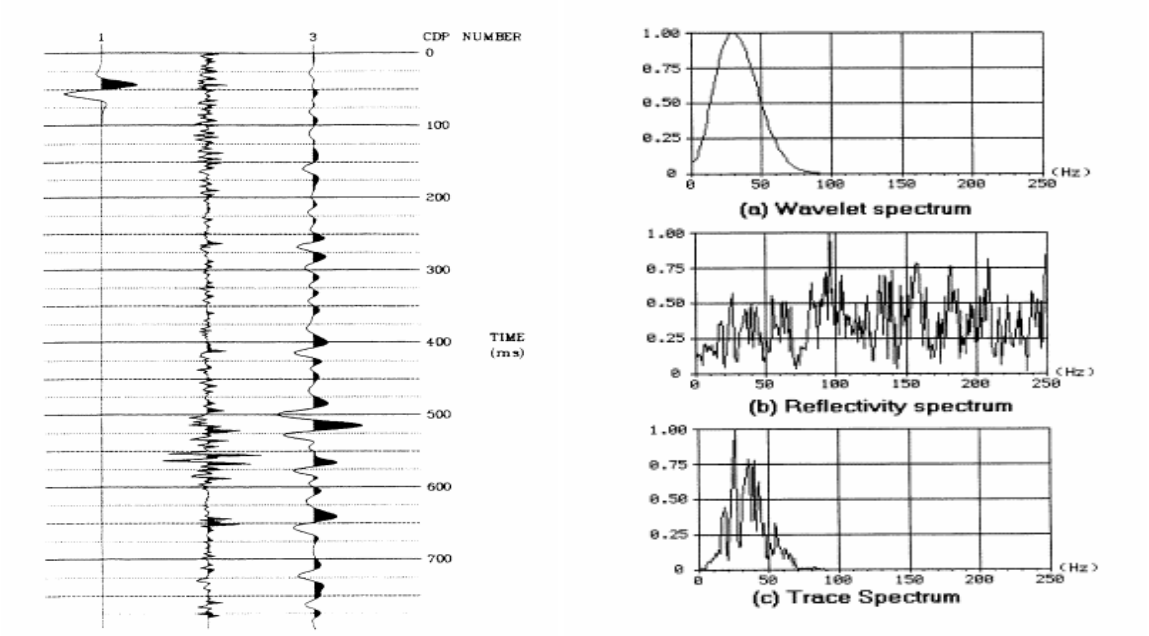


Figure 2.20. An illustration of convolution using a Ricker wavelet and well-log derived reflectivity, where (a) shows convolution in the time domain (left to right: the wavelet, reflectivity, and seismic trace), and (b) shows the convolution in the frequency domain (top to bottom: the amplitude spectrum of the wavelet, the amplitude spectrum of the reflectivity, and the amplitude spectrum of the seismic trace), (Russell, 1988).

Equation (2.31) can also be thought of as integration. To understand this, consider the effect of inverting M reflection coefficients, which gives

$$Z_M = Z_1 \prod_{i=2}^{M} \left[\frac{1+r_i}{1-r_i} \right]. \tag{2.32}$$

Taking the logarithm of equation 2.32, we get

$$\ln(Z_M) = \ln(Z_1) + \sum_{i=2}^{M} 2\left[r_i + \frac{r_i^3}{3} + \frac{r_i^5}{5} + \ldots \right], \tag{2.33}$$

where I have made use of the series expansion $\ln\left(\frac{1+x}{1-x}\right) = 2\left[x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots \right]$. Since the reflection coefficients r_i are typically of the order of 0.1 or less, we can drop all terms greater than first order in the summation, so that equation (2.33) can be rewritten as

$$L_M = 2\sum_{i=2}^{M} r_i \,, \tag{2.34}$$

where $L_M = ln(Z_M) - ln(Z_1)$. Finally, note that equation (2.34) can also be written in convolutional form as

$$L = HR, \tag{2.35}$$

where $L = \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_{N+M-1} \end{bmatrix}$, $R = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_M \end{bmatrix}$, and $H = \begin{bmatrix} 2 & 0 & \cdots & 0 \\ 2 & \ddots & \ddots & \vdots \\ 2 & \cdots & 2 & 0 \\ 2 & \cdots & 2 & 2 \end{bmatrix}$.

Except for a factor of 2, and that the operation is being applied to the reflectivity rather than the seismic trace, notice that equation (2.34) is identical to equation (2.9). This suggests that the operation of recursive inversion is equivalent to trace integration if we replace the seismic trace with the reflectivity. From equation (2.10) this also tells us that recursive inversion of the reflectivity will apply both a -90-degree phase shift and a ω^{-1} frequency-domain scaling (Russell and Lindseth, 1982).

2.8.2 Acoustic impedance inversion

Acoustic impedance inversion was developed in the 1970's by making use of equation 2.30, and substituting the seismic trace for the reflectivity (Lindseth, 1979). However, as shown in Figure 2.20, the seismic trace is bandlimited by the seismic wavelet, and the resulting inverted impedance will also be bandlimited. The lack of the high frequencies will remove detail from the final result. The loss of the low frequencies will remove the trend from the resulting pseudo-impedance log. The approach proposed by Lindseth (1979) to recover the low frequency component is to create an impedance model from well logs, apply a high-cut filter to this model to create a low frequency trend, and then add back the trend to the pseudo-logs. This technique, which I will call bandlimited inversion, is illustrated in Figure 2.21.

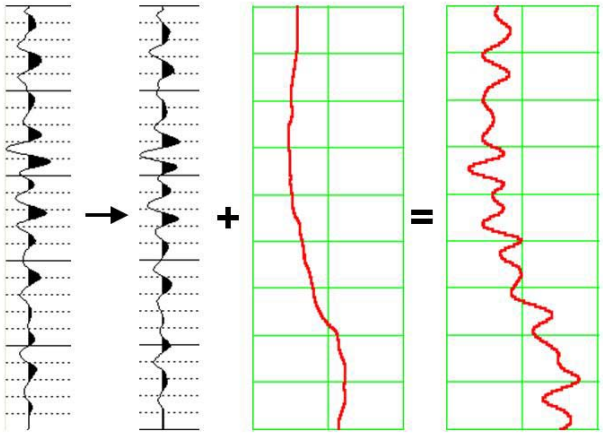


Figure 2.21. An illustration of the Lindseth (1979) inversion technique. Left to right, the original seismic trace is inverted using equation (2.30) to produce the second trace, and then added to the low frequency component to produce the final inverted log. (from Hampson and Russell, 1992).

Figure 2.22 shows the bandlimited inversion of the seismic line shown in Figure 2.2. To create the model for this inversion, I used the sonic and density logs from the wells shown in Figure 2.1, and stretched this model over the 3D volume using the seismic picks.

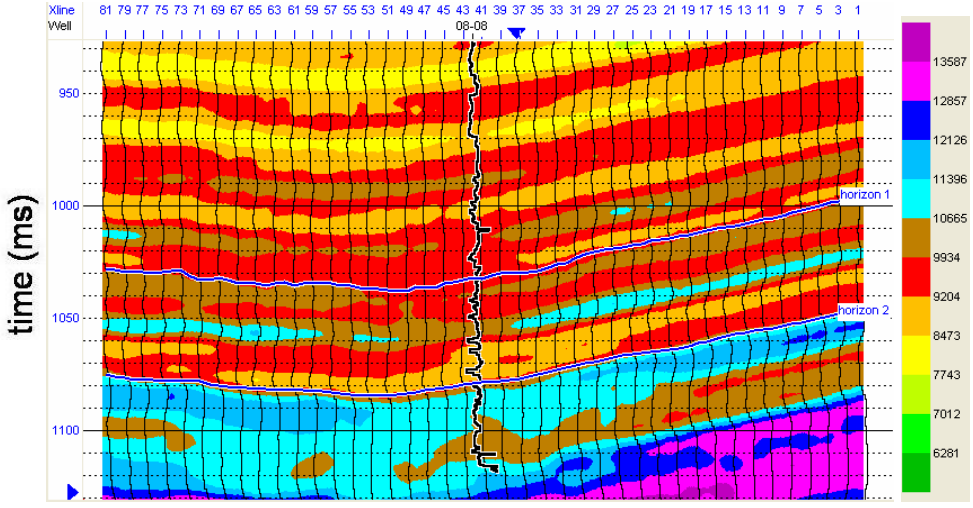


Figure 2.22. The bandlimited inversion of the seismic line shown in Figure 2.2.

Since the development of the bandlimited inversion technique, many more approaches to acoustic impedance inversion have been proposed. Two such approaches are sparse-spike inversion (Oldenburg et al., 1983) and model-based inversion (Hampson

and Russell, 1990). In sparse-spike inversion, an estimate of the sparse reflectivity is first extracted from the seismic data, using a linear programming technique in which frequency domain constraints are used to recover the low and high ends of the seismic spectrum lost due to the bandlimited seismic wavelet. Once the sparse reflectivity is extracted, the reflectivity is integrated and constrained by the initial model. The model is built as described for the bandlimited approach.

In model-based inversion, the initial geological model is perturbed so that the error between the synthetic generated from the perturbed model and the original seismic data is minimized. Mathematically, this problem is one of minimizing the error function J given by the following equation:

$$J = w_1 (L - HR)^T (L - HR) + (1 - w_1)(S - WR)^T (S - WR), \tag{2.36}$$

where L, H, S, W, and r are as defined in equations (2.27) and (2.35), and w_1 and w_2 are weights that satisfy the requirement that $w_1 + w_2 = 1$. In equation (2.36), the first term minimizes the error in the model and the second term minimizes the error in the seismic trace. Figure 2.23 shows the model-based inversion for the seismic line shown in Figure 2.2. Notice the extra detail that was not seen in Figure 2.22.

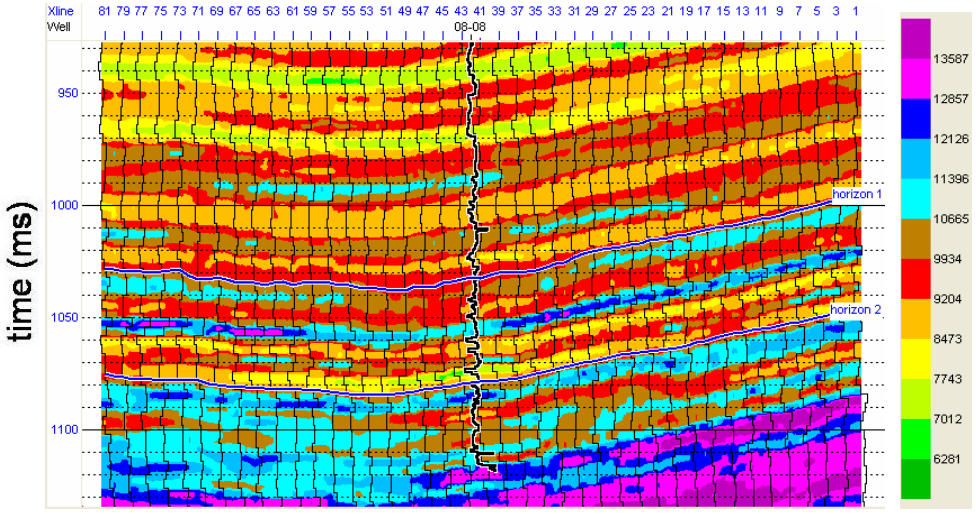


Figure 2.23. The model-based inversion of the seismic line shown in Figure 2.2.

2.8.3 Inversion of the AVO attributes

As I discussed in the section on AVO attributes, the weighted stack technique allows us to extract estimates of the *P* and *S* reflectivity sections, R_P and R_S. Inverting the R_P section should give us a better estimate of the P-impedance section. The inversion of the R_P section is shown in Figure 2.24. Figure 2.25 shows the inversion of the R_S attribute.

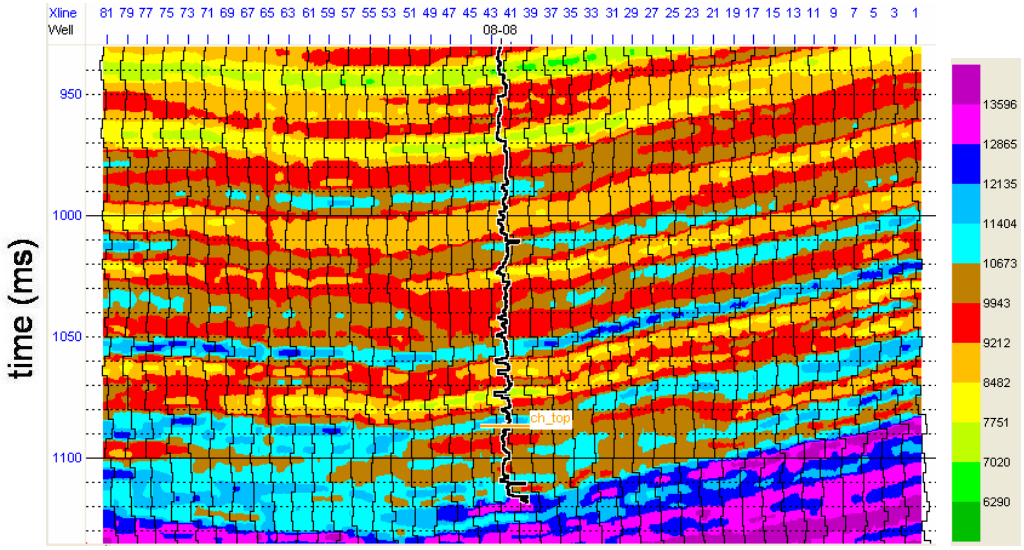


Figure 2.24. The model-based inversion of the R_{P0} attribute shown in Figure 2.18.

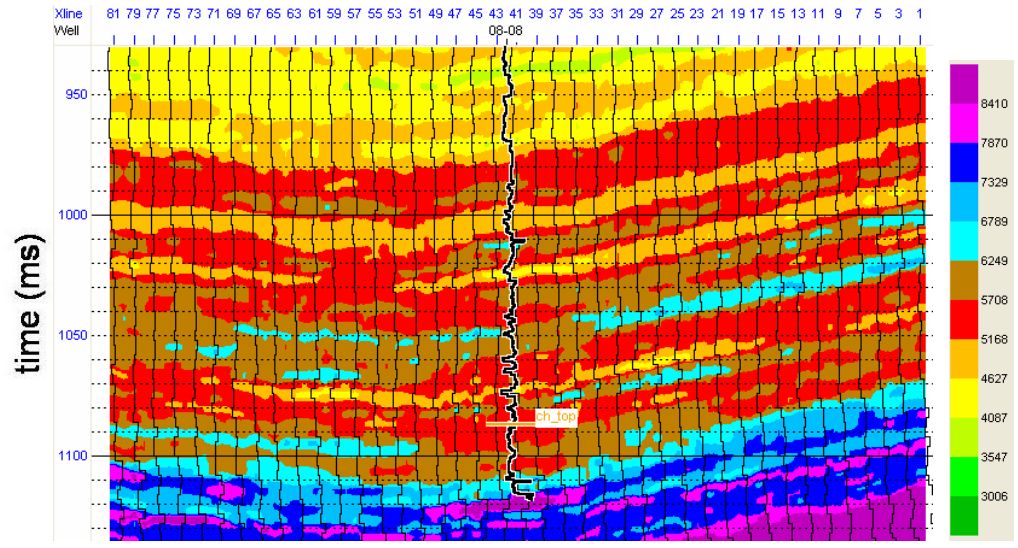


Figure 2.25. The model-based inversion of the R_{S0} attribute shown in Figure 2.18

2.9 Fluid-property discrimination with model-based attributes

2.9.1 Introduction

The last model-based attribute to be discussed in this chapter combines AVO and inversion attributes with rock physics theory in an attempt to extract information about the fluid content and rigidity of our reservoir rocks (Mavko et al., 1998). This topic has been addressed by Goodway et al. (1997), Hedlin (2000), Hilterman (2001) and Russell et al. (2003a).

To start our discussion, recall that the equations for P velocity in isotropic, non-porous media can be written in two different forms as

$$V_P = \sqrt{\frac{\lambda + 2\mu}{\rho}} = \sqrt{\frac{K + \frac{4}{3}\mu}{\rho}}, \tag{2.37}$$

where ρ is density, λ is the 1st Lamé parameter, μ is the 2nd Lamé parameter or shear modulus, and K is the bulk modulus, or the inverse of compressibility. For these two forms of the P-wave velocity to hold, we see that the relationship between K and λ can be written as

$$K - \lambda = \tfrac{2}{3}\mu. \tag{2.38}$$

The equation for S-wave velocity involves only density and shear modulus, and is written

$$V_s = \sqrt{\frac{\mu}{\rho}}. \tag{2.39}$$

When we turn our attention to porous, saturated rocks, the situation becomes more complex. Referring to Figure 2.26, note that a cube of porous rock can be characterized by four components: the rock mineral, the pore/fluid system, the dry-rock frame, or skeleton, and the saturated rock itself.

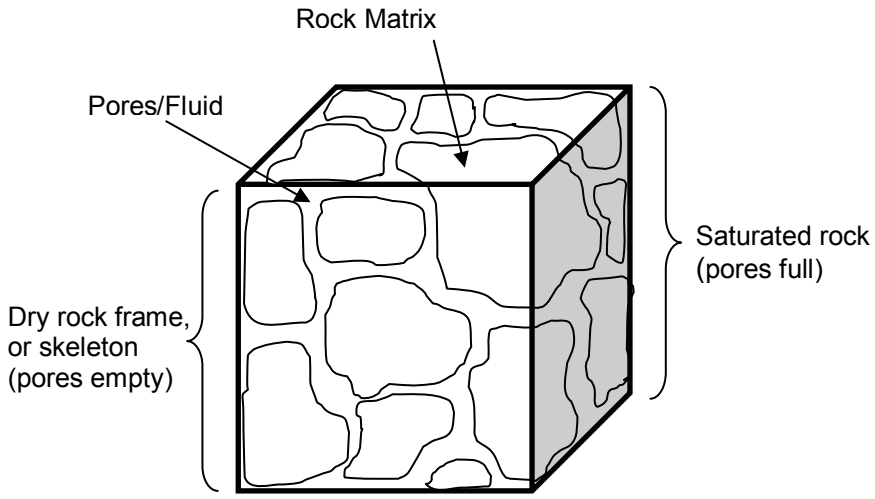


Figure 2.26. In Biot-Gassmann theory, a cube of rock is characterized by four components: the rock matrix, the pore/fluid system, the dry rock frame, and the saturated frame.

The density effects of the saturated rock shown in Figure 2.26 can be computed quite accurately with the volume average equation as

$$\rho_{sat} = \rho_m(1-\phi) + \rho_w S_w \phi + \rho_{hc}(1-S_w)\phi, \tag{2.40}$$

where ρ_{sat} is the total density value, ρ_m is the density of the rock matrix, ρ_w is the density of water (brine), ρ_{hc} is the density of the hydrocarbons, ϕ is the porosity of the rock, and S_w is the water saturation.

The velocity effects of the saturated rock were derived by Biot (1941) and Gassmann (1951) using apparently different approaches. However, as shown by Krief et al. (1990), their approaches are identical. Biot used the Lamé parameters and showed that (Krief et al, 1990) the saturated 1st Lamé parameter could be written

$$\lambda_{sat} = \lambda_{dry} + \beta^2 M\ , \tag{2.41}$$

where λ_{sat} and λ_{dry} are the 1st Lamé parameter for the saturated and dry frame, β is the Biot coefficient, or the ratio of the volume change in the fluid to the volume change in the formation when hydraulic pressure is constant, and M is the modulus, or the pressure needed to force water into the formation without changing the volume.

Gassmann used the bulk and shear moduli, and derived the following relationship (Krief et al, 1990):

$$K_{sat} = K_{dry} + \beta^2 M \,, \tag{2.42}$$

where K_{sat} and K_{dry} are the bulk modulii of the saturated and dry rock, and β and M are the same as in equation (2.41). By equating equations (2.41) and (2.42), and using equation (2.38), we find that

$$\mu_{sat} = \mu_{dry} \,, \tag{2.43}$$

which tells us that the shear modulus is unaffected by the pore fluid.

Using equations (2.41) and (2.42), we can now write the equation for P-wave velocity in the saturated case using the two separate forms given by

$$V_P = \sqrt{\frac{\lambda_{dry} + 2\mu + \beta^2 M}{\rho_{sat}}} = \sqrt{\frac{K_{dry} + \frac{4}{3}\mu + \beta^2 M}{\rho_{sat}}} \,, \tag{2.44}$$

or, more succinctly, as

$$V_P = \sqrt{\frac{s+f}{\rho_{sat}}} \,, \tag{2.45}$$

where f is a fluid/porosity term equal to $\beta^2 M$, and s is a dry-skeleton term which can be written either as $K_{dry} + \frac{4}{3}\mu$ or $\lambda_{dry} + 2\mu$. Note that $\mu = \mu_{sat} = \mu_{dry}$.

2.9.2 Extracting fluid and rigidity terms

Since I will be applying this method to seismic data, a practical limitation discussed in section 2.8.2 is that we can estimate only the P and S-wave impedances, Z_P and Z_S, where

$$Z_P = \rho V_P = \sqrt{\rho(f+s)}, \tag{2.46}$$

and

$$Z_S = \rho V_S = \sqrt{\rho\mu} \,. \tag{2.47}$$

By squaring the impedances we can then find a constant, c, such that

$$\rho f = Z_P^2 - cZ_S^2 = \rho(f + s - c\mu). \tag{2.48}$$

Referring to equation (2.44), note that c can be written in one of three ways:

$$c = \frac{\lambda_{dry}}{\mu} + 2 = \frac{K_{dry}}{\mu} + \frac{4}{3} = \left[\frac{V_P}{V_S}\right]^2_{dry}. \tag{2.49}$$

There are several approaches for computing the c term. The first is to estimate the dry-rock Poisson's ratio, σ_{dry}, noting that this is given by:

$$\sigma_{dry} = \frac{\left[\frac{V_P}{V_S}\right]^2_{dry} - 2}{2\left[\frac{V_P}{V_S}\right]^2_{dry} - 2} = \frac{c-2}{2c-2}. \tag{2.50}$$

Generally, the accepted value of σ_{dry} is on the order of *0.1*, which corresponds to a V_P/V_S ratio of *1.5*, or a c value of 2.25.

A second approach is to perform laboratory measurements. Murphy et al. (1993) measured the K_{dry}/μ ratio for clean quartz sandstones over a range of porosities and found that this value was, on average, equal to 0.9. This corresponds to a c value of 2.233. If the K_{dry}/μ value is rounded to 1.0, this implies a σ_{dry} of 0.125, and a corresponding c value of 2.333.

Thus, there are a range of values of c that depend on the particular reservoir being studied. Table 2.1 shows a range of c values and the range of respective elastic constants. The value of c in this table ranges from a high of 3, which implies that λ_{dry}/μ is equal to 1, to a low of 1 1/3, which implies that K_{dry}/μ is equal to *0* and that we have a negative Poisson's ratio. (This last value is therefore probably not physically meaningful).

Goodway et al. (1997) attribute all of the fluid effect to the λ term in equation (2.37), and thus derive their $\lambda\rho$ value as

$$\lambda\rho = Z_P^2 - 2Z_S^2. \tag{2.60}$$

Note that this means that *c* is equal to *2*, which implies a zero dry-rock Poisson's ratio.

$c = \left(\frac{V_P}{V_S}\right)^2_{dry}$	$\left(\frac{V_P}{V_S}\right)_{dry}$	σ_{dry}	$\frac{K_{dry}}{\mu}$	$\frac{\lambda_{dry}}{\mu}$
3.000	1.732	0.325	1.667	1.000
2.500	1.581	0.167	1.167	0.500
2.333	1.528	0.125	1.000	0.333
2.250	1.500	0.100	0.917	0.250
2.233	1.494	0.095	0.900	0.233
2.000	1.414	0.000	0.667	0.000
1.333	1.155	-1.000	0.000	-0.667

Table 2.1: A table of values for *c* ranging from 3 to 1 1/3 and showing the equivalent values for the various elastic constant ratios.

Hedlin (2000) incorporated the experimental results of Murphy et al., to arrive at the K_{dry}/μ ratio of *0.9* and a *c* value of *2.233*. Finally, Hilterman (2001) assumes that K_{dry}/μ is equal to *1.0*, which implies a *c* value of *2.333*.

2.9.3 Well-log example

Our well-log example comes from the Whiterose area of offshore eastern Canada. Figure 2.27 shows the V_S, V_P, density and porosity logs over the producing zone, overlain by a Cretaceous shale. There is 85 m of gas sand, 97 m of oil sand, and 95 m of wet sand. These well-log curves were converted to the equivalent ρf and ρs curves and crossplotted.

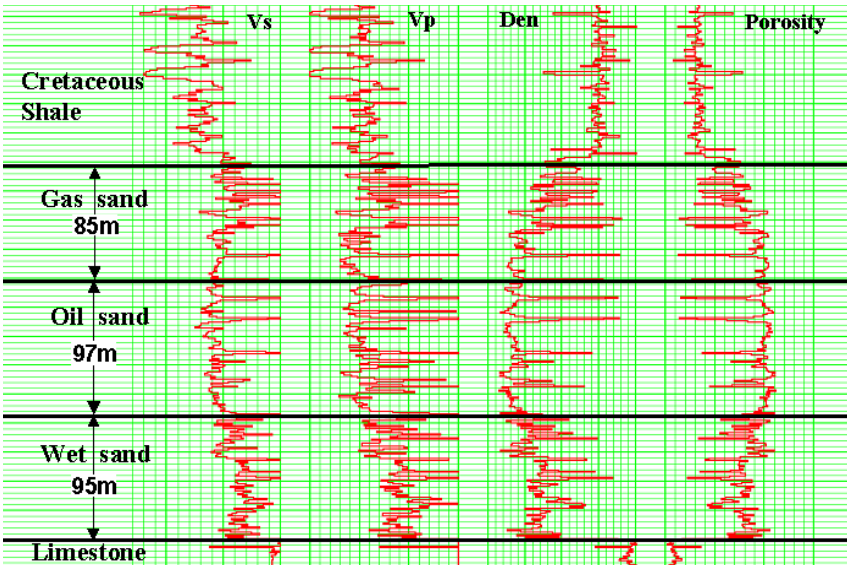
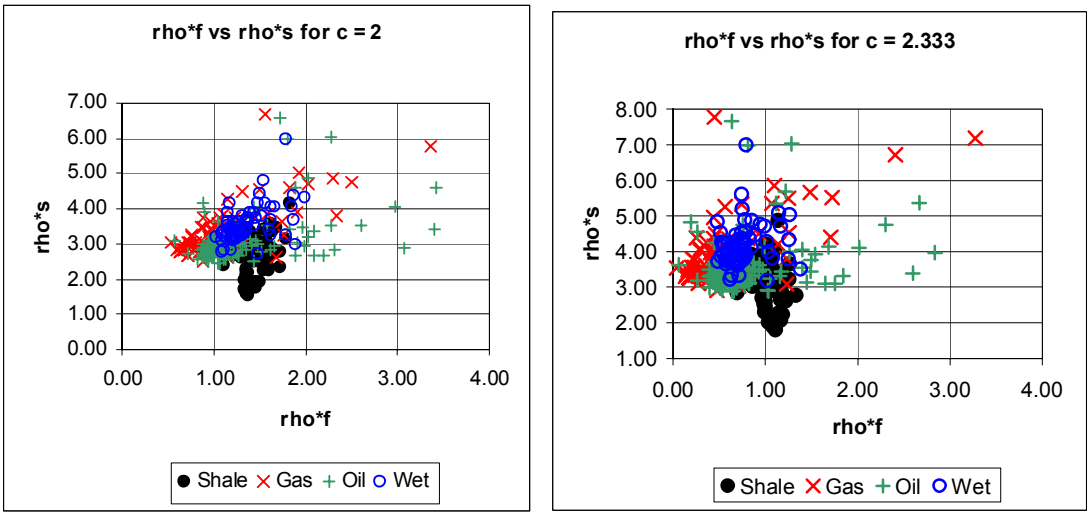


Figure 2.27: The V_S, V_P, ρ and porosity logs over the producing zone in the Whiterose L-08 well.

Figures 2.28(a) and (b) show crossplots of ρs versus ρf for values of c equal to *2.0* and *2.333*, respectively.



(a) (b)

Figure 2.28: A crossplot of ρf vs ρs for the Whiterose L-03 well for (a) c=2.0, and (b) c=2.333.

In Figure 2.28, each lithology and pore-fluid saturant is indicated by a different symbol. From our previous discussion, we wish to find the c value that produces the best ρf separation between the gas and non-gas-saturated zones. When choosing between the

two *c* values, notice that the ρf separation is almost the same. However, the better choice would appear to be *2.333* since the points with a higher ρs value show better separation. Also, for the *c* value of *2.333*, the cloud of gas points is closer to zero on the ρf axis.

2.9.4 A seismic example

Figures 2.29 and 2.30 show the *P* and *S*-wave impedance inversions for a shallow clastic gas sand in Alberta.

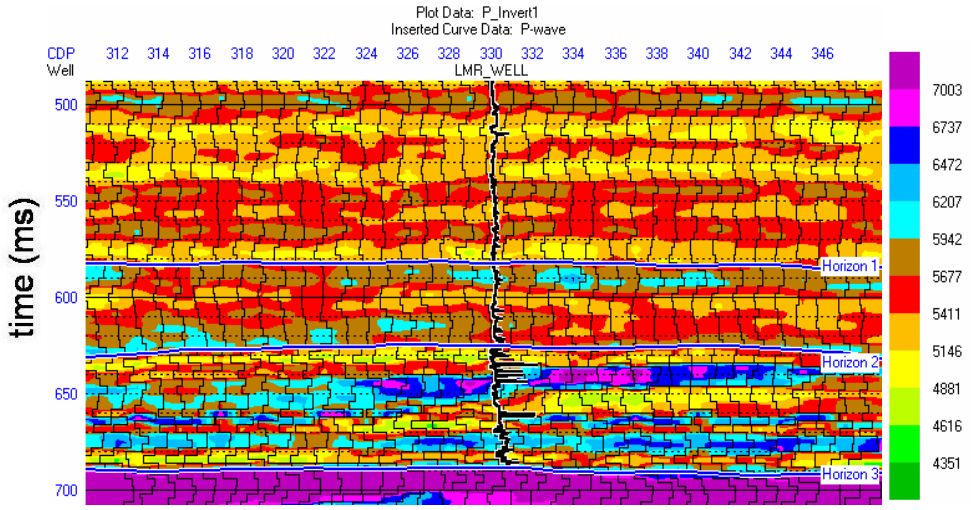


Figure 2.29: The P-wave impedance, Z_P, found by inverting the R_{P0} estimate of a gas sand in Alberta.

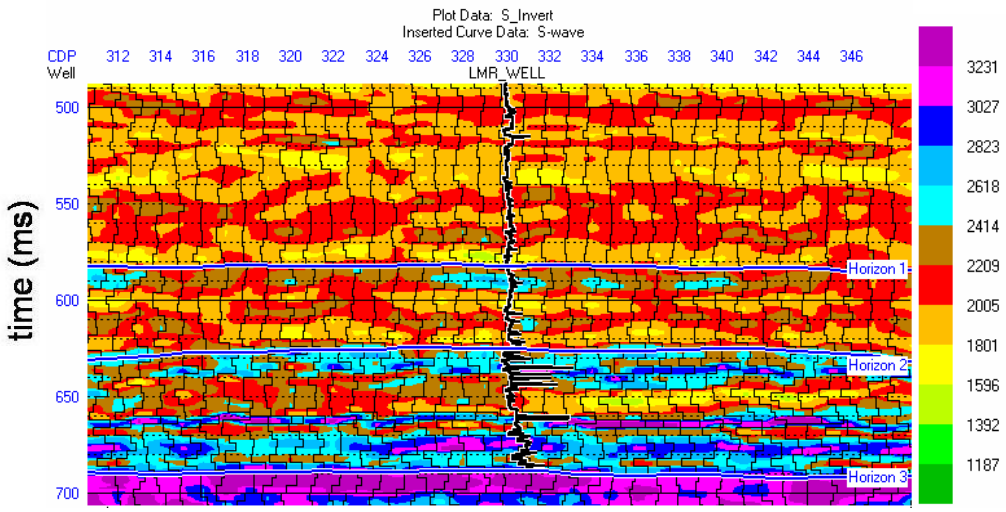


Figure 2.30: The S-wave impedance, Z_S, found by inverting the R_{S0} estimate of a gas sand in Alberta.

Horizon 2 in both figures is the top of the gas sand. In Figure 2.29 the gas sand shows a drop in P-impedance with respect to the encasing shale. However the S-impedance (Fig. 2.30) does not show the same decrease as we move into the gas sand. This can be understood when we recall that S-wave velocity is insensitive to the fluid, whereas P-wave velocity shows a sudden decrease when gas is introduced into the reservoir.

Figures 2.31 and 2.32 show the fluid and skeleton terms (ρf and ρs) computed from the P- and S-impedance sections of Figures 2.29 and 2.30, where I used a c value of 2.333. These sections behave exactly as we would expect. That is, ρf (Fig. 2.31) shows a strong decrease in the gas-filled reservoirs, whereas ρs (Fig. 2.32) shows an increase in the reservoir (since the sand matrix has a higher value than the overlying shale).

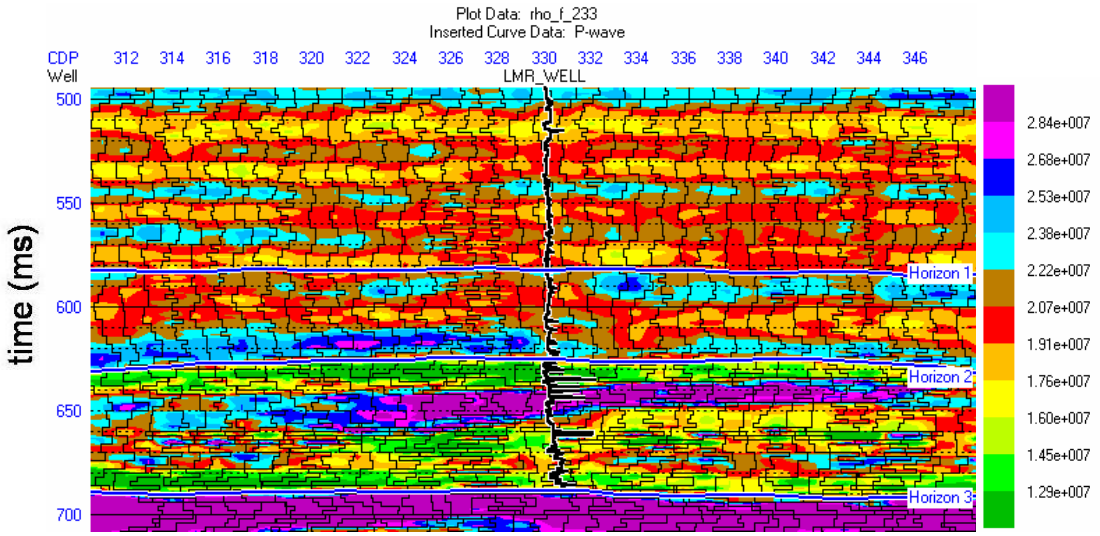


Figure 2.31: The ρf section found by combining the Z_P and Z_S inversions of Figure 2.29 and 2.30 using a c value of 2.333.

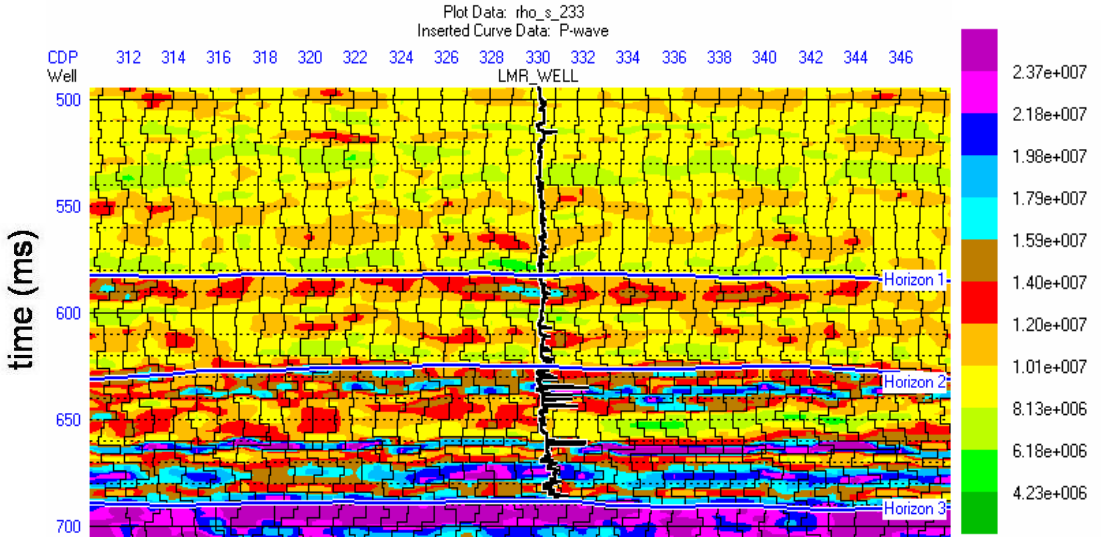


Figure 2.32: The ρs section found by combining the Z_P and Z_S inversions of Figure 2.29 and 2.30 using a c value of 2.333.

Figure 2.33 shows a crossplot of ρf vs ρs between the productive zones for the two sections of Figures 2.31 and 2.32, where the gas sand is again clearly visible in the red region. Figure 2.34 then shows the corresponding zones on the seismic section plotted from the crossplot of Figure 2.33.

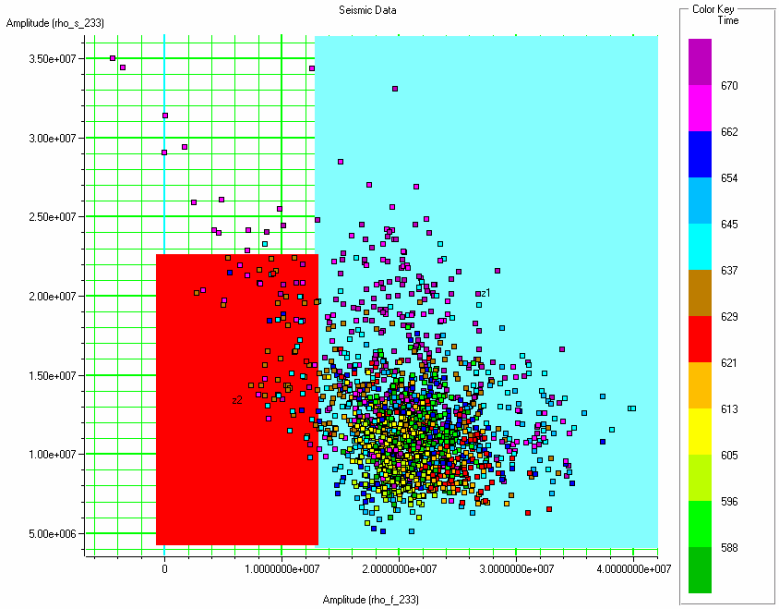


Figure 2.33: A crossplot between the sections of the previous two figures over the productive zone.

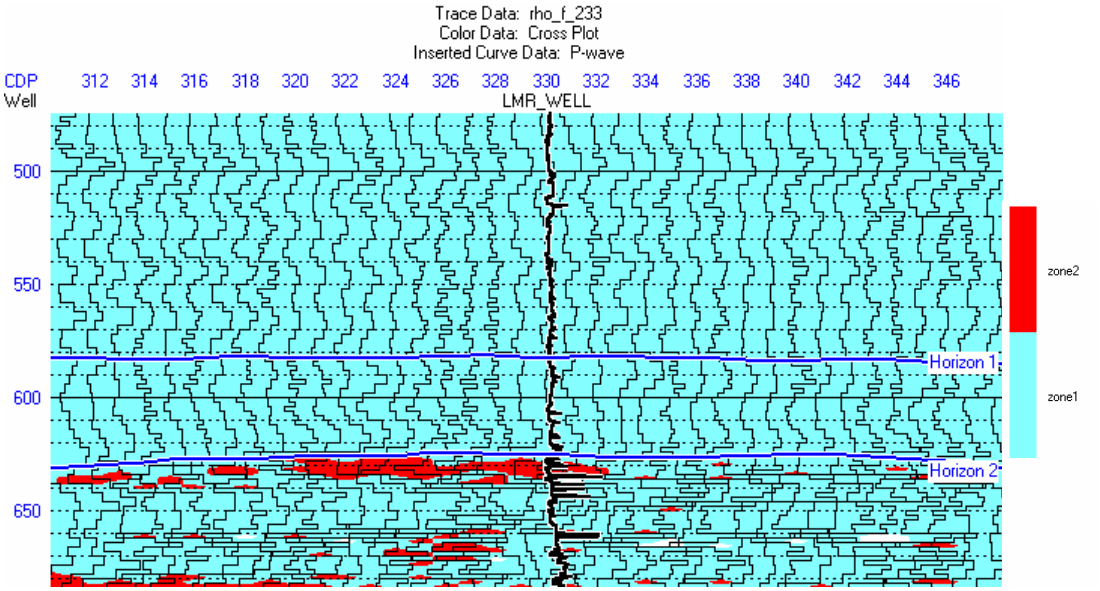


Figure 2.34: The portion of the seismic section corresponding to the gas and non-gas zones. The "red" gas region is where expected.

2.10 Conclusions

In this chapter, I discussed the various seismic attributes that will be used in subsequent chapters. These included single trace attributes, multi-traces attributes, and model-based attributes. Single trace attributes consist of instantaneous attributes, windowed-frequency attributes, recursive attributes, and bandpass attributes.

Multi-trace attributes consist of coherency and AVO. In the coherency method, we use a window of traces from a 3D seismic volume to extract information about discontinuities within the seismic volumes, such as faults and fractures. AVO attributes are extracted from pre-stack traces and generally involve two independent attributes that relate to information about the fluid content of the reservoir.

Finally, model-based attributes involved the combination of information derived from the seismic trace with a geologically based model, and included post-stack seismic inversion, inversion of AVO attributes. These are also referred to as deterministic attributes.

CHAPTER 3 : MULTILINEAR REGRESSION

3.1 Introduction

In Chapter 2, I discussed the seismic attributes that will be used in this study. In this chapter I will begin the analysis of how to optimally combine these attributes in order to predict reservoir parameters. Specifically, I will focus on the technique of multilinear regression, which is a generalization of the solution to finding the best least-squares fit to a straight line. Multilinear regression will be used in two fundamental ways in this work. First, it can be used as our primary approach for the technique of reservoir parameter prediction (Russell et al., 1997). As we will see in the case study at the end of the chapter, this is especially true when we apply this approach to well log prediction using other well logs. Second, multilinear regression gives us a fast and efficient way to group our seismic attributes prior to analysis with the more powerful nonlinear neural networks to be discussed in subsequent chapters (Hampson et al., 2001).

In this chapter I will start by discussing multivariate statistics. I will consider both univariate and bivariate statistics, before generalizing to N-dimensional attribute space. I will then review the multivariate normal distribution, looking at the univariate, bivariate and trivariate cases before generalizing to N dimensions. I will then discuss multilinear regression, and show how this technique can be used to solve for the prediction of reservoir parameters from seismic attributes. Finally, these techniques will be applied to an actual case study.

3.2 A word about notation

Before I start discussing multivariate statistics and multilinear regression, it is important to understand the terminology used in this study when referring to seismic attributes, and the distinction between the attribute vector and the sample vector (this is

given in more detail in Appendix 1). This is best seen in Figure 3.1, which was also shown in Chapter 1 as Figure 1.5. This figure shows the basic problem of predicting a target log using multiple seismic attributes.

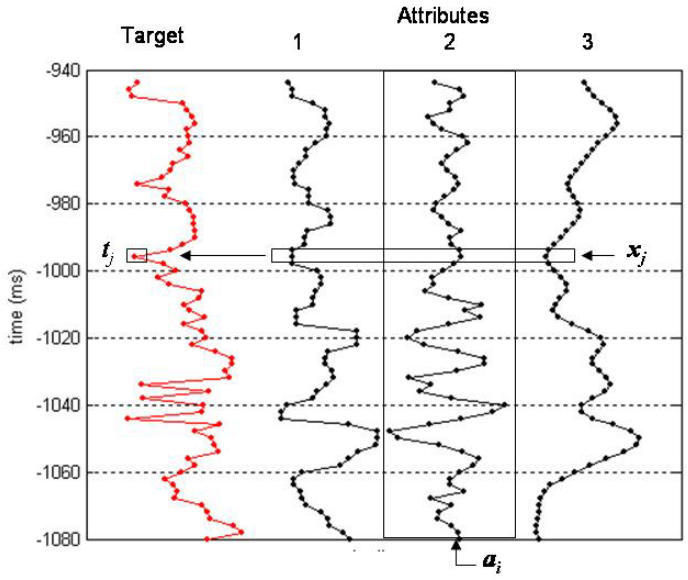


Figure 3.1: The basic multilinear regression problem, showing the sample vector $\boldsymbol{x_j}$, attribute vector $\boldsymbol{a_j}$, and the target value t_j.

In Figure 3.1, the i^{th} attribute vector $\boldsymbol{a_i}$ represents all N samples of a particular seismic attribute. In the case shown in the figure there are three attribute vectors. However, in general, there will be M attribute vectors. In Figure 3.1, notice that I have shown only a single sample of the target log, written t_j. However, there are also N samples in the target, so we can think of the target log as the vector $\boldsymbol{t}^T = [t_1 \quad \cdots \quad t_N]$. The M attribute vectors are the same length as the target vector, and can be written $\boldsymbol{a}_i^T = [a_{1i} \quad \cdots \quad a_{Ni}], i = 1,\ldots,M$.

Referring back to Figure 3.1, note that the j^{th} sample vector, $\boldsymbol{x_j}$, is the M-dimensional vector of attribute values associated with the j^{th} target sample t_j, and can be written as $\boldsymbol{x}_j^T = [x_{1j} \quad \cdots \quad x_{Mj}], j = 1,\ldots,N$.

If we consider the attribute vectors as the columns of this matrix, we get the N row by M column matrix A, or

$$A = [\boldsymbol{a}_1 \quad \cdots \quad \boldsymbol{a}_M] = \begin{bmatrix} a_{11} & \cdots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NM} \end{bmatrix}. \tag{3.1}$$

If we consider the sample vectors as the columns of the matrix, we get the M row by N column matrix X, or

$$X = [\boldsymbol{x}_1 \quad \cdots \quad \boldsymbol{x}_N] = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{MN} \end{bmatrix}. \tag{3.2}$$

Note that matrices X and A contain all of the seismic samples that will be used in the reservoir prediction problem and are the transpose of each other, which can be written as $A = X^T$. In multivariate statistics, matrices A and X represent what is referred to as a single multivariate observation (e.g. Johnson and Wichern, 1998, page 117).

3.3 Multivariate statistics

I will now review the concepts of multivariate statistics using the dataset shown in Figure 3.1. As already mentioned, this figure shows a target sonic log on the left and three seismic attributes on the right. Our objective is to combine these attributes in an optimum way to approximate the target log. The target log is the *P*-wave sonic log from well 01-08 in Figure 2.1. To relate the attributes shown in Figure 3.1 to those discussed in the last chapter, Attribute 1 is inverted seismic impedance using a model-based inversion algorithm, Attribute 2 is the derivative of the seismic trace with respect to time, and attribute 3 is the integration of the seismic trace with respect to time. All three attributes were extracted from the 3D seismic volume shown in Figure 2.1, where the attribute traces within a one trace radius of the well location have been averaged. The reason that these particular attributes were chosen will be discussed later in this chapter.

3.3.1 Univariate statistics

Let us start by looking at the statistics of each attribute individually. From a statistical point of view, each attribute is called a variate, and I will thus first discuss univariate statistics. To understand the range and distribution of values in the target log and each of the attributes, it is useful to create a histogram of these values. In a histogram, we find the minimum and maximum values of each variate, divide this range into N divisions, and determine how many values fall into each division. Figure 3.2 shows a histogram of each variate, where I have chosen $N = 10$. There are 69 points in each of our variates.

The histograms in Figure 3.2 show us the symmetry (or lack of it) of the distributions of the values in the target and each attribute, and also the basic statistics of the variates. The distribution of points shown in Figures 3.2(a), (c), and (d), are reasonably symmetrical, whereas the distribution in Figure 3.2(b) is slightly skewed. I will talk more about the shape of these distributions in the next section.

The two basic statistics that I will derive for each of our variates are the mean and variance. The mean is the arithmetic average of the sample values and the variance is the average of the sum of the squared difference between the sample values and the mean. Using the notation shown in Figure 3.1, the means of the target and attributes can be written as

$$\mu_t = \frac{1}{N}\sum_{j=1}^{N} t_j \,, \tag{3.3}$$

and

$$\mu_i = \frac{1}{N}\sum_{j=1}^{N} a_{ji} \,, \tag{3.4}$$

where I have used the subscript i to represent the i^{th} attribute mean.

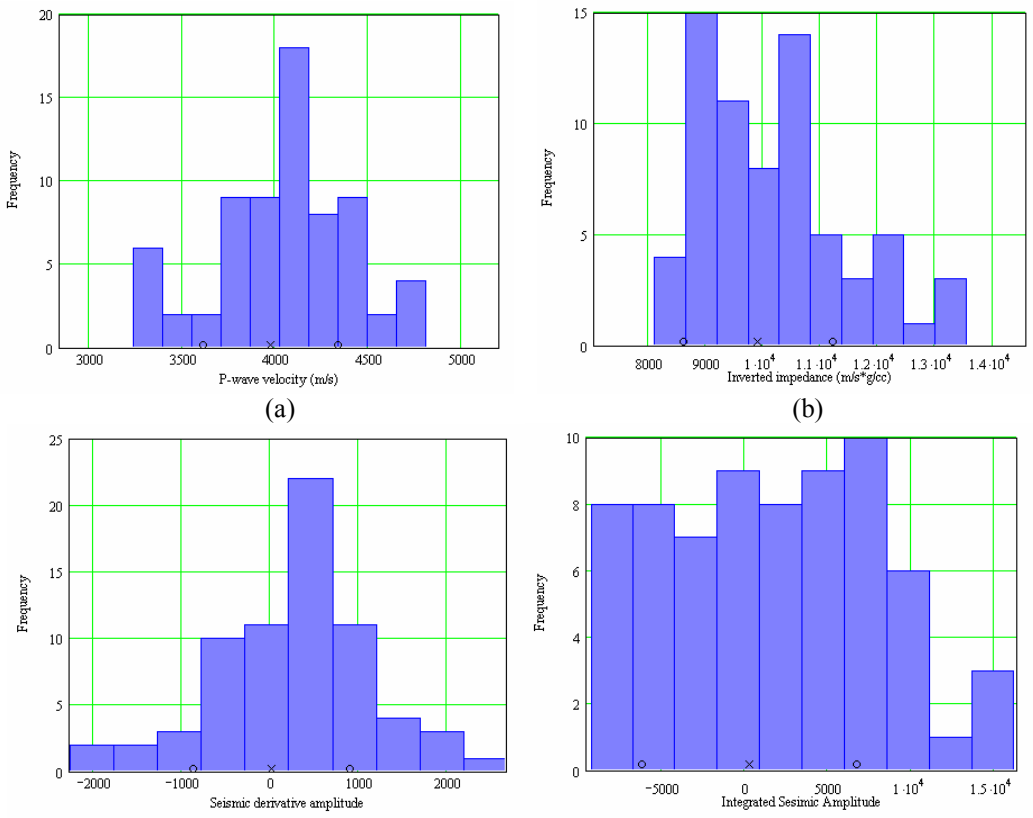


Figure 3.2: Histograms of the target and attributes shown in Figure 3.1, where (a) is the histogram of the target log, (b) is the histogram of attribute 1, (c) is the histogram of attribute 2, and (d) is the histogram of attribute 3. In each figure, *x* represents the position of the mean value, and *o* represents the position of the mean plus and minus the standard deviation, respectively.

The mean is also called the first moment. Another important statistic is the second moment, which is related to the square of the samples. To compute the second moment, we simply replace the terms after the summation sign in equations (3.3) and (3.4) with their squared values. A more common measurement is the variance, which is defined as the second moment away from the mean, and can be written as

$$\sigma_t^2 = \frac{1}{N}\sum_{j=1}^{N}(t_j - \mu_a)^2 = \frac{1}{N}\sum_{j=1}^{N} t_j^2 - \left(\frac{1}{N}\sum_{j=1}^{N} t_j\right)^2 \tag{3.5}$$

for the target, and as

$$\sigma_i^2 = \frac{1}{N}\sum_{j=1}^{N}(a_{ji} - \mu_a)^2 = \frac{1}{N}\sum_{j=1}^{N} a_{ji}^2 - \left(\frac{1}{N}\sum_{j=1}^{N} a_{ji}\right)^2 \tag{3.6}$$

for the attributes. Notice that the variance can be written either as the normalized sum of the squared differences between the values and their mean, or as the difference between the second moment and the square of the mean.

An alternate expression for the variance involves using vector notation. In this case we observe that the variance is the scalar product of the vectors, after subtracting the means, divided by the number of samples. In equations, we write

$$\sigma_t^2 = \frac{(\boldsymbol{t} - \boldsymbol{\mu}_t)^T (\boldsymbol{t} - \boldsymbol{\mu}_t)}{N}, \text{ and} \tag{3.7}$$

$$\sigma_i^2 = \frac{(\boldsymbol{a}_i - \boldsymbol{\mu}_i)^T (\boldsymbol{a}_i - \boldsymbol{\mu}_i)}{N}, \tag{3.8}$$

where $\boldsymbol{\mu}_t = \mu_t \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$ and $\boldsymbol{\mu}_i = \mu_i \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$, both vectors of length N. An alternate definition of the variance uses a divisor of $N-1$, rather than N. There are sound theoretical reasons for this, since dividing by $N-1$ leads to an unbiased maximum likelihood estimator (Hogg and Craig, 1995). However, this only becomes crucial as the value N gets very small, and our datasets are usually quite large. Also, it is more convenient to use the divisor N when we discuss the covariance matrix. Another important statistic is the standard deviation, σ, which is the square root of the variance. In Figure 3.2, both the means and the mean ± one standard deviation have been shown for the target and attributes. A summary of these values is given in Table 3.1.

	Mean	Variance	Standard Deviation
Target	3975 m/s	1.303×10^5 $(m/s)^2$	361 m/s
Attribute 1	9903 m/s·g/cc	1.709×10^6	1307 m/s·g/cc
Attribute 2	15.3	7.875×10^5	887
Attribute 3	264	4.187×10^7	6471

Table 3.1. The univariate statistics of the target and attribute vectors shown in Figure 3.1.

3.3.2 Bivariate statistics

The univariate statistics just discussed give us information about the individual variates, but not about their inter-relationships. The inter-relationships are best shown by crossplotting each pair of variates, and Figure 3.3 displays the four-by-four matrix of crossplots shown in Figure 1.2. In each crossplot, the means have been set to zero, and the values have been scaled so that the largest absolute value is equal to one.

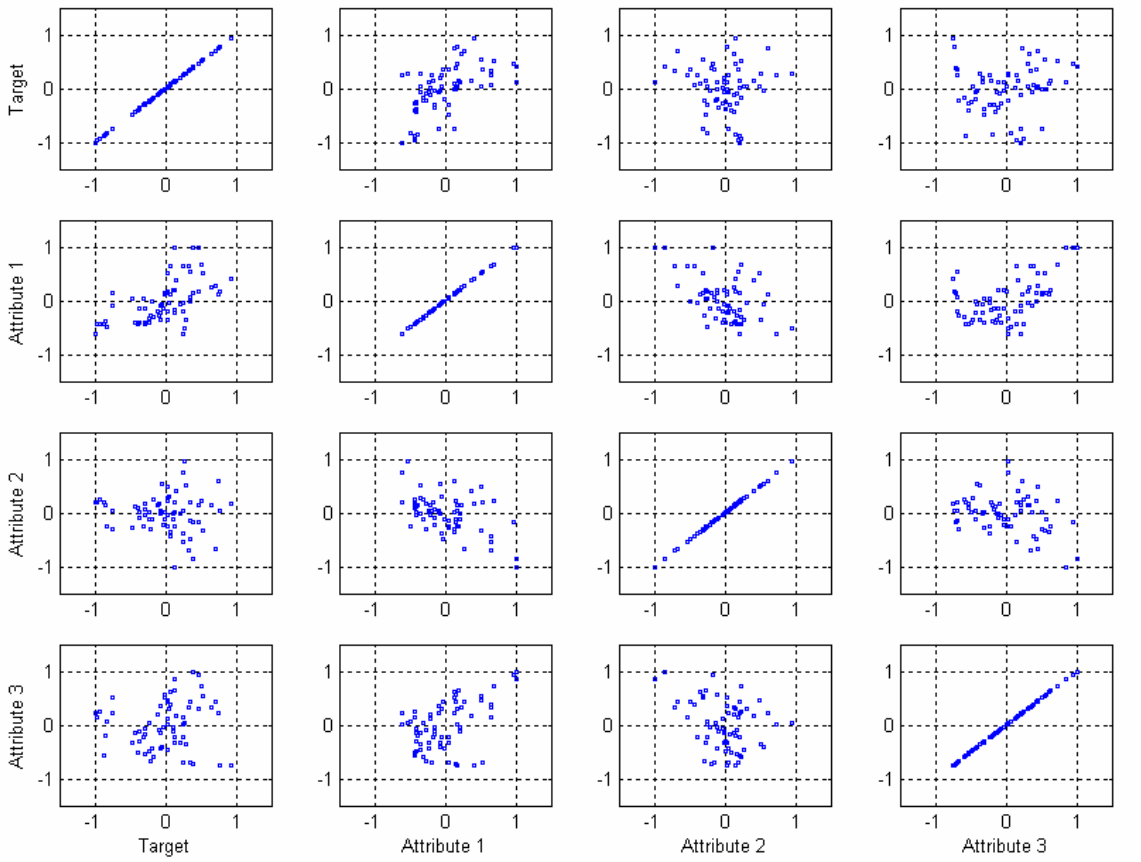


Figure 3.3: A four-by-four matrix of crossplots, where the target log and each of the attributes shown in Figure 3.1 have been crossplotted against each other.

The quantitative information contained in a crossplot can be found by computing the covariance between the two variates, which is a natural extension of the variance. Notice that there are two types of covariance indicated in Figure 3.3, the covariance

between the target and an attribute, which relates to the predictive ability of each attribute, and the covariance between a pair of attributes, which relates to the independence of the attributes. Let us first consider the covariance between two attributes, where the two attributes are $\boldsymbol{a_i}$ and $\boldsymbol{a_k}$. The expression for covariance is very similar to that of variance in Equation 3.6, and can be written in summation form as

$$\sigma_{ik} = \frac{1}{N}\sum_{j=1}^{N}(a_{ji} - \mu_j)(a_{jk} - \mu_k) = \frac{1}{N}\sum_{j=1}^{N} a_{ji}a_{jk} - \left(\frac{1}{N}\sum_{j=1}^{N} a_{ji}\right)\left(\frac{1}{N}\sum_{j=1}^{N} a_{jk}\right). \quad (3.9)$$

In equation (3.9) the covariance has been written first as the normalized product of the differences between the values and their means and, second, as the difference between the normalized sum of the products and the product of the means. The normalized sum of the products is called the *correlation* of the two attributes. This second expression will be useful in a later section when interpreting the regression coefficients. It is also clear that the variance written in equation (3.6) is a special type of covariance, in which $i = k$. This means that $\sigma_{ii} = \sigma_i^2$. The variance is often referred to as the auto-covariance and can be seen on Figure 3.3 as straight line fits.

As with the variance, we can also write the covariance in vector form as

$$\sigma_{ik} = \frac{(\boldsymbol{a_i} - \boldsymbol{\mu_i})^T(\boldsymbol{a_k} - \boldsymbol{\mu_k})}{N}, \quad (3.10)$$

where again the vector means are simply the product of the scalar means and an N-length vector of ones. In the same way, the covariance between the target and one of the attributes can be written as

$$\sigma_{it} = \frac{1}{N}\sum_{j=1}^{N}(a_{ji} - \mu_a)(t_j - \mu_t) = \frac{1}{N}\sum_{j=1}^{N} a_{ji}t_j - \left(\frac{1}{N}\sum_{j=1}^{N} a_{ji}\right)\left(\frac{1}{N}\sum_{j=1}^{N} t_j\right), \quad (3.11)$$

or as

$$\sigma_{ti} = \frac{(\boldsymbol{t} - \boldsymbol{\mu_t})^T(\boldsymbol{a_i} - \boldsymbol{\mu_i})}{N}, \quad (3.12)$$

Again, note that the auto-covariance of the target is identical to the variance of the target.

Another useful statistic is the normalized covariance, or the correlation coefficient, which can be written for two attributes as

$$\rho_{ik} = \frac{\sigma_{ik}}{\sigma_i \sigma_k}. \tag{3.13}$$

That is, the correlation coefficient is the covariance between the two vectors divided by the product of the standard deviations of each vector. Obviously, the correlation coefficient for auto-covariance is equal to one. A correlation coefficient of *0* would indicate no correlation between the two vectors but, in this study, I have found that any value below *0.5* indicates poor correlation. A similar expression for the correlation coefficient between the target and one of the attributes can be easily written down. Table 3.2 shows the correlation coefficients between the target and attributes of Figure 3.1.

	Target	Attribute 1	Attribute 2	Attribute 3
Target	1	0.57	-0.03	0.107
Attribute 1	0.57	1	-0.486	0.513
Attribute 2	-0.03	-0.486	1	-0.289
Attribute 3	0.107	0.513	-0.289	1

Table 3.2 . The correlation coefficients between the target and attribute vectors shown in Figure 3.1.

3.3.3 The covariance matrix

Table 3.2 can be written as a matrix, called either the covariance matrix or the correlation matrix, depending on which value is being computed. In the discussion of this section I will be restricting the discussion to the attribute covariance matrix, but these remarks are also valid for the situation shown in Table 3.2 if we consider the target vector to be the first attribute. The covariance matrix for *M* attributes can be written:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1M} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{M1} & \sigma_{M2} & \cdots & \sigma_{MM} \end{bmatrix}, \tag{3.14}$$

where each of the terms $\boldsymbol{\sigma}_{ik}$ in the covariance matrix are computed using equation (3.10). The correlation matrix can be written

$$\boldsymbol{\rho} = \begin{bmatrix} 1 & \rho_{12} & \cdots & \rho_{1M} \\ \rho_{21} & 1 & \cdots & \rho_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{M1} & \rho_{M2} & \cdots & 1 \end{bmatrix}, \tag{3.15}$$

where the terms ρ_{ik} are as defined in equation 3.13. By combining equations (3.1) and (3.10), we can show that the covariance matrix can be computed as

$$\boldsymbol{\Sigma} = \frac{1}{N} A_0^T A_0 . \tag{3.16}$$

where $A_0 = \begin{bmatrix} \boldsymbol{a_1} - \mu_1 & \cdots & \boldsymbol{a_M} - \mu_M \end{bmatrix} = \begin{bmatrix} a_{11} - \mu_1 & \cdots & a_{1M} - \mu_M \\ \vdots & \ddots & \vdots \\ a_{N1} - \mu_1 & \cdots & a_{NM} - \mu_M \end{bmatrix}$ is the zero-mean equivalent of matrix A. Another way to think of equation (3.16) is as a matrix of vector inner products, or

$$\boldsymbol{\Sigma} = \frac{1}{N} \begin{bmatrix} \boldsymbol{a}_{10}^T \boldsymbol{a}_{10} & \boldsymbol{a}_{10}^T \boldsymbol{a}_{20} & \cdots & \boldsymbol{a}_{10}^T \boldsymbol{a}_{M0} \\ \boldsymbol{a}_{20}^T \boldsymbol{a}_{10} & \boldsymbol{a}_{20}^T \boldsymbol{a}_{20} & \cdots & \boldsymbol{a}_{20}^T \boldsymbol{a}_{M0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{a}_{M0}^T \boldsymbol{a}_{10} & \boldsymbol{a}_{M0}^T \boldsymbol{a}_{20} & \cdots & \boldsymbol{a}_{M0}^T \boldsymbol{a}_{M0} \end{bmatrix}, \tag{3.17}$$

where $\boldsymbol{a_{i0}} = \boldsymbol{a_i} - \boldsymbol{\mu_i}$. Equation (3.17) illustrates the basic structure of the covariance matrix, where each element in the matrix, σ_{ik}, is the inner product of the zero-mean attributes $\boldsymbol{a_{i0}}$ and $\boldsymbol{a_{k0}}$.

In the above derivation of covariance I have used the attribute vectors, $\boldsymbol{a_i}$, shown in Figure 3.1. An alternate way to define the covariance matrix is based on the sample vectors, $\boldsymbol{x_i}$, also shown in Figure 3.1. To derive the covariance matrix in this way, we will compute the mean values slightly differently. That is, we compute an M-dimensional vector of means given by

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{j=1}^{N}\boldsymbol{x}_j = \frac{1}{N}\left\{\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{M1} \end{bmatrix} + \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{M2} \end{bmatrix} + \cdots + \begin{bmatrix} x_{1N} \\ x_{2N} \\ \vdots \\ x_{MN} \end{bmatrix}\right\} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_M \end{bmatrix}. \tag{3.18}$$

Note that the individual values in the mean vector $\boldsymbol{\mu}$ are the attribute means given by equation (3.4). The computation of the covariance matrix is then found by computing the sum of the normalized *outer products* of $\boldsymbol{x}_j$, where $j = 1$ to N, and dividing by N, or

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{j=1}^{N}(\boldsymbol{x}_j - \boldsymbol{\mu})(\boldsymbol{x}_j - \boldsymbol{\mu})^T \tag{3.19}$$

Equation (3.19) will again produce an M x M matrix, which is the same calculation as

$$\boldsymbol{\Sigma} = \frac{1}{N} X_0 X_0^T, \tag{3.20}$$

where $X_0 = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu} & \cdots & \boldsymbol{x}_N - \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} x_{11} - \mu_1 & \cdots & x_{1N} - \mu_1 \\ \vdots & \ddots & \vdots \\ x_{M1} - \mu_M & \cdots & x_{MN} - \mu_M \end{bmatrix}$.

3.4 The multivariate normal distribution

3.4.1 The general case

For an M-dimensional vector $\boldsymbol{x}$, the multivariate normal distribution is written:

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{M/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right], \tag{3.21}$$

where $\boldsymbol{\mu}$ is the M-dimensional vector of means defined in equation 3.18 and $\boldsymbol{\Sigma}$ is the covariance matrix given by equation (3.14). The term inside the exponential of equation (3.21), written

$$\Delta^2 = (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}), \tag{3.22}$$

is called the statistical, or Mahalanobis distance (Bishop, 1995), and measures the statistical distance from $\boldsymbol{x}$ to $\boldsymbol{\mu}$ and defines hyperellipsoids of constant probability density.

3.4.2 The univariate case

The simplest case of equation (3.21) is the univariate normal distribution ($M = 1$), which is written:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2} \right], \tag{3.23}$$

where the mean μ and variance σ^2 are as defined in equations (3.3) through (3.7). In Figure 3.4, the histograms shown in Figure 3.2 have been fitted with normal distributions based on the means and standard deviations shown in Table 3.1.

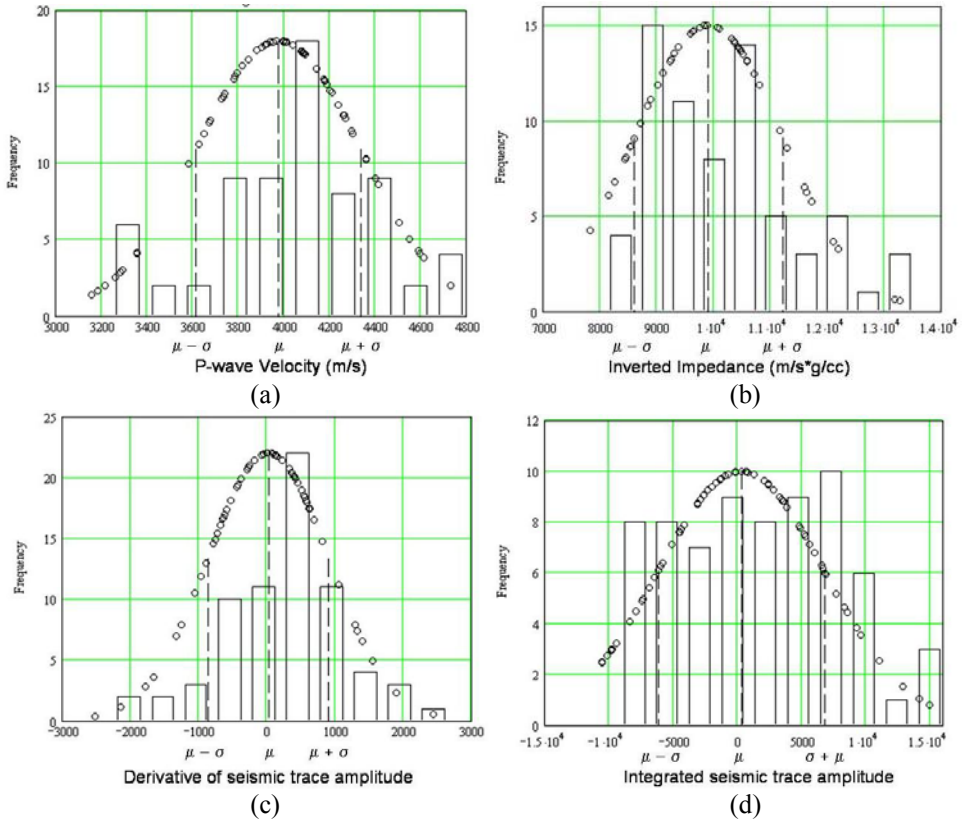


Figure 3.4: The histograms shown in Figure 3.2 have been fitted with the normal distributions based on their means and standard deviations, where (a) is the distribution of the target log, (b) is the distribution of attribute 1, (c) is the distribution of attribute 2, and (d) is the distribution of attribute 3, all from Figure 3.1

From Figure 3.2, it is clear that Figures 3.4(a) and (c), the target log and attribute 2 (derivative of the seismic trace), fit the normal distribution quite well. However, Figures 3.4(b) and (d), attribute 1 (inverted impedance) and attribute 3 (integrated seismic

trace), do not fit the normal distribution very well. Figure 3.4(b) is positively skewed and would appear to fit better to a log normal distribution, given by

$$p(x) = \begin{cases} \dfrac{1}{\sigma x\sqrt{2\pi}} \exp\left[-\dfrac{1}{2}\dfrac{(\log(x)-\mu)^2}{\sigma^2}\right], x > 0 \\ 0, \quad x \le 0 \end{cases}. \tag{3.24}$$

But Figure 3.4(b) would appear to fit better to a uniform distribution, given by

$$p(x) = \begin{cases} 0, x < a \\ \dfrac{1}{b-a}, a \le x \le b \\ 0, x > b \end{cases} \tag{3.25}$$

Despite the fact that the normal distribution does not fit our data in all cases, it usually gives a reasonable fit and will be used as our statistical model.

3.4.3 The bivariate case

For $M = 2$, equation (3.21) becomes the bivariate normal distribution, which is the simplest multivariate example and can be used to help visualize the multivariate normal distribution. In this case, the covariance matrix can be written as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{bmatrix}. \tag{3.26}$$

In equation (3.26), note that the cross-covariance terms σ_{12} are always equal to each other, whereas the auto-covariance terms σ_{11} and σ_{22} may be distinct. Depending on the nature of the auto and cross-covariance terms, there are three cases of interest for the bivariate covariance matrix. In the most general case the cross-covariances are non-zero (which implies that the attributes are statistically dependent), and the auto-covariances are distinct. In this case, the inverse covariance matrix can be written

$$\boldsymbol{\Sigma}^{-1} = \frac{1}{\sigma_{11}\sigma_{22} - \sigma_{12}^2} \begin{bmatrix} \sigma_{22} & -\sigma_{12} \\ -\sigma_{12} & \sigma_{11} \end{bmatrix}, \tag{3.27}$$

and the bivariate normal distribution can be written:

$$p(\boldsymbol{x}) = \frac{1}{2\pi\sqrt{\sigma_{11}\sigma_{22} - \sigma_{12}^2}} \exp\left[-\frac{(x_1 - \mu_1)^2 \sigma_{22} - 2(x_1 - \mu_1)(x_2 - \mu_2)\sigma_{12} + (x_1 - \mu_2)^2 \sigma_{11}}{2(\sigma_{11}\sigma_{22} - \sigma_{12}^2)} \right] \quad (3.28)$$

Using the normalized variates $z_1 = \frac{x_1 - \mu_1}{\sigma_1}$ and $z_2 = \frac{x_2 - \mu_2}{\sigma_2}$, the covariance matrix assumes the simple form:

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}, \quad (3.29)$$

where $\rho = \rho_{12}$. Equation 3.29 has an inverse given by

$$\boldsymbol{\Sigma}^{-1} = \frac{1}{1-\rho^2} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}, \quad (3.30)$$

and we can therefore re-write equation 3.28 as the standardized bivariate normal density

$$p(\boldsymbol{x}) = \frac{1}{2\pi\sqrt{1-\rho^2}} exp\left[-\frac{1}{2}\frac{1}{1-\rho^2}\left(z_1^2 - 2\rho z_1 z_2 + z_2^2\right) \right] \quad (3.31)$$

Equation (3.31) makes it clear that a contour of constant density for the bivariate distribution is defined by the equation (Anderson, 1984)

$$\frac{1}{1-\rho^2}\left(z_1^2 - 2\rho z_1 z_2 + z_2^2\right) = c^2, \quad (3.32)$$

which is an ellipse aligned at either $+45^o$, if $\rho > 0$, or -45^o, if $\rho < 0$, where c is an arbitrary constant. For $\rho > 0$, the lengths of the semi-major and semi-minor axes are $c\sqrt{1+\rho}$ and $c\sqrt{1-\rho}$, respectively, whereas for $\rho < 0$, the lengths of the semi-major and semi-minor axes are $c\sqrt{1+\rho}$ and $c\sqrt{1-\rho}$, respectively. Returning to the target and attributes shown in Figure 3.1, Table 3.1 shows that their correlation coefficients are $\rho_{ta1} = 0.57$, $\rho_{a1a2} = -0.486$, $\rho_{a1a3} = 0.107$, and $\rho_{a2a3} = -0.289$. Figure 3.5 shows the crossplots of these four cases, with elliptical contours corresponding to $c = 1$ and $c = 2$, where (a) shows the target versus attribute1, (b) shows attribute 1 against attribute 2, (c) shows attribute 1 against attribute 3, and (d) shows attribute 2 against attribute 3.

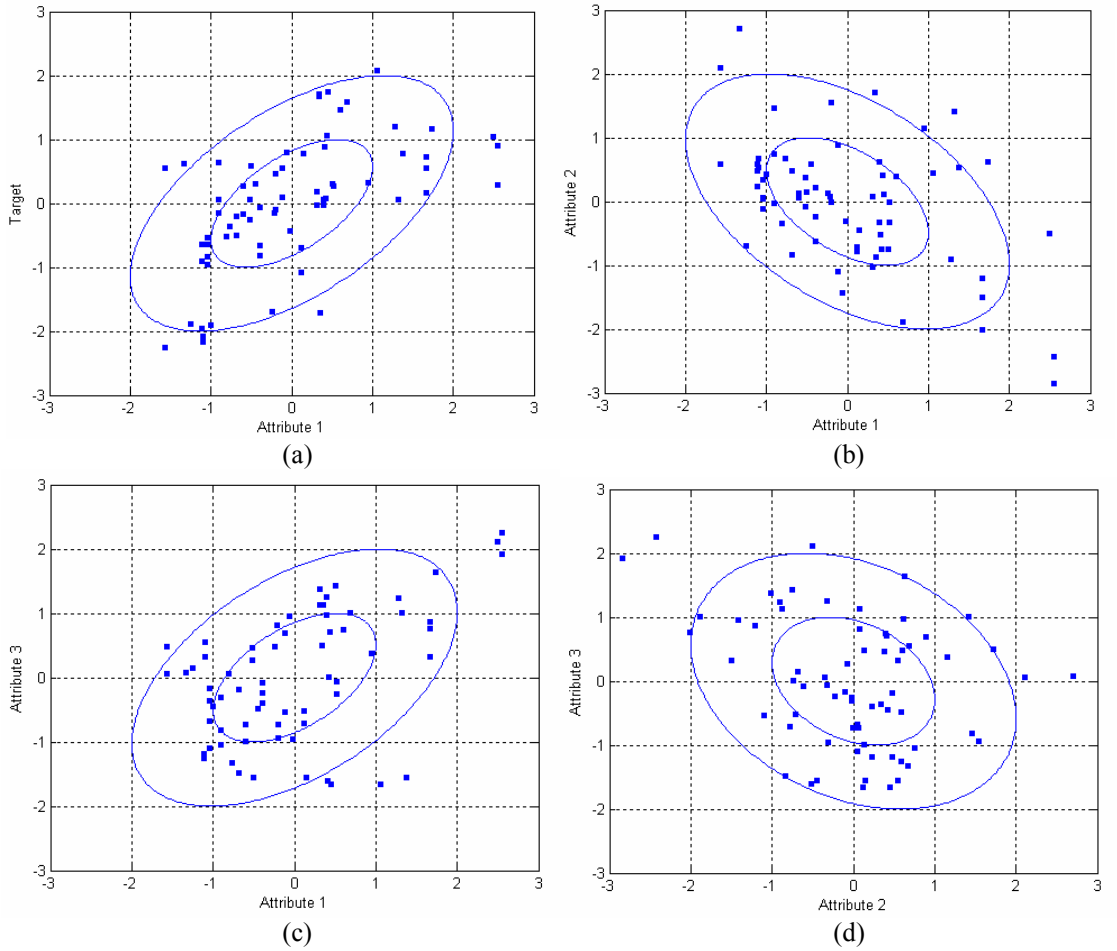


Figure 3.5: Crossplots of (a) the target versus attribute 1, (b) attribute 1 versus attribute 2, (c) attribute 1 versus attribute 3, and (d) attribute 2 versus attribute 3, with elliptical contours corresponding to $c = 1$ and 2 in equation (3.32).

Let us now consider the simpler case in which the attributes are statistically independent, which means that the cross-covariances are equal to zero, and the auto-covariances are given by the variances σ_1^2 and σ_2^2. In this case, the inverse covariance matrix becomes

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}^{-1} = \frac{1}{\sigma_1^2 \sigma_2^2} \begin{bmatrix} \sigma_2^2 & 0 \\ 0 & \sigma_1^2 \end{bmatrix} = \begin{bmatrix} \sigma_1^{-2} & 0 \\ 0 & \sigma_2^{-2} \end{bmatrix} \tag{3.33}$$

and the bivariate normal distribution can be written

$$p(\boldsymbol{x}) = \frac{1}{2\pi\sigma_1\sigma_2}\exp\left[-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right] = \frac{1}{2\pi\sigma_1\sigma_2}\exp\left[-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}\right]\exp\left[-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right] \tag{3.34}$$

The resulting distribution is the product of two gaussian curves.

In the simplest bivariate case, the attributes are statistically independent and the auto-covariances are identical and equal to the variance σ^2. In this case, the ellipses become circles, the inverse covariance matrix becomes

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix}\sigma^{-2} & 0\\ 0 & \sigma^{-2}\end{bmatrix} = \sigma^{-2}\boldsymbol{I}\,, \tag{3.35}$$

and the bivariate normal distribution can be written:

$$p(\boldsymbol{x}) = \frac{1}{2\pi\sigma^2}\exp\left[-\frac{(x_1-\mu_1)^2+(x_2-\mu_2)^2}{2\sigma^2}\right]. \tag{3.36}$$

3.4.4 The trivariate case

I will next extend our analysis to three dimensions, which is rarely considered in textbooks because of its complexity and difficulty in being visualized. Let us first compute the inverse of the covariance matrix, which can be written

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix}\sigma_{11} & \sigma_{12} & \sigma_{13}\\ \sigma_{12} & \sigma_{22} & \sigma_{23}\\ \sigma_{13} & \sigma_{23} & \sigma_{33}\end{bmatrix}^{-1} = \frac{1}{\det}\begin{bmatrix}\sigma_{22}\sigma_{33}-\sigma_{23}^2 & \sigma_{13}\sigma_{23}-\sigma_{12}\sigma_{33} & \sigma_{12}\sigma_{23}-\sigma_{13}\sigma_{22}\\ \sigma_{13}\sigma_{23}-\sigma_{12}\sigma_{33} & \sigma_{11}\sigma_{33}-\sigma_{13}^2 & \sigma_{12}\sigma_{13}-\sigma_{11}\sigma_{23}\\ \sigma_{12}\sigma_{23}-\sigma_{13}\sigma_{22} & \sigma_{12}\sigma_{13}-\sigma_{11}\sigma_{23} & \sigma_{11}\sigma_{22}-\sigma_{12}^2\end{bmatrix}, \tag{3.37}$$

where $\det = \sigma_{11}(\sigma_{22}\sigma_{33}-\sigma_{23}^2)-\sigma_{12}(\sigma_{12}\sigma_{33}-\sigma_{13}\sigma_{23})+\sigma_{13}(\sigma_{12}\sigma_{23}-\sigma_{13}\sigma_{22})$ is the determinant. To make this clearer, the inverse of the correlation matrix can be written

$$\boldsymbol{\rho}^{-1} = \begin{bmatrix}1 & \rho_{12} & \rho_{13}\\ \rho_{12} & 1 & \rho_{23}\\ \rho_{13} & \rho_{23} & 1\end{bmatrix}^{-1} = \frac{1}{\det}\begin{bmatrix}1-\rho_{23}^2 & \rho_{13}\rho_{23}-\rho_{12} & \rho_{12}\rho_{23}-\rho_{13}\\ \rho_{13}\rho_{23}-\rho_{12} & 1-\rho_{13}^2 & \rho_{12}\rho_{13}-\rho_{23}\\ \rho_{12}\rho_{23}-\rho_{13} & \rho_{12}\rho_{13}-\rho_{23} & 1-\rho_{12}^2\end{bmatrix}, \tag{3.38}$$

where $\det = (1-\rho_{23}^2)-\rho_{12}(\rho_{12}-\rho_{13}\rho_{23})+\rho_{13}(\rho_{12}\rho_{23}-\rho_{13}) = 1-\rho_{12}^2-\rho_{13}^2-\rho_{23}^2$.

Using correlation coefficients, the symmetry in equation (3.38) becomes more apparent. Since both the normalized covariance matrix and its inverse are each symmetric, we can write their general form as

$$M = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}. \tag{3.39}$$

Using equation (3.39) and the normalized variates $z_1 = \frac{x_1 - \mu_1}{\sigma_1}$, $z_2 = \frac{x_2 - \mu_2}{\sigma_2}$ and $z_3 = \frac{x_3 - \mu_3}{\sigma_3}$, we can write the simplified form of the trivariate normal density as

$$g(\boldsymbol{x}) = \frac{1}{2\pi\sqrt{\det}} \exp\left[-\frac{1}{2\det}\left(a z_1^2 + d z_2^2 + f z_3^2 + b z_1 z_2 + c z_1 z_3 + e z_2 z_3 \right) \right] \tag{3.40}$$

where the terms *a* through *f* can be evaluated by comparing equations (3.38) and (3.39). The terms *a, d,* and *f* correspond to the auto-covariances in the main diagonal, and the terms *b, c* and *e* correspond to the cross-covariance terms.

Thus, the general *M*-dimensional multivariate distribution will have *2M* terms, the first *M* terms corresponding to the auto-covariances, and the second *M* terms corresponding to the cross-covariances. As will be discussed in the next section, we can always rotate the covariance matrix back to its principal directions, in which case the cross-covariance terms disappear.

Figure 3.6 shows the three-dimensional crossplot of attributes 1, 2, and 3, with the surfaces corresponding to $g(\boldsymbol{x})$ equal to exp(-1/2) and exp(-1). The three lines shown in Figure 3.6 correspond to the three principal components of the trivariate normal distribution, and will be discussed in the next section.

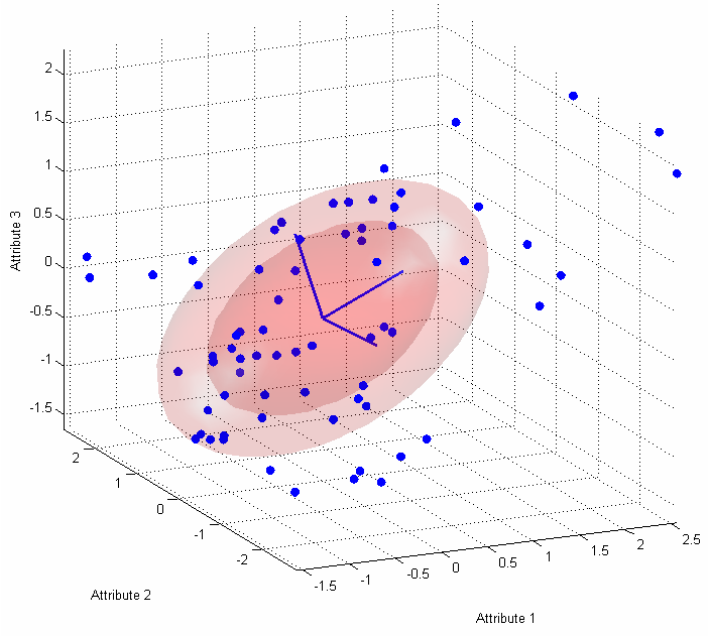


Figure 3.6: The trivariate crossplot and normal distribution contours corresponding to attributes 1, 2 and 3, showing levels corresponding to exp(-0.5) and exp(-1.0). The lines emanating from the origin show the principal axes, discussed in the next section.

3.4.5 Eigendecomposition of the multivariate normal distribution

From our preceding discussion, note that the multivariate normal distribution can be written:

$$p(\boldsymbol{x}) = s \cdot \exp\left[-\frac{\Delta^2}{2}\right], \tag{3.41}$$

where $\Delta^2 = (\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ is the Mahalanobis distance and $s = (2\pi)^{-M/2}\left|\boldsymbol{\Sigma}\right|^{-1/2}$ is a scale factor. The surfaces of constant probability density of the multivariate normal distribution are therefore hyperellipsoids defined by constant Mahalanobis distance. Although we considered the bivariate and trivariate cases in the last two sections, we can arrive at a fuller understanding of these hyperellipsoids by finding the principal axes of Δ^2, using the eigenvalue equation

$$\boldsymbol{\Sigma}\,\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i, \tag{3.42}$$

where λ_i and $\boldsymbol{u_i}$ are the i^{th} eigenvalue and normalized eigenvector corresponding to the covariance matrix $\boldsymbol{\Sigma}$. If we consider the inverse covariance matrix, we can write

$$\boldsymbol{\Sigma}^{-1}\boldsymbol{u}_i = \lambda_i^{-1}\boldsymbol{u}_i. \tag{3.43}$$

However, since by definition we know that $\boldsymbol{u_i}^T\boldsymbol{u_i}=1$, we can rewrite equation (3.43) as

$$\boldsymbol{u}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{u}_i = \lambda_i^{-1} \ . \tag{3.44}$$

Equating the right-hand side of equation (3.44) to the Mahalanobis distance defined in equation (3.41), it is clear that $\Delta^2 = \lambda_i^{-1}$ and that that the principal axes of the ellipsoids are equal in length to the inverse square root of the eigenvalues. To illustrate this for the bivariate case we can use the normalized covariance matrix of equation (3.29) to find that the two eigenvalues are

$$\begin{vmatrix} 1-\lambda & \rho \\ \rho & 1-\lambda \end{vmatrix} = 0 \Rightarrow \lambda^2 - 2\lambda + (1-\rho^2) = 0 \Rightarrow \lambda_1 = 1-\rho, \lambda_2 = 1+\rho.$$

The eigenvectors associated with λ_1 and λ_2 are:

$$\lambda_1 = 1-\rho \Rightarrow \begin{bmatrix} \rho & \rho \\ \rho & \rho \end{bmatrix}\begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \boldsymbol{u_1} = \begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

and

$$\lambda_2 = 1+\rho \Rightarrow \begin{bmatrix} -\rho & \rho \\ \rho & -\rho \end{bmatrix}\begin{bmatrix} u_{12} \\ u_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \boldsymbol{u_2} = \begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

We can then write the eigenvector matrix as

$$U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix},$$

and show that the original correlation matrix can be rotated to its principal axes by the following equation:

$$\Lambda = U^T\Sigma U = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1-\rho & 0 \\ 0 & 1+\rho \end{bmatrix}. \tag{3.45}$$

Notice in equation (3.45) that the rotated matrix Λ corresponds to the diagonal eigenvalue matrix, and that the principal components of the bivariate ellipse are given by the square roots of the eigenvalues, as discussed earlier. Since we are using the normalized covariance matrix, or correlation matrix, the operation in equation (3.45) corresponds to a 45^o rotation. Another way to look at this is to realize that U corresponds to the rotation matrix given by

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \Rightarrow R(45^o) = U = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}.$$

The preceding discussion can be generalized to M dimensions and also to the un-normalized covariance matrix, in which case the rotation is through an arbitrary angle.

An alternate and equivalent interpretation to the one given above is

$$\Lambda = (R^T X^T)(XR) = R^T (X^T X) R = R^T \Sigma R, \tag{3.46}$$

where the rotations are applied to both the original matrix and its transpose.

3.5 Multivariate regression

3.5.1 Introduction to multivariate regression

As shown in Figure 3.1, I assume that we have a target log $\boldsymbol{t}$, with N samples, and that we have M attributes $\boldsymbol{a_i}$, also each with N samples. In general, we have a lot more samples than attributes, so that $M << N$ (for example, in Figure 3.1, we have three attributes and 69 time samples). We can then write the fundamental formula for linear prediction as

$$\boldsymbol{t} = w_0 \boldsymbol{a}_0 + w_1 \boldsymbol{a}_1 + \cdots + w_M \boldsymbol{a}_M, \tag{3.47}$$

where $\boldsymbol{t^T} = \begin{bmatrix} t_1 & t_2 & \cdots & t_N \end{bmatrix}$, and $\boldsymbol{a_i^T} = \begin{bmatrix} a_{1i} & a_{2i} & \cdots & a_{Ni} \end{bmatrix}$. Note that the zeroth attribute vector is written $\boldsymbol{a_0^T} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$.

Equation (3.47) can be written more compactly as

$$\boldsymbol{t} = A\,\boldsymbol{w}, \tag{3.48}$$

where $A = \begin{bmatrix} 1 & a_{11} & \cdots & a_{1M} \\ 1 & a_{21} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{N1} & \cdots & a_{NM} \end{bmatrix}$, and $\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}$.

As shown in Figure 3.1, there is a second way to visualize the problem. This is to consider each value in the target log, t_j, as the scalar product of the weight vector and a vector $\boldsymbol{x_j}$, which is an (M+1)-dimensional vector given by the attribute values at a given sample. This can be written:

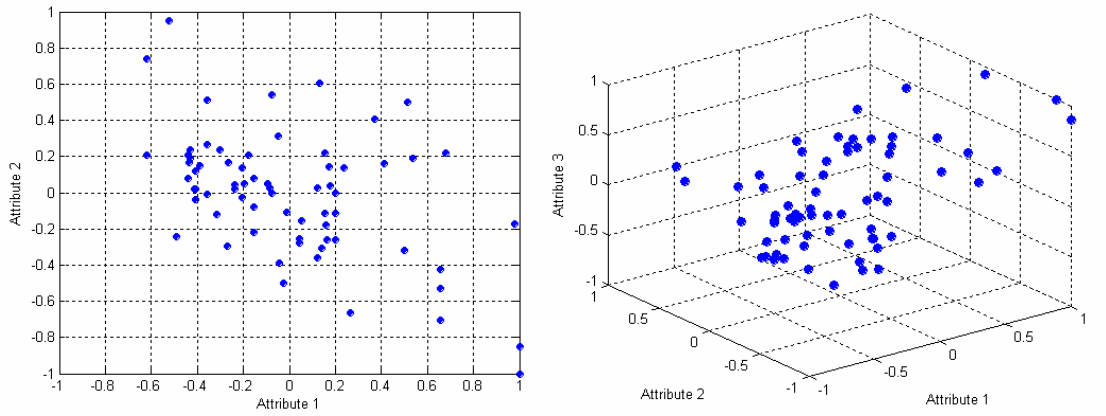
$$t_j = \boldsymbol{w}^T \boldsymbol{x}_j, \quad j = 1, \ldots, N, \tag{3.49}$$

where $\boldsymbol{w}^T = \begin{bmatrix} w_0 & w_1 & \cdots & w_M \end{bmatrix}$, and $\boldsymbol{x}_j^T = \begin{bmatrix} 1 & x_{1j} & \cdots & x_{Mj} \end{bmatrix}$. As discussed in the previous section, the matrix given by the collection of all N vectors $\boldsymbol{x_j}$ is the transpose of the matrix given by the collection of all M vectors $\boldsymbol{a_i}$. That is worth restating at this point. That is, we can write the two matrices as

$$A = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_M \end{bmatrix} = \begin{bmatrix} 1 & a_{11} & \ldots & a_{1M} \\ 1 & a_{21} & \ldots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{N1} & \ldots & a_{NM} \end{bmatrix} = X^T = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \ldots & x_{M1} \\ 1 & x_{12} & \ldots & x_{M2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1N} & \ldots & x_{MN} \end{bmatrix} \tag{3.50}$$

The importance of equation (3.50) for an understanding of the problem to be solved in this dissertation cannot be overstated. The vectors $\boldsymbol{a_i}$ are fundamental to the regression problem, whereas the vectors $\boldsymbol{x_j}$ are fundamental to both classification and basis function neural networks. Figure 3.7 shows the $\boldsymbol{x_j}$ vectors in two- and three-dimensional space. Notice that the two-dimensional crossplot shown in Figure 3.7(a) can be thought of as the top-down projection of the three-dimensional crossplot shown in

Figure 3.7(b). We could extend this analysis to any number of dimensions but cannot visualize more than three dimensions.



(a) (b)

Figure 3.7: Crossplots of the attributes shown in Figure 3.1, where (a) shows the first two attributes, and (b) shows all three attributes. Each point in the crossplots can be thought of as a vector $\boldsymbol{x_j}$ without the zeroth term.

Since equation (3.47) constitutes an overdetermined problem (that is, more observations than unknowns), its least-squares solution can be given as

$$\boldsymbol{w} = (A^T A + \lambda I)^{-1} A^T \boldsymbol{t}\,, \tag{3.51}$$

where λ is a pre-whitening factor and I is the M x M identity matrix. A complete discussion of equation (3.51) is given in Appendix 2.

3.5.2 Solving for the weights in the two-dimensional case

Before discussing the general case, let us consider the two-dimensional case of equation (3.47), given by

$$\boldsymbol{t} = w_0 \boldsymbol{a_0} + w_1 \boldsymbol{a_1}\,. \tag{3.52}$$

Equation (3.52) can be written in matrix form as

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} 1 & a_1 \\ 1 & a_2 \\ \vdots & \vdots \\ 1 & a_N \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}\,. \tag{3.53}$$

In the linear problem of equations (3.52) and (3.53), w_0 is called the intercept and w_1 the gradient. The solution to equation (3.53) can be computed using the least-squares solution of equation (3.51), giving (with $\lambda = 0$)

$$\begin{aligned}\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} &= \left\{ \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_N \end{bmatrix} \begin{bmatrix} 1 & a_1 \\ 1 & a_2 \\ \vdots & \vdots \\ 1 & a_N \end{bmatrix} \right\}^{-1} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_N \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ y_N \end{bmatrix} \\ &= \begin{bmatrix} N & \sum_{j=1}^{N} a_j \\ \sum_{j=1}^{N} a_j & \sum_{j=1}^{N} a_j^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{j=1}^{N} t_j \\ \sum_{j=1}^{N} a_j t_j \end{bmatrix} \\ &= \frac{1}{N\sum_{j=1}^{N} a_j^2 - \left(\sum_{j=1}^{N} a_j\right)^2} \begin{bmatrix} \sum_{j=1}^{N} a_j^2 \sum_{j=1}^{N} t_j - \sum_{j=1}^{N} a_j \sum_{j=1}^{N} a_j t_j \\ N\sum_{j=1}^{N} a_j t_j - \sum_{j=1}^{N} a_j \sum_{j=1}^{N} t_j \end{bmatrix}\end{aligned} \tag{3.54}$$

We can therefore write out the full computation for the intercept as

$$w_0 = \frac{\sum_{j=1}^{N} a_j^2 \sum_{j=1}^{N} t_j - \sum_{j=1}^{N} a_j \sum_{j=1}^{N} a_j t_j}{N\sum_{j=1}^{N} a_j^2 - \left(\sum_{j=1}^{N} a_j\right)^2}, \tag{3.55}$$

and for the gradient as

$$w_1 = \frac{N\sum_{j=1}^{N} a_j t_j - \sum_{j=1}^{N} a_j \sum_{j=1}^{N} t_j}{N\sum_{j=1}^{N} a_j^2 - \left(\sum_{j=1}^{N} a_j\right)^2}. \tag{3.56}$$

If we divide both the numerator and denominator in both equations by N^2, the denominator is the variance given in equation (3.6). Also, the numerator in equation (3.56) becomes the covariance from equation (3.11).

For the gradient, we find that

$$w_1 = \frac{\sigma_{at}}{\sigma_a^2}. \tag{3.57}$$

which is the covariance of ***a*** and ***t*** divided by the variance of the independent variable ***a***.

We can also expand equation (3.55) and show that the intercept can be written as

$$w_0 = \mu_t - w_1\mu_a, \tag{3.58}$$

which is the mean of the dependent variable minus the product of the gradient and the mean of the independent variable. From this statistical interpretation, two important points can be made. First, the regression of ***t*** on ***a*** is generally different from the regression of ***a*** on ***t***. To see this, note that the regression of ***a*** on ***t*** can be written as:

$$\boldsymbol{a} = w_0^* + w_1^*\boldsymbol{t}, \tag{3.59}$$

where $w_1^* = \frac{\sigma_{at}}{\sigma_t^2}$, and $w_0^* = \mu_a - w_1\mu_t$. It is then obvious that $w_1^* = w_1$ only if $\sigma_a^2 = \sigma_t^2$, and that $w_0^* = w_0$ only if $\sigma_a^2 = \sigma_t^2$ and $\mu_a = \mu_t$. In other words, the gradients will be identical only if the variances of the target log and seismic attribute are identical, and the intercepts will be identical only if both the means and the variances of the target log and seismic attribute are identical.

Second, recall that the normalized regression coefficient can be written as the covariance of the target log and seismic attribute divided by the product of the standard deviations (the square root of the variance) of the target log and seismic attribute, or

$$\rho_{at} = \frac{\sigma_{at}}{\sigma_a\sigma_t}. \tag{3.60}$$

Thus, if $\sigma_a = \sigma_t$ the two gradients are also equal to the regression coefficient.

3.5.3 The general multivariate case

The preceding analysis can be extended to multivariate linear regression (Johnson and Wichern, 1998). Let us rewrite equation (3.47) as

$$\boldsymbol{t} = w_0 + A\boldsymbol{w}, \tag{3.61}$$

where $\boldsymbol{w}^{\boldsymbol{T}} = [w_1 \quad w_2 \quad \cdots \quad w_M]$ and $X = [\boldsymbol{a}_1 \quad \boldsymbol{a}_2 \quad \cdots \quad \boldsymbol{a}_M]$. Then, the solution to the weights can be given as

$$\boldsymbol{w} = \Sigma^{-1}\boldsymbol{\sigma}_{At}, \tag{3.62}$$

and

$$w_0 = \mu_t - \boldsymbol{w}^T \boldsymbol{\mu}_A, \tag{3.63}$$

where

$$\boldsymbol{\sigma}_{At} = \begin{bmatrix} \sigma_{a_1 t} \\ \sigma_{a_2 t} \\ \vdots \\ \sigma_{a_M t} \end{bmatrix} \text{ and } \boldsymbol{\mu}_A = \begin{bmatrix} \mu_{a_1} \\ \mu_{a_2} \\ \vdots \\ \mu_{a_M} \end{bmatrix}.$$

Let us first consider the simplest multivariate case where $M = 2$. Then, we have the regression equation

$$\boldsymbol{t} = w_0 + w_1 \boldsymbol{a}_1 + w_2 \boldsymbol{a}_2. \tag{3.64}$$

From equation (3.62), the second two terms are then given by:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \sigma_{a_1 a_1} & \sigma_{a_1 a_2} \\ \sigma_{a_1 a_2} & \sigma_{a_2 a_2} \end{bmatrix}^{-1} \begin{bmatrix} \sigma_{a_1 t} \\ \sigma_{a_2 t} \end{bmatrix} = \frac{1}{\sigma_{a_1 a_1}\sigma_{a_2 a_2} - \sigma_{a_1 a_2}^2} \begin{bmatrix} \sigma_{a_2 a_2}\sigma_{a_1 t} - \sigma_{a_1 a_2}\sigma_{a_2 t} \\ \sigma_{a_1 a_1}\sigma_{a_2 t} - \sigma_{a_1 a_2}\sigma_{a_1 t} \end{bmatrix},$$

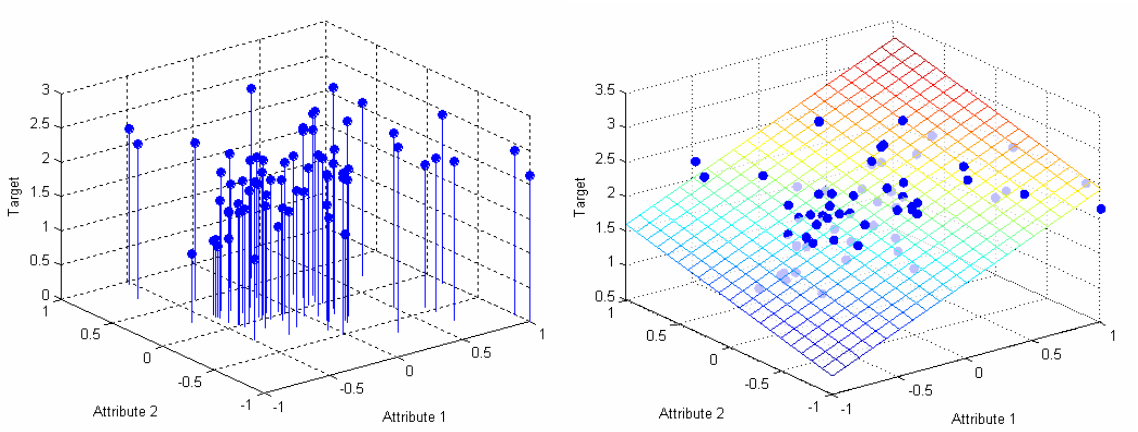
and the first term by

$$w_0 = \mu_y - [w_1 \quad w_2] \begin{bmatrix} \mu_{a_1} \\ \mu_{a_2} \end{bmatrix} = \mu_t - \left(w_1 \mu_{a_1} + w_2 \mu_{a_2}\right). \tag{3.65}$$

In the case where a_1 and a_2 are independent, the relationship between the multivariate case and the univariate case is even more obvious. In this case, we find that

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \sigma_{a_1 a_1} & 0 \\ 0 & \sigma_{a_2 a_2} \end{bmatrix}^{-1} \begin{bmatrix} \sigma_{a_1 t} \\ \sigma_{a_2 t} \end{bmatrix} = \begin{bmatrix} \dfrac{\sigma_{a_1 t}}{\sigma_{a_1 a_1}} \\ \dfrac{\sigma_{a_2 t}}{\sigma_{a_2 a_2}} \end{bmatrix}. \tag{3.66}$$

In other words, the weights are given by the covariance of the attribute with the target divided by the covariance of the attribute. This is identical to the univariate case but is only valid when the attributes are independent. In the next section, I will discuss how to transform dependent attributes into independent attributes. As an example of our discussion, Figure 3.8 shows the regression of the target against the first two attributes shone in Figure 3.1, where (a) shows the points in three dimensions, and (b) shows the best-fit plane to these points.



(a) (b)

Figure 3.8: The regression lines for the target and first two attributes from Figure 3.1, where (a) shows the values themselves and (b) shows the two-dimensional regression, which takes the form of a plane.

3.5.4 Multilinear regression with convolutional weights

Recall that the multilinear regression equation shown in equation (3.47) used scalar weights. We can extend this equation by letting the weights be vectors. This can be written:

$$\boldsymbol{t} = w_0\boldsymbol{a}_0 + \boldsymbol{w}_1 * \boldsymbol{a}_1 + \cdots + \boldsymbol{w}_M * \boldsymbol{a}_M \,, \tag{3.67}$$

where $\boldsymbol{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$ is the target vector, $\boldsymbol{w}_i = \begin{bmatrix} w_i(0) \\ \vdots \\ w_i(L-1) \end{bmatrix}$ is an L-point weight vector with the first sample at time $t = 0$, $\boldsymbol{a}_i = \begin{bmatrix} a_{1i} \\ \vdots \\ a_{Ni} \end{bmatrix}$ is the i^{th} attribute, where $i = 1, \ldots, M$, $\boldsymbol{a_0} = \boldsymbol{1}$, the identity vector, and * denotes convolution.

The difference between equations (3.64) and (3.67) is shown schematically in Figure 3.9.

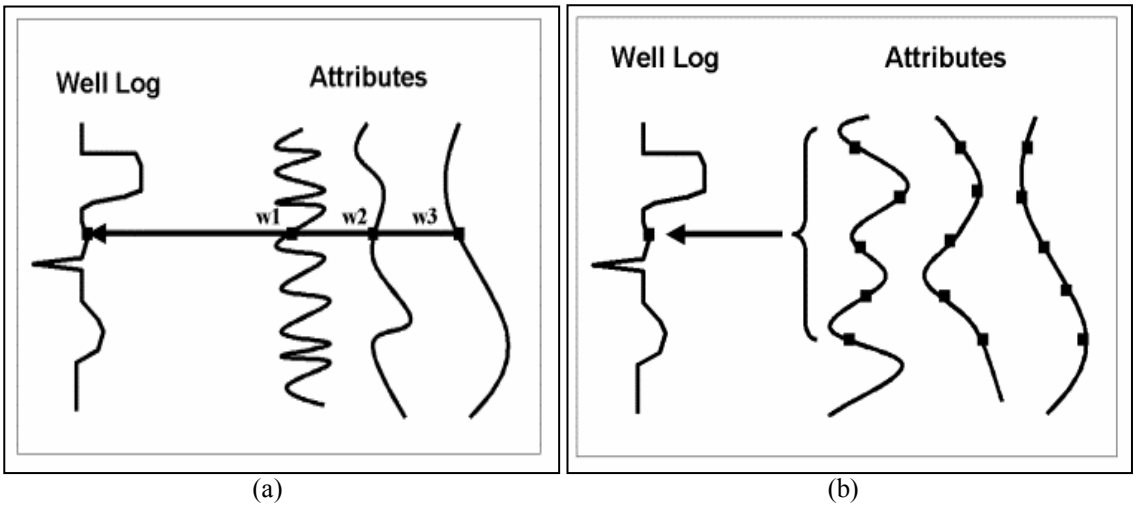


Figure 3.9: A schematic example of the difference between (a) the single-point weights given by equation (3.65) and (b) the convolutional weights given by equation (3.66) (from Hampson et al., 2001).

Convolution was discussed in Chapter 2 in the context of the convolutional model of the earth, and is also covered in detail in Appendix 3, where it is shown that linear equations 3.65 and 3.66 can both be derived from the general theory of multichannel digital filtering. As a quick review of convolution, note that if we consider the case in which we have only one attribute, the full convolutional equation can be written using matrix notation as

$$\boldsymbol{t} = w_0 \boldsymbol{a_0} + W\boldsymbol{a}\,, \tag{3.68}$$

where the convolutional matrix is written in similar fashion to the one shown in equation (2.27). However, the matrix W shown in equation (2.27) was of dimension $N+L-1$ rows by N columns, so that the output also had $N+L-1$ points, rather than N points as shown in equation 3.66. To avoid this problem, we truncate matrix W to N rows, with the term $w(0)$ in both the upper left and lower right elements of the matrix.

Let us consider an example in which $N = 4,\ L = 3$ *and* $M = 2$. That is, we have two attributes with four values each, and an operator of length three points. This gives us the equation

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = w_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} w_1(0) & 0 & 0 & 0 \\ w_1(1) & w_1(0) & 0 & 0 \\ w_1(2) & w_1(1) & w_1(0) & 0 \\ 0 & w_1(2) & w_1(1) & w_1(0) \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} + \begin{bmatrix} w_2(0) & 0 & 0 & 0 \\ w_2(1) & w_2(0) & 0 & 0 \\ w_2(2) & w_2(1) & w_2(0) & 0 \\ 0 & w_1(2) & w_2(1) & w_2(0) \end{bmatrix} \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \end{bmatrix} \tag{3.69}$$

Often, it is preferable to use a symmetrical wavelet, with time zero at the centre, as shown in Figure 3.19(b). For our three point case of equation (3.68) this would be written $\boldsymbol{w}^T = \begin{bmatrix} w(-1) & w(0) & w(+1) \end{bmatrix}$, and the resulting matrix expression is then given by the matrix equation

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = w_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} w_1(0) & w_1(-1) & 0 & 0 \\ w_1(+1) & w_1(0) & w_1(-1) & 0 \\ 0 & w_1(+1) & w_1(0) & w_1(-1) \\ 0 & 0 & w_1(+1) & w_1(0) \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} + \begin{bmatrix} w_2(0) & w_2(-1) & 0 & 0 \\ w_2(+1) & w_2(0) & w_2(-1) & 0 \\ 0 & w_2(+1) & w_2(0) & w_2(-1) \\ 0 & 0 & w_2(+1) & w_2(0) \end{bmatrix} \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \end{bmatrix} . \tag{3.70}$$

Collecting terms, we can rewrite equation (3.69) as

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = w_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + w_1(-1) \begin{bmatrix} a_{21} \\ a_{31} \\ a_{41} \\ 0 \end{bmatrix} + w_1(0) \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} + w_1(+1) \begin{bmatrix} 0 \\ a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} + \cdots , \tag{3.71}$$

where I have only considered the first attribute. The interpretation of equation (3.71) is that the effect of a convolutional operator is to create a new set of attributes that are simply shifted versions of the original attributes. Thus, the number of effective attributes in our process is the number of actual attributes multiplied by the operator length *L*.

An alternate interpretation of equation (3.71) can be found by re-expressing this equation with the weighting coefficients in column format, or

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = w_0 + \begin{bmatrix} a_{21} & a_{11} & 0 \\ a_{31} & a_{21} & a_{11} \\ a_{41} & a_{31} & a_{21} \\ 0 & a_{41} & a_{31} \end{bmatrix} \begin{bmatrix} w_1(-1) \\ w_1(0) \\ w_1(+1) \end{bmatrix} + \cdots . \tag{3.72}$$

Equation (3.72) can be solved using the generalized inverse approach. Dropping the zero weight and attribute number, and substituting equation (3.71) into equation (3.51), we get

$$\begin{bmatrix} w_1(-1) \\ w_1(0) \\ w_1(+1) \end{bmatrix} = \left\{ \begin{bmatrix} a_2 & a_3 & a_4 & 0 \\ a_1 & a_2 & a_3 & a_4 \\ 0 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} a_2 & a_1 & 0 \\ a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 \\ 0 & a_4 & a_3 \end{bmatrix} \right\}^{-1} \begin{bmatrix} a_2 & a_3 & a_4 & 0 \\ a_1 & a_2 & a_3 & a_4 \\ 0 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}, \quad (3.73)$$

or, in more complete form as

$$\begin{bmatrix} w_1(-1) \\ w_1(0) \\ w_1(+1) \end{bmatrix} = \begin{bmatrix} \sum_{j=2}^{4} a_j^2 & \sum_{j=1}^{3} a_j a_{j+1} & \sum_{j=1}^{2} a_j a_{j+2} \\ \sum_{j=1}^{3} a_j a_{j+1} & \sum_{j=1}^{4} a_j^2 & \sum_{j=1}^{3} a_j a_{j+1} \\ \sum_{j=1}^{2} a_j a_{j+2} & \sum_{j=1}^{3} a_j a_{j+1} & \sum_{j=1}^{3} a_j^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{j=2}^{4} a_j t_{j-1} \\ \sum_{j=1}^{4} a_j t_j \\ \sum_{j=1}^{3} a_j t_{j+1} \end{bmatrix}. \quad (3.74)$$

Equation (3.74) can be extended for all of the attributes and involves nonzero lag autocorrelations and cross-correlations. It is known as the Wiener-Levinson equation (Claerbout, 1976), and has Toeplitz structure. In seismic processing terms, we are de-convolving the attributes to get a better match to the log values.

3.6 A practical methodology

3.6.1 Introduction

In the preceding five sections, I have discussed the theory of multivariate statistics and multilinear regression. This gives us the theoretical basis for reservoir prediction using multiple seismic attributes. In this section, I will describe a practical implementation of this methodology. This discussion is based on work described by Russell et al. (1997) and Hampson et al. (2001). The two key problems in the analysis can be summarized as follows: which attributes should be used, and which of these attributes are statistically significant? These two questions will be addressed in the next two sections.

3.6.2 Finding the best attributes

In Chapter 2, I discussed the various seismic attributes that can be used to predict reservoir parameters. This was only a partial list, as many more attributes can be devised which are either combinations of those discussed in Chapter 2, or are based on new algorithms. But how many attributes should we use, and in what order? The procedure adopted here is to try various combinations of attributes and minimize the least-squares error between the training samples, which are the values on the logs to be predicted, and the attributes. The least-squares error criterion is given by

$$E^2 = \frac{1}{N}\sum_{j=1}^{N}(t_j - w_0 - w_1 a_{j1} - \ldots - w_M a_{jM})^2 \,. \qquad (3\text{-}75)$$

As discussed in section 3.5.4, the weights can also be vectors, which is equivalent to introducing new time-shifted attributes equal to the number of attributes multiplied by the length of the operator. The most obvious approach is to find the best set of M attributes from a total collection of T attributes. If we do this by a procedure called the "exhaustive search", in which the least-squares error is found for each possible combination and then we choose the set with the lowest error, the result is that we must compute a prohibitive number of cases.

An efficient method of finding the attributes is a technique called step-wise regression (Masters, 1995), which consists of the following steps:

(1) Find the best attribute by an exhaustive search of all the attributes, using equation (3.75) to compute the prediction error for each attribute (i.e. $M = 1$) and choosing the attribute with the lowest error.

(2) Find the best pair of attributes from all combinations of the first attribute and one other. Again, the best pair is the pair that has the lowest prediction error from equation (3.75), with $M = 2$.

(3) Find the best triplet, using the pair from step (2) and combining it with each other attribute.

(4) Continue the process as long as desired.

Step-wise regression therefore gives us a very efficient way of finding the best set of *M* attributes, since these will have the lowest least-squared error. But how do we choose the value for *M*? Actually, this method will indicate that *M* should be as large as possible, since increasing the number of attributes will either decrease the error or keep it the same. (As pointed out by Hampson et al. (2001), if this were not true, all we would have to do is set the weight of the last attribute to zero to make it true.) In the next section, we will discuss a method for finding the best value for the number of attributes.

3.6.3 Cross-validation

Step-wise regression will give us a set of attributes that is guaranteed to reduce the total error as the number of attributes goes up. So when do we stop? This is done using a technique called cross-validation, in which we leave out a training sample and then predict it from the other samples. We then re-compute the error using equation (3.75), but this time from the training sample that was left out. We repeat this procedure for all the training samples and average the error, giving us a total validation error. This computation is done as a function of the number of attributes, and the resulting graph usually shows an increase in validation error past some small number of attributes such as five or six. Rather than perform this procedure for all samples, we perform it on a well-by-well basis. This is a reasonable assumption and speeds up the process on the computer.

I will now apply all of this theory to a case study from the Blackfoot area of central Alberta.

3.7 A multiattribute case study

3.7.1 Introduction

In this section, I will use the multivariate approach just discussed to first predict well logs from combinations of other well logs, and, second, using these new well logs, predict well log parameters from multiple seismic attributes over a 3D volume. In both cases, we will be predicting S-wave sonic logs.

I will be using a seismic dataset acquired over the Blackfoot area of Alberta. A 3C-3D seismic survey was recorded in this area in October 1995, with the primary target being the Glauconitic member of the Mannville group. The reservoir occurs at a depth of around 1550 m, where Glauconitic sand and shale fill valleys incised into the regional Mannville stratigraphy. The objectives of the survey were to delineate the channel and distinguish between sand-fill and shale-fill. The well log input consists of nine wells, each with P-wave sonic, density, and gamma ray, and three with S-wave sonic. Figure 3.10 shows the distribution of wells throughout the 3D survey area, where the three wells that contain S-wave logs have been indicated with arrows.

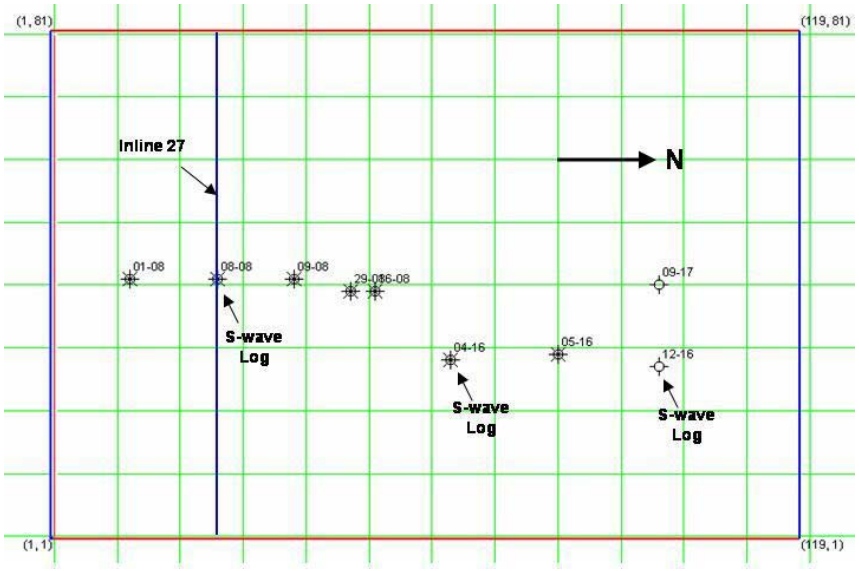


Figure 3.10: The map from the Blackfoot area showing the wells used in the study. Wells 08-08, 04-16 and 12-16 contain S-wave sonic logs.

3.7.2 Predicting S-wave curves from other log curves

We will now use the multivariate procedure to predict new pseudo-S-wave logs at each of the six well locations in which the S-wave curve has not been measured. The logs from one of the wells in which the S-wave sonic log is present, well 08-08, are shown in Figure 3.11.

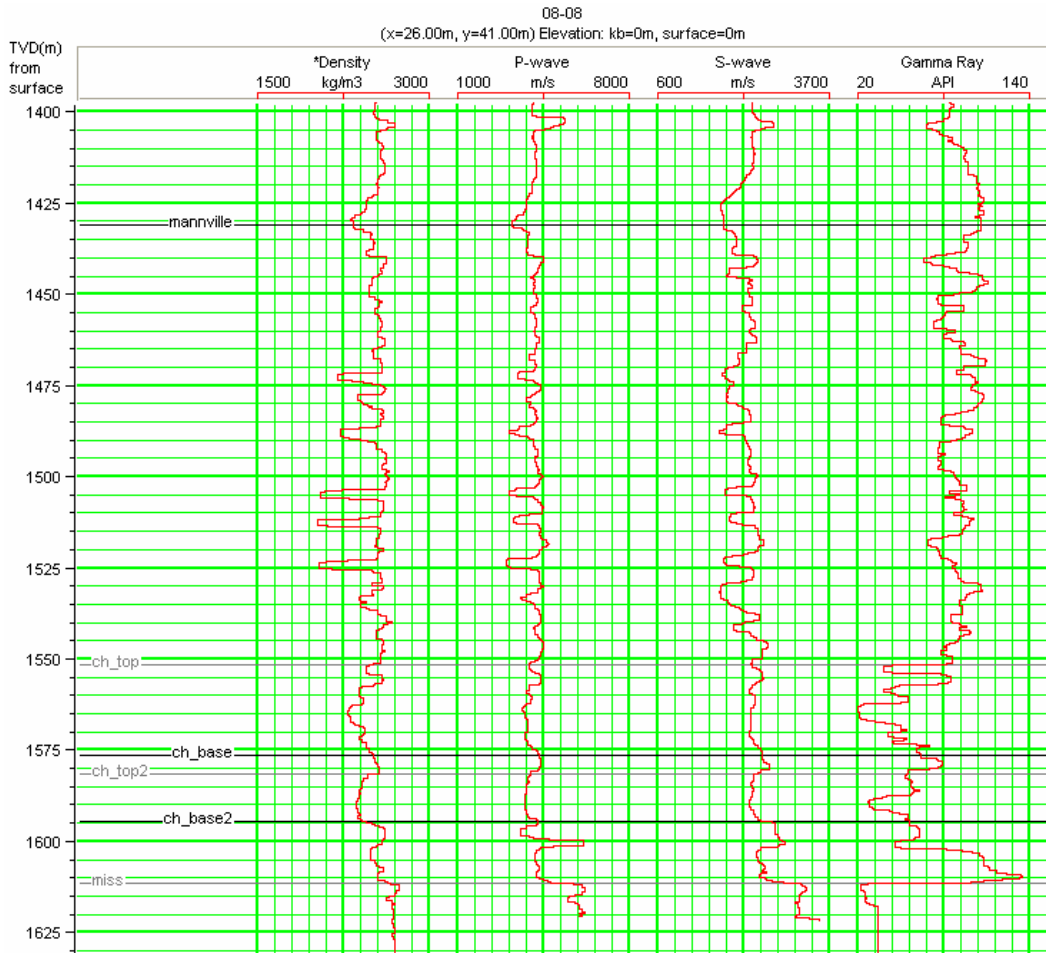


Figure 3.11: Wells 08-08, displaying the density, P-wave, S-wave, and gamma ray log curves.

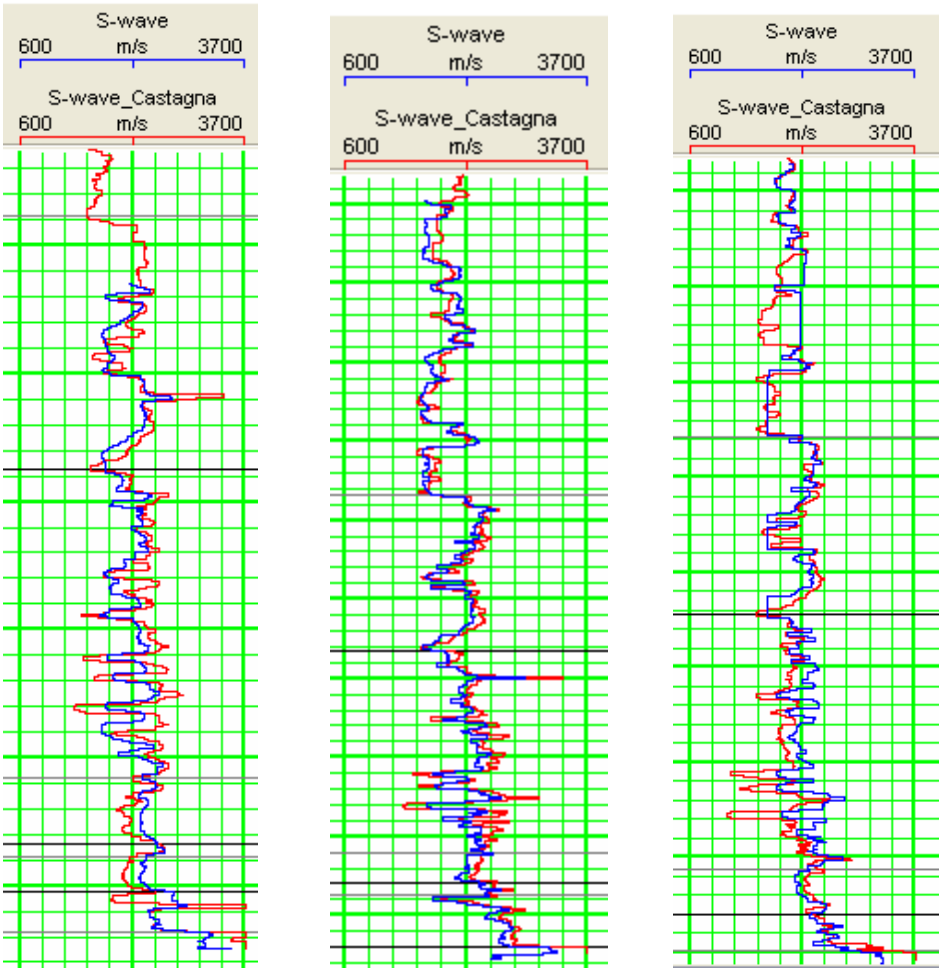
The traditional approach to S-wave curve prediction (Castagna et al., 1985) is to find the linear regression fit between the P-wave and S-wave curves given by

$$V_S = a + bV_P . \tag{3.76a}$$

The coefficients derived in the above reference were given as

$$V_P = 1360\ m/s + 1.16 V_S, \tag{3.76b}$$

which can be re-arranged for V_S as the dependent variable. Figure 3.12 shows the application of equation (3.76b) to the P-wave sonic logs from each of the three wells, with the correct S-wave log superimposed on the computed log. Although the fit for the wells is reasonable in Figure 3.12, we could do better by finding a local fit. In fact, we will extend our analysis beyond just a fit between P-wave and S-wave logs.



(a) (b) (c)

Figure 3.12: The application of equation (3.76) using the coefficients derived by Castagna et al. (1985), where (a) shows well 04-16, (b) shows well 08-08, and (c) shows well 12-16. In all cases the blue line shows the original S-wave log and the red line shows the computed S-wave log.

Figures 3.13 through 3.16 show the correlations between S-wave log and the density log, gamma ray log and P-wave log for wells 04-16, 08-08, and 12-16. For each of these regressions, we can write the generalized form of equation (3.64) as

$$V_S = a + bL, \tag{3.77}$$

where L represents an arbitrary log. In addition to the a and b coefficients, we will also derive the correlation coefficient and RMS error for each of the regressions.

For well 04-16, shown in Figure 3.13, it is obvious that the best fit in a least-squares sense is with the P-wave curve. It would appear that the second best is with the gamma ray curve, and the third best is with the density curve.

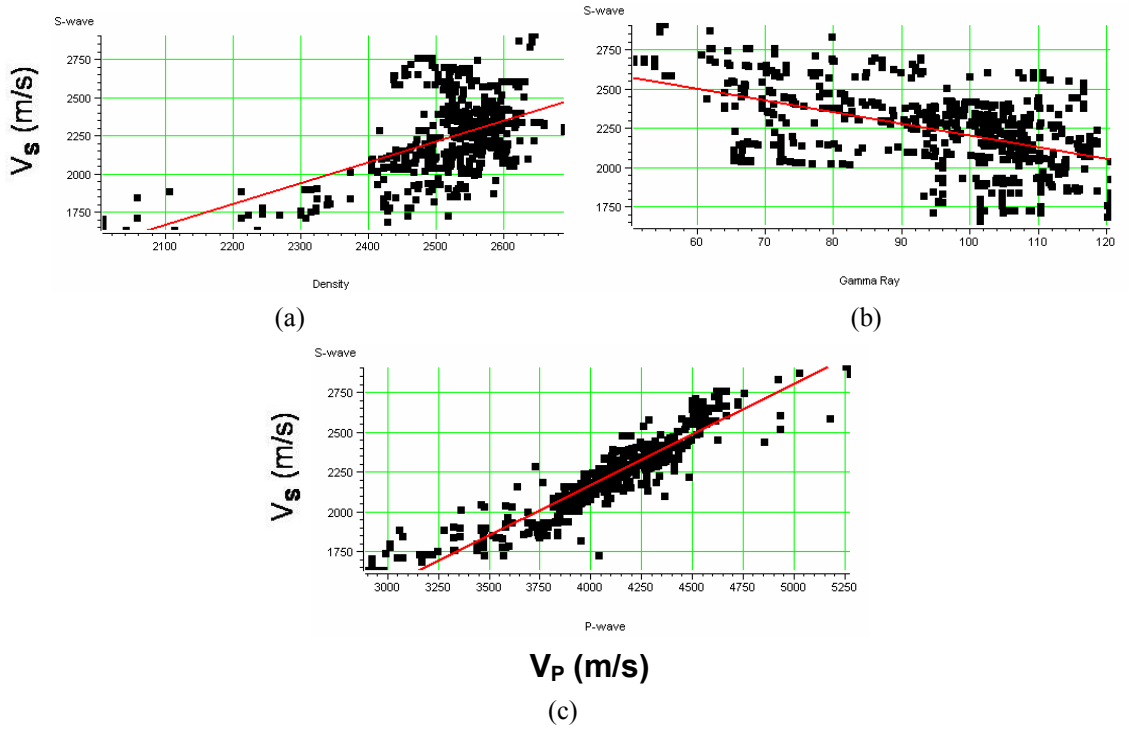


Figure 3.13: The regression fits to the S-wave velocity log from the curves for well 04-16, between the Mannville and Mississippian tops, where (a) is versus density, (b) is versus gamma ray, and (c) is versus P-wave velocity.

However, when we look at the correlation coefficients and errors given in Table 3.3, we see that the density fit is actually slightly better than the gamma ray fit. Notice that the gamma ray has a negative correlation with the S-wave log.

S-wave velocity vs:	P-wave velocity	Density	Gamma Ray
Intercept (*a*)	-366.95	-1182.76	2948.41
Slope (*b*)	0.634	1.357	-7.434
Correlation Coeff.	0.9305	0.5030	-0.4845
RMS Error	92.251	217.683	220.329

Table 3.3: Regression parameters for well 04-16.

The crossplots for well 08-08 are shown in Figure 3.14, and the correlation coefficients and errors are given in table 3.4. Notice that the P-wave fit is the best, the density fit is second, and the gamma ray fit is the third best .

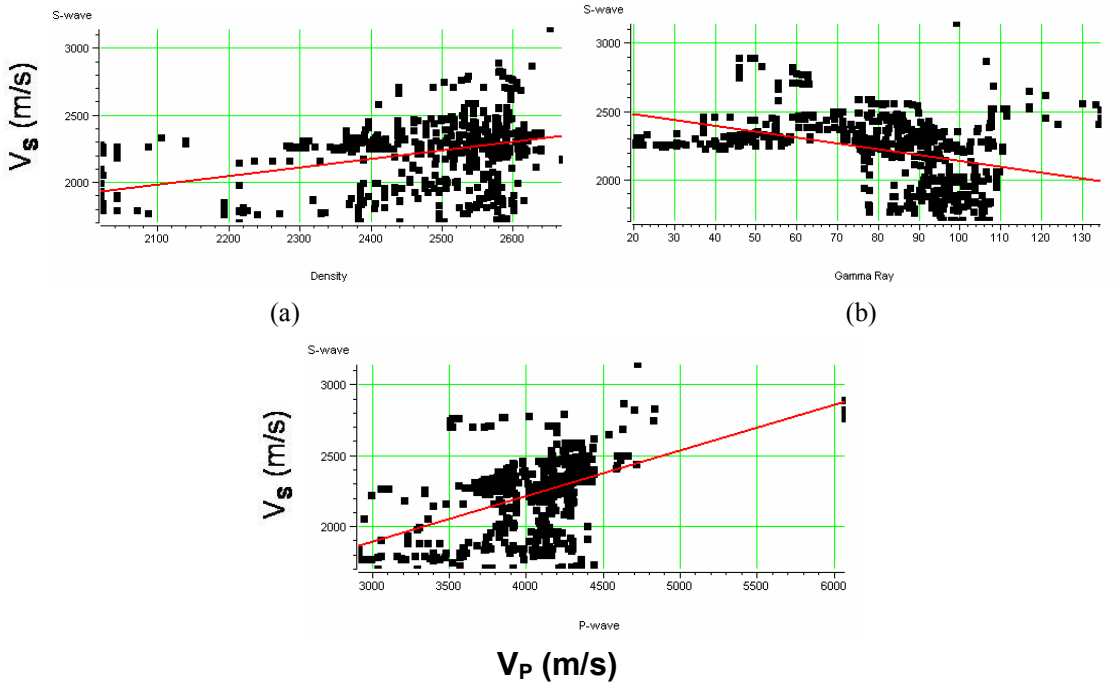


Figure 3.14: The regression fits to the S-wave velocity log from the curves for well 08-08, between the Mannville and Mississippian tops, where (a) is versus density, (b) is versus gamma ray, and (c) is versus P-wave velocity.

S-wave velocity vs:	P-wave velocity	Density	Gamma Ray
Intercept (*a*)	182.507	-482.073	2541.01
Slope (*b*)	0.496	1.070	-3.586
Correlation Coeff.	0.7766	0.5376	-0.2460
RMS Error	140.992	188.716	216.921

Table 3.4: Regression parameters for well 08-08.

Finally, the crossplots for well 12-16 are shown in Figure 3.15, and the correlation coefficients and errors given in table 3.5. Note now that the P-wave fit is the best, the gamma ray fit is now second, and the density is third. Both the density and gamma ray logs have correlation coefficients below 0.5, indicating a poor fit.

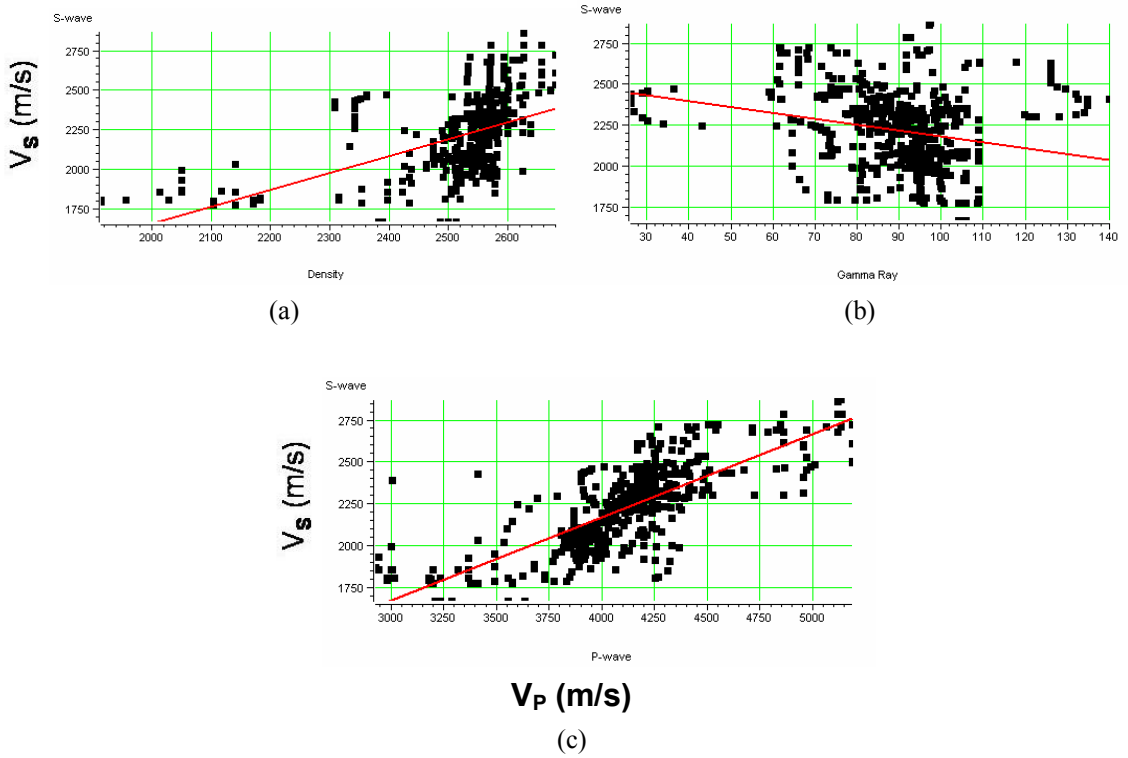


Figure 3.15: The regression fits to the S-wave velocity log from the curves for well 12-16, between the Mannville and Mississippian tops, where (a) is versus density, (b) is versus gamma ray, and (c) is versus P-wave velocity.

S-wave velocity vs:	P-wave velocity	Density	Gamma Ray
Intercept (*a*)	932.143	661.755	2567.9
Slope (*b*)	0.321	0.631	-4.254
Correlation Coeff.	0.4901	0.3283	-0.3864
RMS Error	214.734	232.702	227.22

Table 3.5: Regression parameters for well 12-16.

The result of combining all three wells is shown in Table 3.6, arranged in order of increasing RMS error and decreasing correlation coefficient. As expected, the P-wave log correlates best, followed by the density log, followed by the gamma ray log. This would suggest that we will find the coefficients for the multilinear regression given by

$$V_S = a + bV_P + c\rho + d\gamma, \tag{3.78}$$

where ρ = density, and γ= gamma ray.

Target	Attribute	Error	Correlation
S-wave	P-wave	165.154755	0.730677
S-wave	Density	218.033646	0.433163
S-wave	Gamma Ray	226.306885	-0.353285

Table 3.6: Regression parameters for all three wells, where the S-wave target is S-wave velocity and the P-wave attribute is P-wave velocity.

The log attributes and coefficients in equation (3.78) were determined using the techniques described in the last section, in which cross-validation is used to determine the optimal ordering of attributes. The results of performing the linear multi-attribute analysis are shown in Figure 3.16, which shows the graphical training and validation error, and Table 3.7, which shows the numerical errors. Note that the training error is the error using all three wells in the prediction, and the cross-validation error is the error when the well to be predicted is left out of the training.

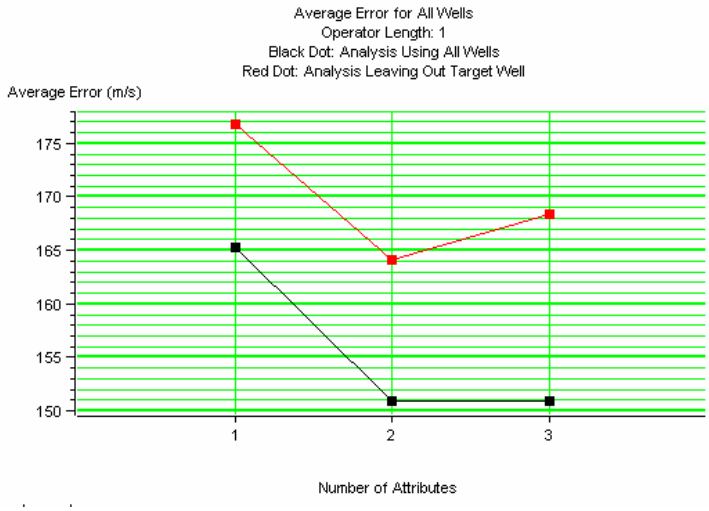


Figure 3.16: The multi-regression fits to the S-wave log from the curves from wells 12-16, 08-08, and 04-16, between the Mannville and Mississippian tops, where the training error is shown by the black dots and validation error is shown by the red dots).

.	Target	Final Attribute	Training Error	Validation Error
1	S-wave	P-wave	165.332158	176.785774
2	S-wave	Gamma Ray	150.936775	164.085386
3	S-wave	Density	150.927284	168.372343

Table 3.7: The numerical results of the analysis shown in Figure 3.16.

Two results are clear from Figure 3.16 and Table 3.7. First, when the multilinear regression technique is used, the gamma ray log is the second best log attribute to use. Second, the validation error shows that the error increases when the density log is included, indicating that the optimum fit is found by using only the P-wave and gamma ray logs in the multi-linear regression. The computed coefficients were

$$V_S = 656.47 + 0.461 V_P - 3.505\,\gamma\,, \tag{3.79}$$

The negative coefficient for the gamma ray log in equation (3.79) is due to the negative correlation of the gamma ray log with the S-wave log.

A better fit can often be introduced by applying nonlinear functions, such as the inverse, log, square and square root, to the attributes before performing the regression fit. Table 3.8 shows the numerical error for this fit.

.	Target	Final Attribute	Training Error	Validation Error
1	S-wave	P-wave	165.332158	176.785774
2	S-wave	Sqrt(Gamma Ray)	150.507382	162.204631
3	S-wave	1 / (Density)	150.461666	166.217145

Table 3.8: The numerical results when nonlinear functions were applied to both the target and the attribute in the analysis of Figure 3.16.

The optimum nonlinear functions were found to be the square root of gamma ray and the inverse of density. The density log is also again seen to increase the validation error, and therefore is dropped. The computed regression coefficients are

$$V_S = 893.42 + 0.465 V_P - 60.46 \sqrt{\gamma}\,, \tag{3.80}$$

As a comparison with Castagna's equation (3.76), the regression coefficients were also computed using the P-wave velocity alone, and were found to be

$$V_S = 269.125 + 0.480 V_P \tag{3.81}$$

The fit between the resulting pseudo-sonic logs and the original logs in the three wells is shown in Figures 3.17 and 3.18, where Figure 3.17 shows the training results and Figure 3.18 shows the validation result. In both cases, equation (3.79) is shown in (a), and equation (3.80) is shown in (b).

By comparing Figures 3.17(a) and (b) it is obvious that the addition of the gamma ray log has improved the fit. The improvement is quite small in well 04-16, where the correlation between S-wave and P-wave values was high, but is noticeable in the deeper section of well 08-08. Also, notice that the correlation coefficient has gone from 0.73, in the single regression case, to 0.78, in the multiple regression case.

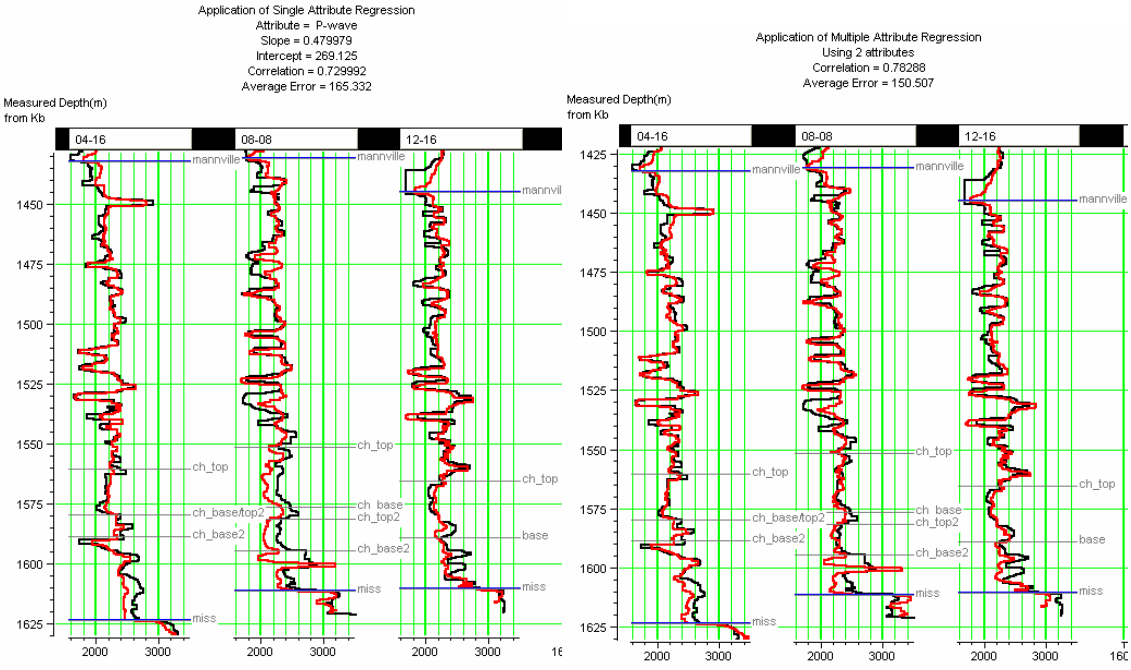


Figure 3.17: The result of applying the training results for (a) equation (3.72), and (b) equation (3.71), where the black lines show the original logs and the red lines show the computed logs.

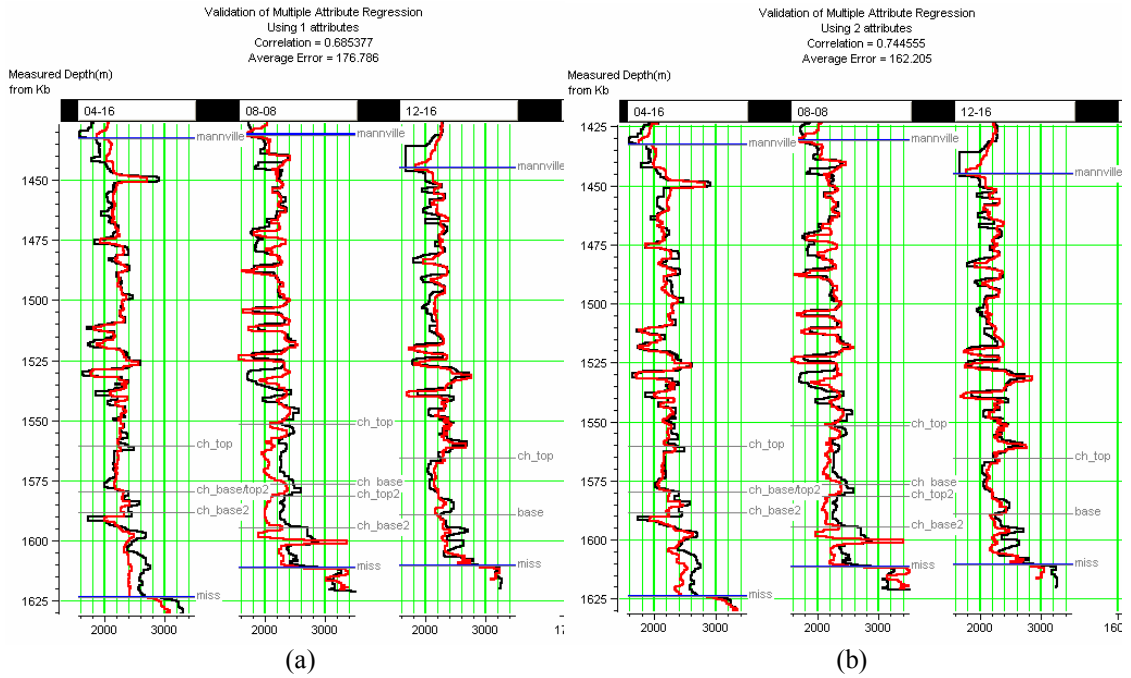


Figure 3.18: The result of applying the validation results for (a) equation (3.72), and (b) equation (3.71), where the black lines show the original logs and the red lines show the computed logs.

Figure 3.18, which shows the validation results, also shows improvement between the single regression case and the multiple regression case. As expected, the correlation coefficients are slightly worse in this case, and the correlation coefficient has now improved from 0.68, in the single regression case, to 0.74 in the multiple regression case.

Equation (3.71) was then applied to the P-wave and gamma ray logs in the other six wells shown in Figure 3.20, to produce pseudo-S-wave sonic logs in each well. These pseudo-well logs will be used in the next part of this study.

3.7.3 Seismic analysis, inversion, and AVO

Our next objective is to predict pseudo-well log curves at each seismic trace location. Three steps are involved in the prediction of these pseudo-well log curves. The first step involves extracting the composite seismic at each well location (an average of the traces in a circular radius around the well) and the correlation of the well log to this composite trace, using the sonic log to build the depth-time relationship. The second step is the training step, in which we build the multi-linear relationship between the well log curve and the seismic trace attributes. The third step involves applying the multilinear relationship to the seismic trace locations which do not have wells, using the seismic attributes derived at these locations.

Having created pseudo-S-wave sonic logs in the six wells of Figure 3.10 that did not originally contain S-wave logs, we now have a full suite of logs: P-wave, S-wave, density, and gamma ray, for each of the nine wells shown on that map. I will now use the multilinear regression approach to create pseudo-logs for each of the traces in the seismic volume shown on the map. Pseudo-volumes will be created for two of the well log curves: S-wave and P-wave sonic.

Let us first examine the input seismic data that will be used for the analysis. Figure 3.19 shows a set of CDP super-gathers over inline 27 from the map in Figure 3.10, with the P-wave sonic from well 08-08 spliced in at its correct location. These supergathers were created by performing partial stacks over five separate offset ranges in the original dataset, and then re-grouping these offset stacks into a set of gathers.

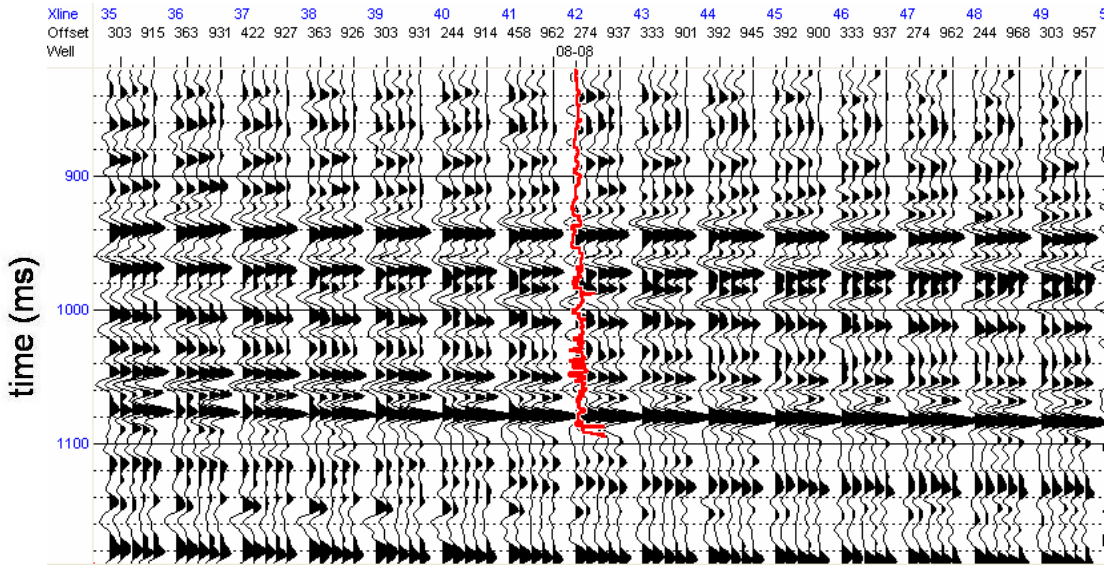


Figure 3.19: The supergathers from inline 27 on the map in Figure 3.10

The stack of the CDP gathers is shown in Figure 3.20. Note that the P-wave sonic log has again been spliced in at its tie point, and I have picked two of the seismic horizons of interest, the Mannville and Lower Mannville. The elliptical region highlights an area that will be later compared to the R_{P0} section in Figure 3.34

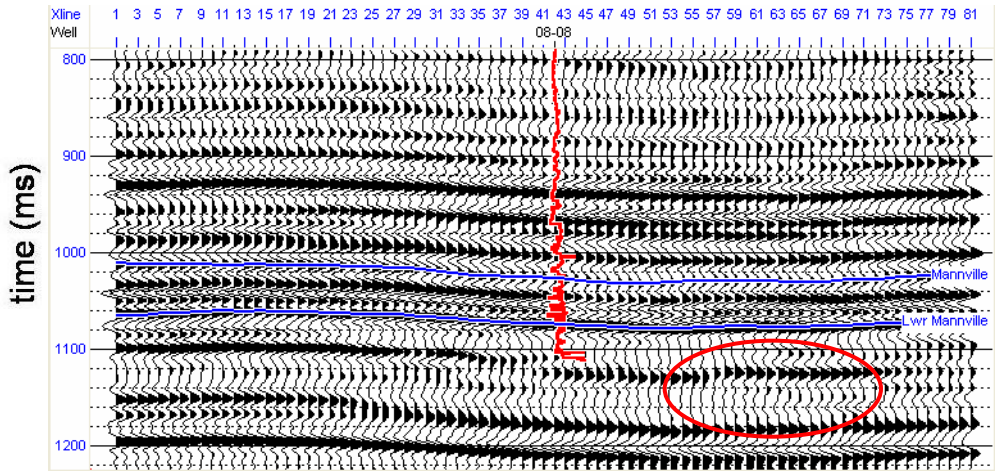
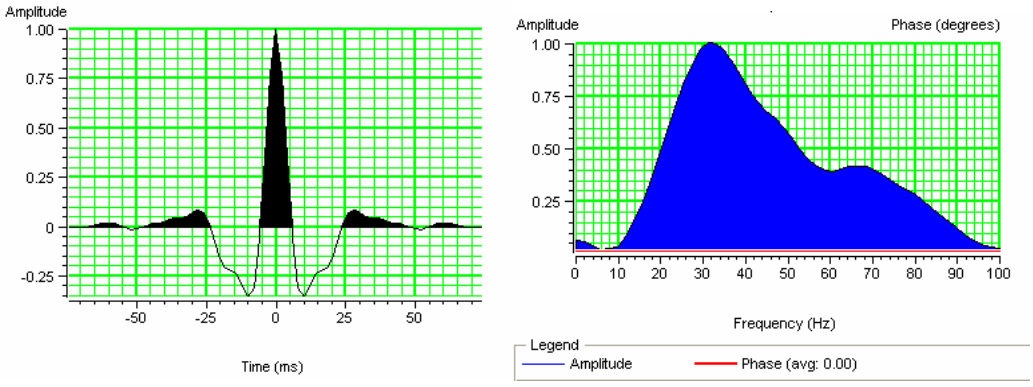


Figure 3.20: The stack of the supergathers from inline 27 shown in Figure 3.19. The elliptical region highlights an area that will be compared later to the R_{P0} section in Figure 3.34.

The P-wave sonic log shown in Figures 3.19 and 3.20 has been correlated with the seismic data. Correlation of the P-wave sonic log with the seismic data involves 2 steps. First, we estimate an optimum wavelet from the seismic data. Figure 3.21(a) shows the wavelet that was extracted from the seismic data using the autocorrelation method, which assumes that since the reflectivity has a white spectrum the autocorrelation of the seismic trace is equivalent to the autocorrelation of the wavelet. By taking the autocorrelation of the trace, we have lost the phase information about the wavelet, so we must assume we know the phase. In this case, we have assumed a zero phase wavelet. Figure 3.21(b) shows the amplitude and phase spectra of the wavelet.



(a) (b)

Figure 3.21: (a) The extracted wavelet from the stack of Figure 3.20, and (b) the amplitude (blue) and phase (red) spectra of the wavelet.

In the second step, the wavelet shown in Figure 3.21 is used to compute a synthetic seismogram with the reflectivity derived from the seismic, as discussed in section 2.8.1. We then correlate the log to the seismic, as shown in Figure 3.22.

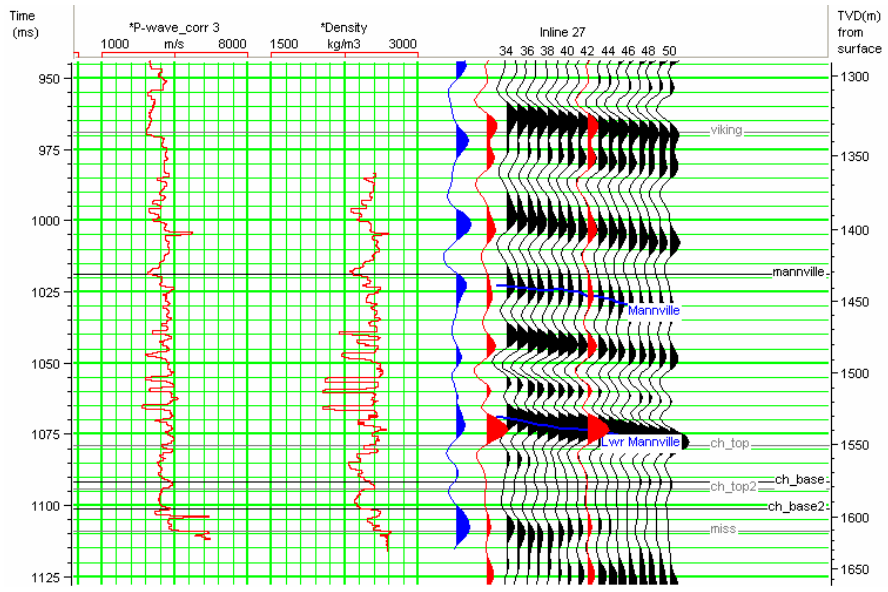


Figure 3.22: The log correlation procedure for well 08-08. The sonic and density logs are shown on the left, and the seismic tie is shown on the right.

In Figure 3.22 the blue trace shown in the centre has been created by convolving the wavelet shown in Figure 3.21 with the reflectivity derived from the sonic and density logs shown on the left of the figure. The red trace is extracted by averaging the seismic traces in a radius from around the well. As discussed in Chapter 2 on seismic attributes, stacking is performed to increase the signal-to-noise ratio of the seismic data. However, it gives us the average amplitude of the CDP gather, and does not correspond to a physically meaningful reflectivity. Using equation (2.20) for extracting AVO attributes, I therefore extracted the P-wave reflectivity (R_{P0}) and the S-wave reflectivity (R_{S0}) from the seismic gathers shown in Figure 3.19. Figure 3.23 shows the P-wave reflectivity. Although quite similar to the stack shown in Figure 3.20, there are some differences. One such difference has been highlighted by the elliptical region shown on the figure.

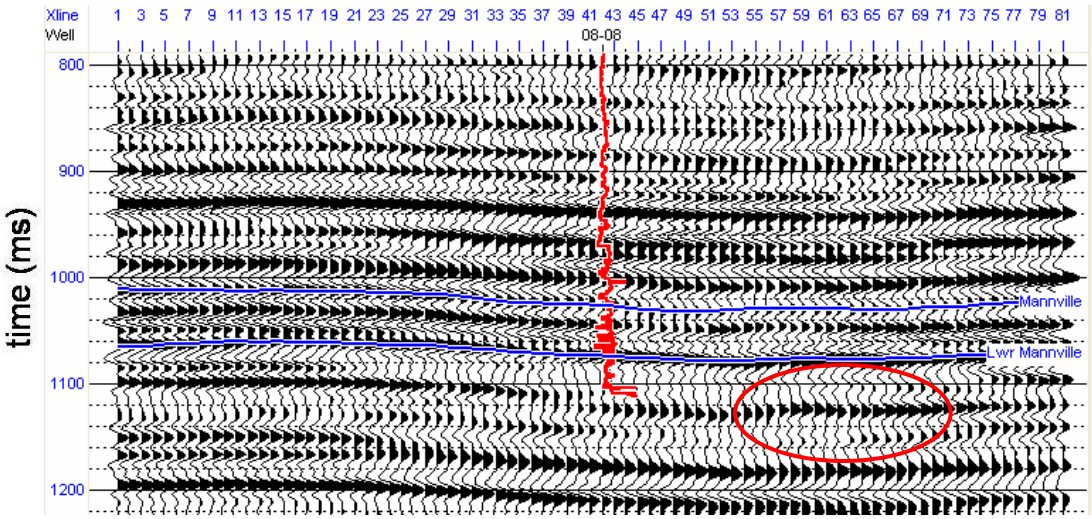


Figure 3.23: The P-wave intercept (R_P) of the supergathers from inline 27 shown in Figure 3.19. The elliptical region highlights a difference with respect to the stack in Figure 3.20.

Figure 3.24 shows the pseudo-S-wave reflectivity, again extracted using equation (2.20). It is important to note that the S-wave reflectivity responds quite differently to the subsurface than the P-wave reflectivity. This can be seen by comparing the seismic data under the picked events. For example, the S-wave reflectivity close to the well at the Lower Mannville event on the S-wave section in Figure 3.24 is a trough, whereas the P-wave reflectivity on the same event in Figure 3.23 is a peak.

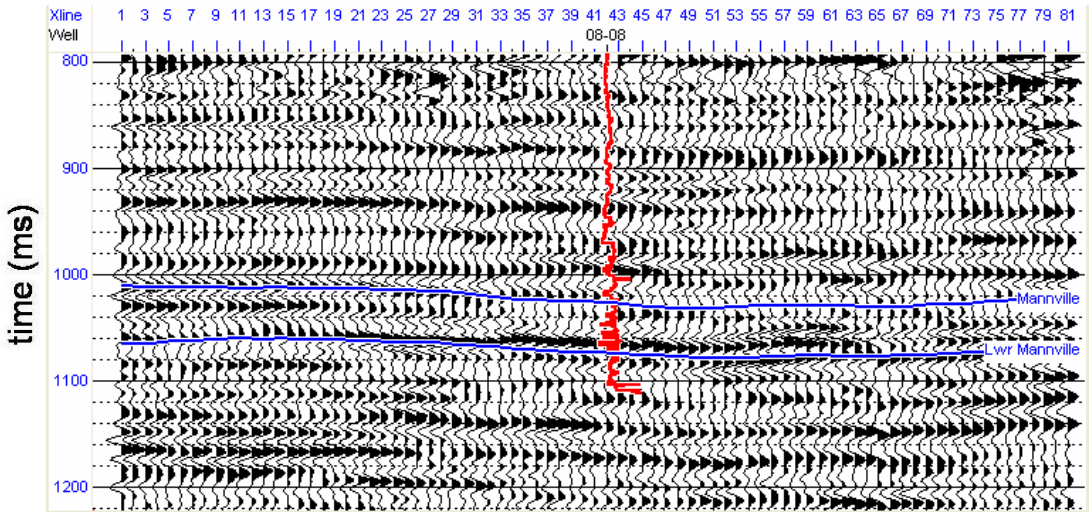


Figure 3.24: The pseudo-S-wave (R_{S0}) section derived from the supergathers from inline 27 shown in Figure 3.19.

Now that we have extracted both the P-wave and S-wave reflectivity, they can be inverted to P and S-impedance, using the model-based inversion scheme described in Chapter 2. Figure 3.25 shows the inverted P-wave impedance volume. The inserted curve is the P-wave velocity.

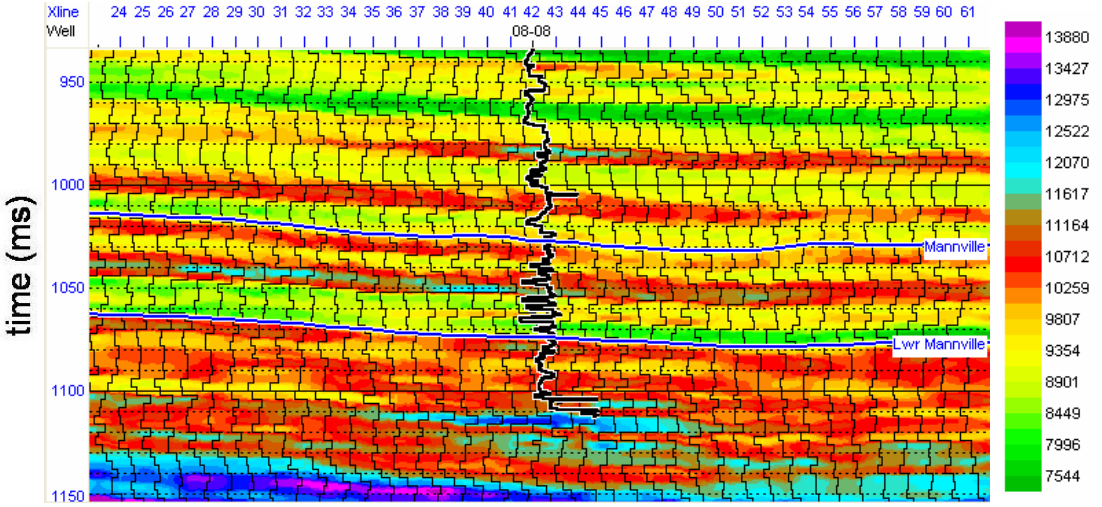


Figure 3.25: The inverted P-wave impedance section derived from the R_{P0} volume in Figure 3.23. The colour bar on the right displays impedance values in units of m/s·g/cc.

Figure 3.26 next shows the inverted S-wave impedance volume. The inserted curve is the S-wave sonic log.

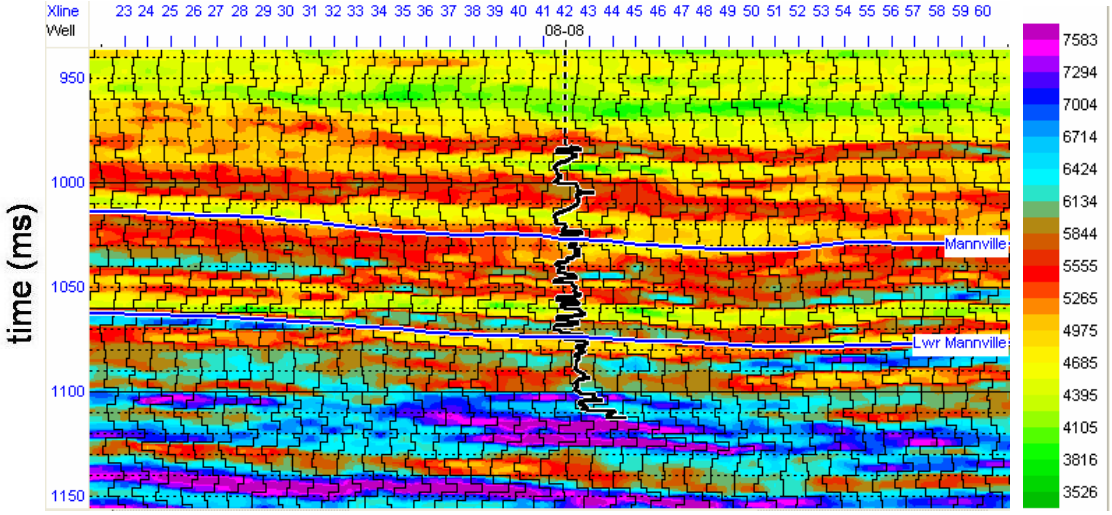


Figure 3.26: The inverted S-wave impedance section derived from the R_{S0} volume in Figure 3.24. The colour bar on the right displays impedance values in units of m/s·g/cc.

3.7.4 Creating S-wave pseudo-logs over the seismic volume

I will now use the multilinear regresssion approach to predict pseudo-logs, starting with the prediction of the S-wave log over the complete seismic volume. The input to the S-wave prediction consists of the nine S-wave sonic logs created in the last section, the R_S stack of Figure 3.24 and the inverted S-wave volume of Figure 3.26. To compute the weighting coefficients, I used a 7-point convolutional set of weights. The results of the training analysis are shown in Table 3.9 and in Figure 3.27, where the table shows the actual attributes, and the figure shows the errors.

.	Target	Final Attribute
1	S-wave	S_Impedance
2	S-wave	Filter 35/40-45/50
3	S-wave	Filter 25/30-35/40
4	S-wave	Integrate
5	S-wave	Average Frequency
6	S-wave	Amplitude Weighted Frequency
7	S-wave	Apparent Polarity

Table 3.9: The computed attributes for the creation of pseudo-S-wave logs by multilinear regression.

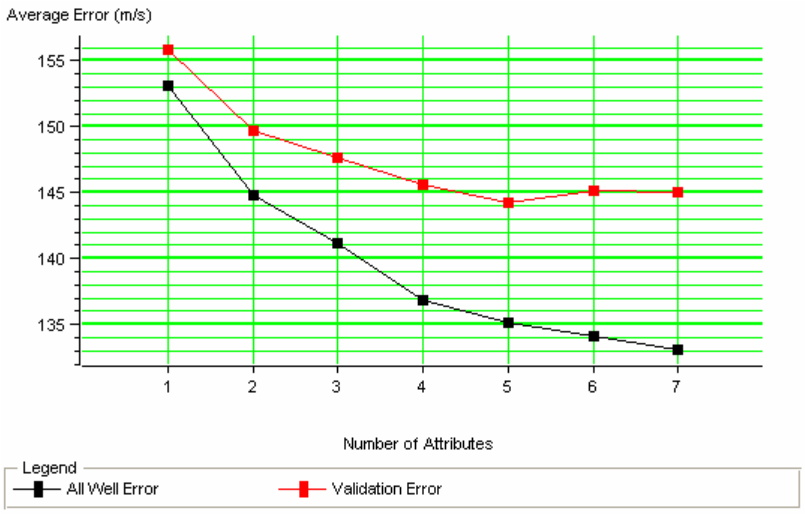
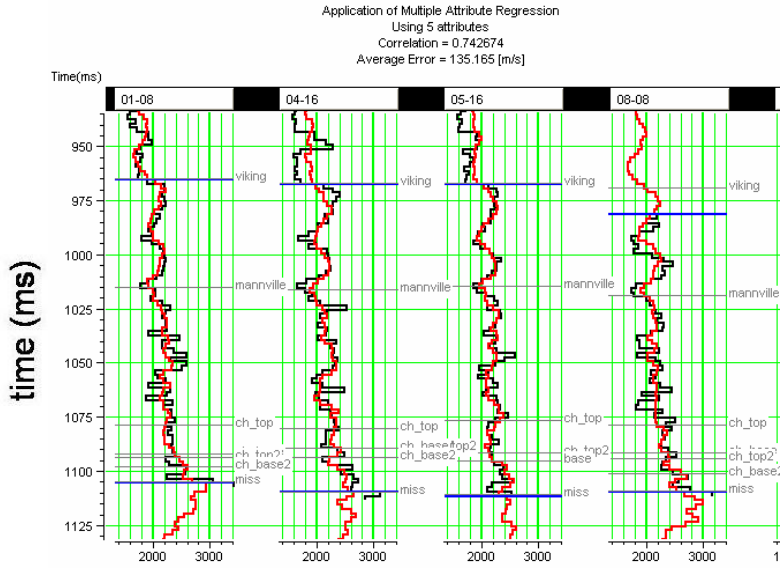
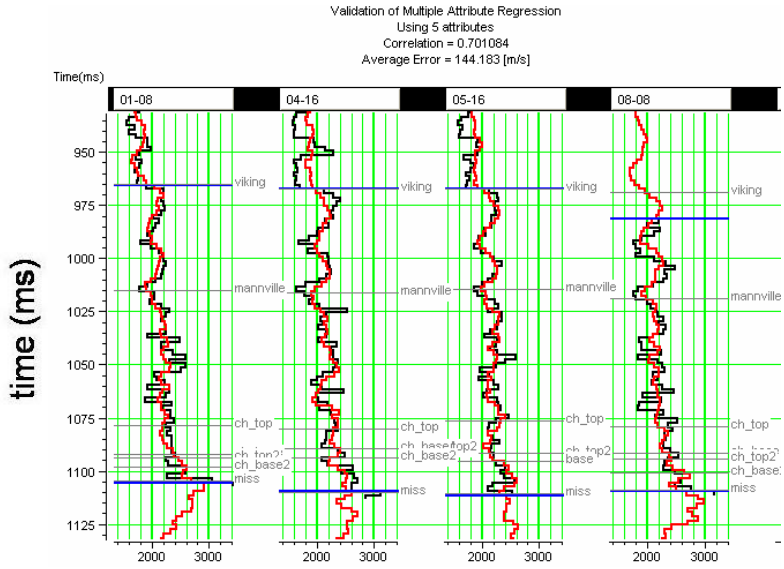


Figure 3.27: The training error (black dots) and validation error (red dots) for the creation of pseudo-S-wave logs by multilinear regression

By observing the validation error, we note that only the first five attributes are statistically significant, since the error starts to increase after this point. These five attributes were then used to create estimates of the pseudo-S-wave logs at the nine wells ties. The first four well ties are shown in Figure 3.28, where (a) shows the training result, and (b) shows the validation result.



(a)



(b)

Figure 3.28: The creation of pseudo-S-wave logs at the well ties, showing (a) the training result and (b) the validation result. The black curves are the true logs, and the red curves are the predicted logs.

Finally, Figure 3.29 shows the application of the multilinear regression coefficients to the attributes along line 27 to create pseudo-S-wave-velocity curves. Notice that the section is now given in m/s, rather than in impedance units. Also, the shear-wave sonic log from well 08-08 has been spliced in, showing an excellent fit at the well tie.

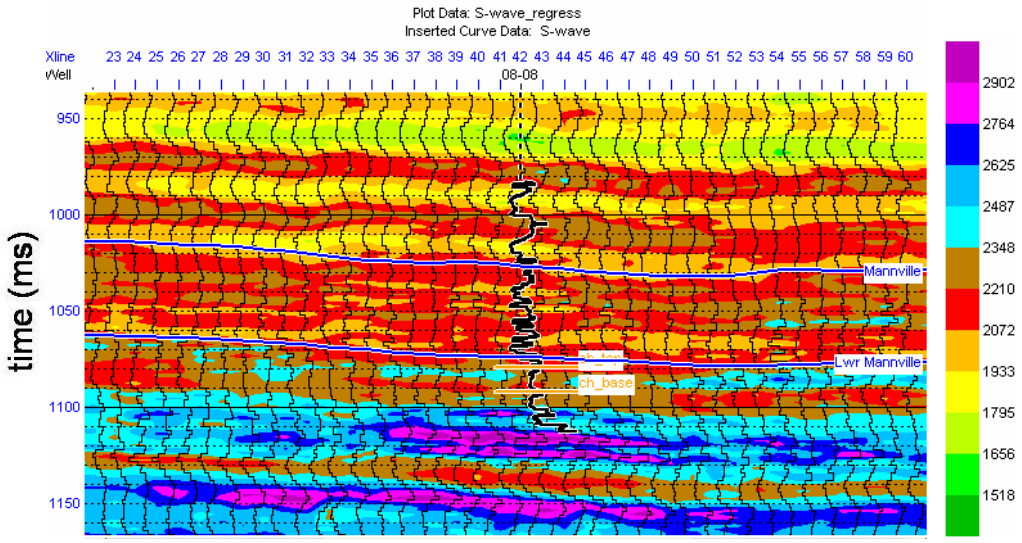


Figure 3.29: The predicted pseudo-S-wave velocity section for line 27. The colour bar on the right displays impedance values in units of m/s.

3.7.4 Creating P-wave pseudo-logs over the seismic volume

I will now use the multiattribute approach to predict the pseudo-P-wave-velocity log over the seismic volume. The input consists of the nine measured P-wave sonic logs, the R_{P0} stack of Figure 3.23 and the inverted P-wave volume of Figure 3.25. As with the S-wave study, a convolutional set of weights were used with an operator length of 7 samples. The results of the training analysis are shown in Table 3.10 and Figure 3.30 where the table shows the list of attributes and the figure shows the training and validation error. By observing the validation error, we can note that only the first four attributes are statistically significant. These attributes are the P-impedance, one filter slice of the R_{P0} stack, the integrated absolute amplitude of the R_P stack, and a second filter slice of the R_{P0} stack.

.	Target	Final Attribute
1	P-wave	P-impedance
2	P-wave	Filter 15/20-25/30
3	P-wave	Integrated Absolute Amplitude
4	P-wave	Filter 45/50-55/60
5	P-wave	Derivative Instantaneous Amplitude
6	P-wave	Integrate
7	P-wave	Apparent Polarity

Table 3.10: A list of the attributes used to predict P-wave velocity.

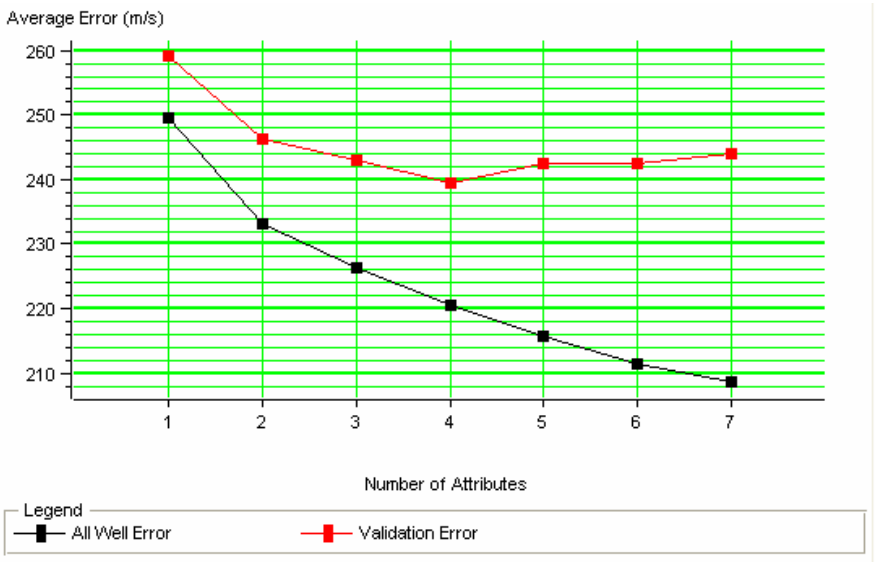
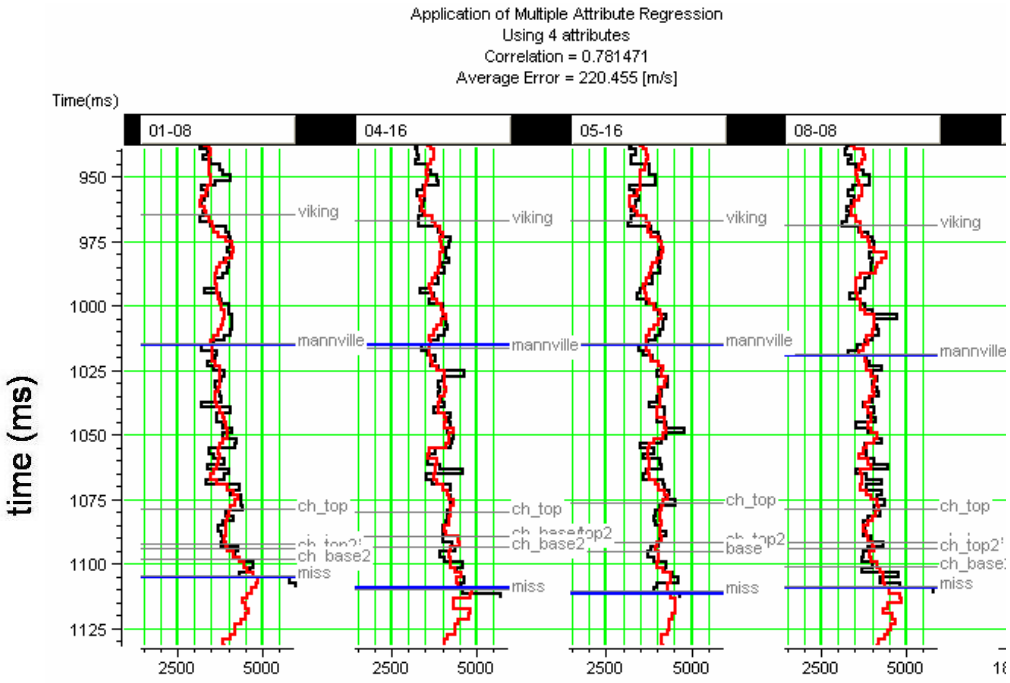
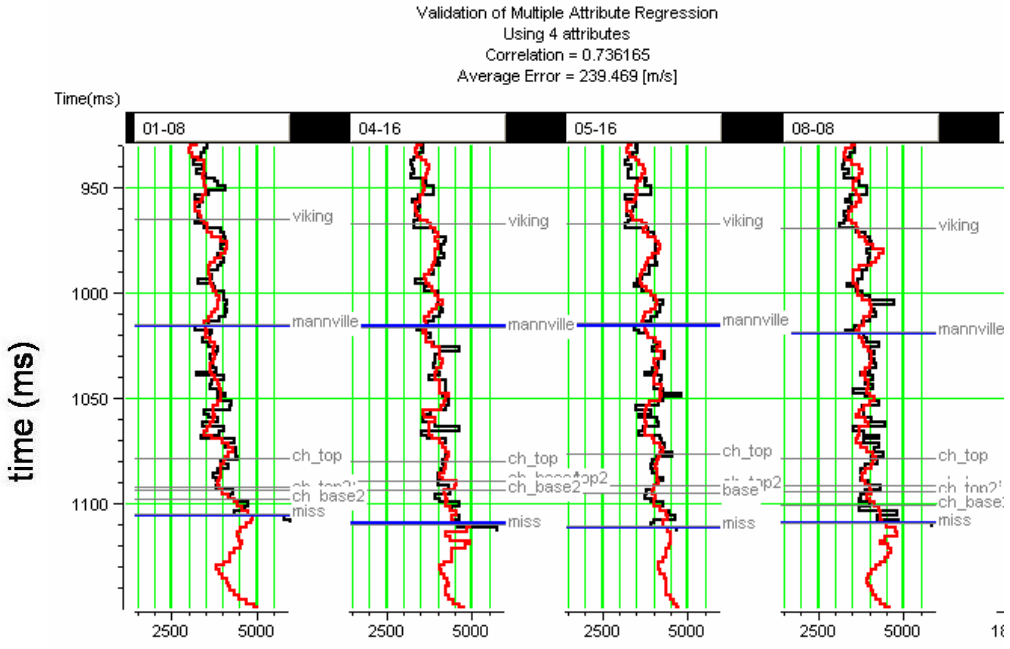


Figure 3.30: The multi-regression analysis for the creation of pseudo-P-wave logs, where the training error is shown by the black dots and the validation error by the red dots.

These four attributes were then used to create estimates of the pseudo-P-wave-velocity logs at the nine wells ties. The first four well ties are shown in Figure 3.31, where (a) shows the training result, in which all wells were used in the prediction, and (b) shows the validation result, in which the well being predicted has been left out of the training.



(a)



(b)

Figure 3.31: The creation of pseudo-P-wave-velocity logs at the well ties, where (a) shows training result with all wells, and (b) shows the validation result. The black curves are the true logs, and the red curves are the predicted logs.

Figure 3.32 shows the application of the multilinear regression coefficients to the attributes along line 27, to create pseudo-P-wave curves. The section is now given in m/s, rather than in impedance units. Also, the P-wave sonic log from well 08-08 has been spliced in, again showing an excellent fit at the well tie.

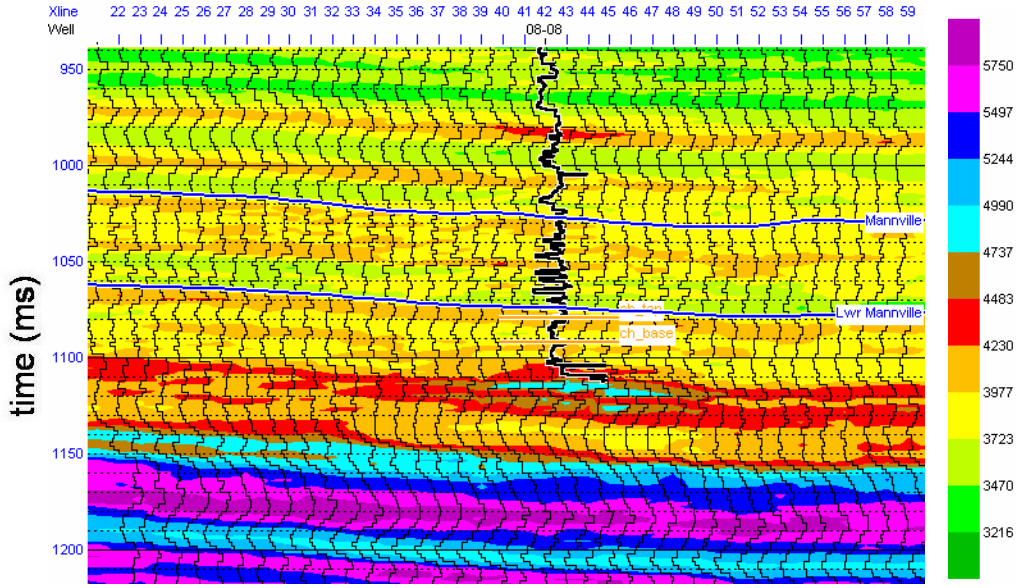


Figure 3.32: The predicted pseudo-P-wave-velocity section for line 27. The colour bar on the right displays impedance values in units of m/s.

3.8 Principal components analysis

In our study so far, I have chosen a combination of M attributes with which to predict our reservoir parameter of interest. Often, it is advantageous to reduce this number to L attributes, where $L < M$. This becomes the statistical problem of mapping a higher-dimensional space into a lower-dimensional space, and can be solved using principal components analysis (Johnson and Wichern, 1998, Bishop, 1995). In this section, I will explain the principal components analysis method as it applies to our problem, and also show an example based on the datasets displayed in the earlier sections.

Note that our input data can be represented by the N x M matrix

$$X = \begin{bmatrix} \boldsymbol{x_1} & \cdots & \boldsymbol{x_N} \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{MN} \end{bmatrix} = A^T = \begin{bmatrix} \boldsymbol{a_1^T} \\ \vdots \\ \boldsymbol{a_M^T} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{N1} \\ \vdots & \ddots & \vdots \\ a_{1M} & \cdots & a_{NM} \end{bmatrix}, \quad (3.82)$$

where N is the number of seismic samples in our training dataset and M is the number of input attributes.

One obvious approach to reducing the dimensionality of the problem is to remove the last few rows from matrices X and A^T in equation (3.82). That is, simply reduce the number of attributes. However, since we have spent a lot of effort finding both the optimum number of attributes and their order, this would not be a very good approach. A better approach is to design a transform matrix U such that we transform input matrix X into a new matrix P by the linear operation

$$P = U^T X, \quad (3.83)$$

where P is called the principal component matrix, and can be written

$$P = \begin{bmatrix} \boldsymbol{p_1} & \cdots & \boldsymbol{p_N} \end{bmatrix} = \begin{bmatrix} p_{11} & \cdots & p_{1N} \\ \vdots & \ddots & \vdots \\ p_{M1} & \cdots & p_{MN} \end{bmatrix} = Q^T = \begin{bmatrix} \boldsymbol{q_1^T} \\ \vdots \\ \boldsymbol{q_M^T} \end{bmatrix} = \begin{bmatrix} q_{11} & \cdots & q_{N1} \\ \vdots & \ddots & \vdots \\ q_{1M} & \cdots & q_{NM} \end{bmatrix}. \quad (3.84)$$

In equation (3.84) the transpose of matrix P has been given a different name, Q, to emphasize the fact that there is a one-to-one correspondence between P and X, and Q and A, when comparing equations (3.84) and (3.82). Also, note that P is composed of N vectors $\boldsymbol{p_j}$, and Q is composed of M vectors $\boldsymbol{q_i}$, again in direct comparison with the vectors $\boldsymbol{x_j}$ and $\boldsymbol{a_i}$. By referring to Figure 3.1, we can therefore think of the $\boldsymbol{p_j}$ vectors as transformed sample vectors $\boldsymbol{x_j}$ and the $\boldsymbol{q_i}$ vectors as transformed attribute vectors $\boldsymbol{a_i}$. The $\boldsymbol{q_i}$ vectors, which are the rows of matrix P, are referred to as the principal components.

The transform matrix U is a rectangular matrix of dimension M x M and can be written

$$U = \begin{bmatrix} \boldsymbol{u_1} & \cdots & \boldsymbol{u_M} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{1M} \\ \vdots & \ddots & \vdots \\ u_{M1} & \cdots & u_{MM} \end{bmatrix} \Rightarrow U^T = \begin{bmatrix} \boldsymbol{u_1^T} \\ \vdots \\ \boldsymbol{u_M^T} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{M1} \\ \vdots & \ddots & \vdots \\ u_{1M} & \cdots & u_{MM} \end{bmatrix}. \quad (3.85)$$

To find the optimum values for the matrix U, note that by combining equations (3.83) through (3.85), we can write the transposed $\boldsymbol{q_i}$ vectors individually as

$$\begin{aligned} \boldsymbol{q_1^T} &= \boldsymbol{u_1^T} X = u_{11}\,\boldsymbol{a_1^T} + u_{12}\,\boldsymbol{a_2^T} + \cdots + u_{1M}\,\boldsymbol{a_M^T} \\ \boldsymbol{q_2^T} &= \boldsymbol{u_2^T} X = u_{21}\,\boldsymbol{a_1^T} + u_{22}\,\boldsymbol{a_2^T} + \cdots + u_{2M}\,\boldsymbol{a_M^T} \\ &\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad\quad \vdots \\ \boldsymbol{q_M^T} &= \boldsymbol{u_M^T} X = u_{M1}\boldsymbol{a_1^T} + u_{M2}\boldsymbol{a_2^T} + \cdots + u_{MM}\boldsymbol{a_M^T} \end{aligned} \quad (3.86)$$

From equation (3.86) it can be seen that the transformed attribute vectors are given either as the product of the transposed $\boldsymbol{u_i}$ vector and the sample matrix X, or by the weighted sum of the original attribute vectors $\boldsymbol{a_i}$, where the weights are the individual components of the $\boldsymbol{u_i}$ vectors. From a standard textbook in multivariate statistics (Johnson and Wichern, 1998) it can be shown that the variance of vector $\boldsymbol{q_i}$ can be written

$$\sigma_i^2 = \boldsymbol{u_i^T} \Sigma \boldsymbol{u_i}, \quad (3.87)$$

and that the covariance of two vectors $\boldsymbol{q_i}$ and $\boldsymbol{q_k}$ can be written

$$\sigma_{ik} = \boldsymbol{u_i^T} \Sigma \boldsymbol{u_k}, \quad (3.88)$$

where Σ is the covariance matrix of X, assuming that X has zero mean. The $\boldsymbol{u_i}$ vectors are then found which maximize the variances of each of the principal components $\boldsymbol{q_i}$ and set the covariance values to zero, with the extra condition that the vectors are orthonormal. This can be written

$$\boldsymbol{u_i^T}\ \boldsymbol{u_k} = \delta_{ik} = \begin{cases} 1, i = k, \\ 0, i \neq k. \end{cases} \quad (3.89)$$

We have already encountered such a set of vectors in our discussion in section 3.4.4 on the eigendecomposition of the multivariate Gaussian distribution. These were the normalized eigenvectors. Note that if we multiply each side of the earlier equation (3.42) by $\boldsymbol{u}_i^T$, we get the equivalent of equation (3.80). That is:

$$\boldsymbol{\Sigma}\,\boldsymbol{u}_i = \lambda_i\,\boldsymbol{u}_i \Rightarrow \boldsymbol{u}_i^T\boldsymbol{\Sigma}\,\boldsymbol{u}_i = \boldsymbol{u}_i^T\lambda_i\,\boldsymbol{u}_i = \lambda_i\ , \tag{3.90}$$

where we have used the orthonormal condition given in equation (3.89). Therefore, the maximized variances are equivalent to the eigenvalues of matrix *X*. The first principal component computed in equation (3.90) is thus derived by using the eigenvector associated with the largest eigenvalue, and so on. Once the principal component transform has been computed, we can achieve our dimensionality reduction by dropping principal components based on a sum-of-error criterion (Bishop, 1995) that uses the eigenvalues. That is, we choose the $L - M$ smallest eigenvalues such that

$$E = \frac{1}{2}\sum_{i=L+1}^{M}\lambda_i\ . \tag{3.91}$$

Several other points can be noted. First, the transform given in equation (3.83) is fully invertible. That is, we can transform matrix *P* back to matrix *X* simply by multiplying by *U*. This can be seen as follows:

$$U\,P = U\,U^T X = X\ , \tag{3.92}$$

since $UU^T = I$. Second, the matrix $\boldsymbol{\Sigma}$ can be reconstructed using the formula

$$\Sigma = U\Lambda U^T\ , \tag{3.93}$$

where $\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_M \end{bmatrix}$ is a matrix that contains the eigenvalues along its main diagonal and zeros elsewhere.

Let us now consider a real data example of the preceding theory. Figure 3.33 shows a well log curve to be predicted on the left, and seven attributes to its right. This figure is an extension of Figure 3.1, which shows only the first three attributes.

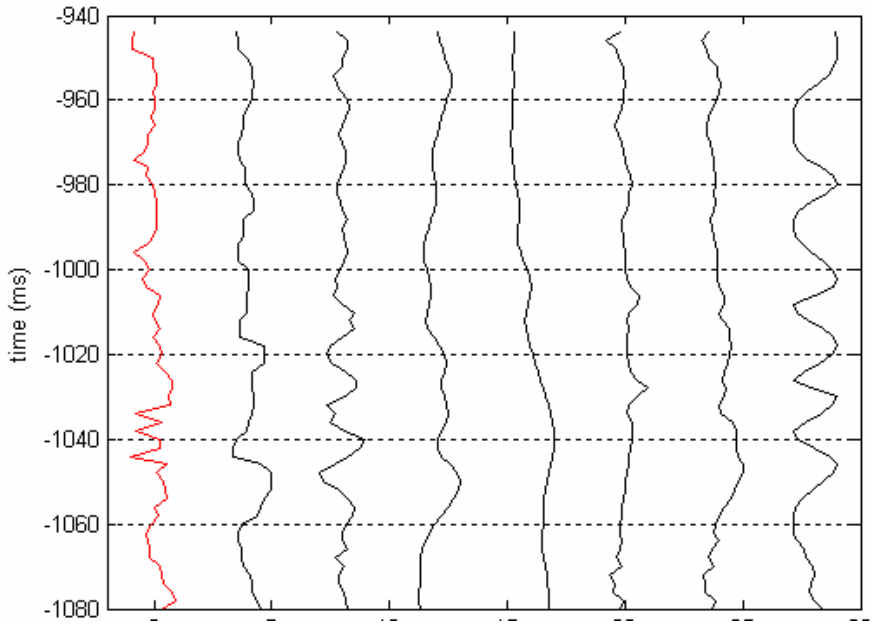


Figure 3.33: The red trace on the left shows the well log curve to be predicted and the seven curves on the right show the extracted attributes.

Taking the covariance matrix of the seven attributes shown in Figure 3.33, and computing the eigenvalues of eigenvectors of this matrix, gives us

$$\Lambda = \begin{bmatrix} 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.4478 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2255 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1553 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0537 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0262 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0027 \end{bmatrix},$$

and

$$U = \begin{bmatrix} 0.5004 & -0.0451 & -0.1324 & -0.1233 & -0.6533 & -0.4930 & -0.2120 \\ -0.0178 & -0.6306 & 0.2408 & 0.7178 & -0.0330 & -0.1659 & 0.0120 \\ 0.1188 & 0.4771 & 0.8220 & 0.1054 & 0.0759 & -0.2038 & -0.1557 \\ 0.5011 & -0.0974 & -0.1624 & -0.0817 & 0.7288 & -0.4131 & 0.0672 \\ 0.4889 & -0.0415 & -0.0281 & 0.1425 & 0.0756 & 0.6238 & -0.5859 \\ 0.4944 & 0.0772 & 0.1153 & 0.0988 & -0.1715 & 0.3381 & 0.7635 \\ -0.0242 & 0.5962 & -0.4565 & 0.6494 & 0.0014 & -0.1175 & -0.0074 \end{bmatrix},$$

where the eigenvalues have been sorted from largest to smallest and scaled so that the largest eigenvalue is equal to 1.0. Figure 3.34 is a display of the seven principal components of the attributes shown in Figure 3.33. That is, using equation (3.83) the transpose of matrix U is multiplied times the matrix X, which contains the values of the attributes as its seven rows. The resulting matrix P contains the attributes shown in Figure 3.33 as its seven rows. The principal components shown in Figure 3.34 have been ordered from largest on the left to smallest on the right. We can now use a subset of these principal components to reconstruct the well log curve shown on the far left.

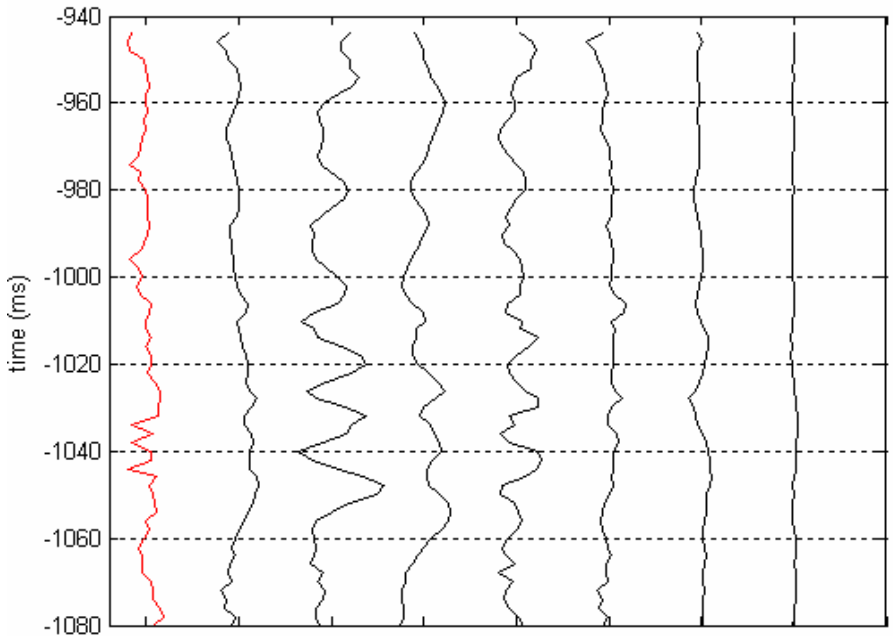


Figure 3.34: The red trace on the left shows the well log curve to be predicted and the seven curves on the right show the principal components computed from the seven attributes of Figure 3.33. These principal components are ordered from largest to smallest.

CHAPTER 4 : LINEAR METHODS FOR CLASSIFICATION

4.1 Introduction

In the previous chapter I discussed the mathematical and statistical basis of multi-linear regression methods as they relate to the prediction of reservoir parameters from seismic attributes. In this chapter I will discuss linear classification methods. As stated in Chapter 1, there is a close relationship between regression and classification. In regression problems, a continuous set of values is assigned to our inputs. That is, for each vector of attributes at a given time sample, we calculate a unique output value. In classification problems, we assign the output values to a discrete set of classes. The number of classes is determined in advance, using a training dataset. In this chapter I will be performing supervised classification, in which we use a training dataset, rather than unsupervised classification, in which we look for natural classes within the data.

The term *discrimination* is almost synonymous with the term *classification*. Discrimination is used to perform classification and involves computing a linear discriminant function. Classification is a more general nonlinear problem and is based in Bayesian statistics. I will therefore first look at the general theory of Bayesian classification and then show that the linear discriminant function can be thought of a subset of Bayesian theory in which we assume that each class has similar statistics. I will then derive the Fisher linear discriminant and apply this approach to a seismic example.

I will then show how the single-layer perceptron neural network can be thought of as a type of linear discriminant, and use a simple AVO classification problem to illustrate this method and its limitations. Finally, I will discuss the generalized linear discriminant.

4.2 Bayesian classification

4.2.1 Theory

In this classification technique, we attempt to classify the multiattribute samples in a seismic volume, $\boldsymbol{x}_j$, into the classes C_k, where $k = 1, \ldots, K$. To determine the class membership of an arbitrary input vector $\boldsymbol{x}$, I introduce the concept of the discriminant function $y_k(\boldsymbol{x})$ (Duda et al., 2001). Membership of $\boldsymbol{x}$ in a particular class C_k is found if

$$y_k(\boldsymbol{x}) > y_j(\boldsymbol{x}), \text{ for all } j \neq k. \tag{4.1}$$

In the general case, the decision boundary between two classes is found by setting $y_k(\boldsymbol{x}) = y_j(\boldsymbol{x})$. Note that for the two class case, we can write:

$$y(\boldsymbol{x}) = y_2(\boldsymbol{x}) - y_1(\boldsymbol{x}) = 0, \tag{4.2}$$

as the boundary between the two classes.

To find the discriminant function, I will use Bayes' Theorem (Duda et al., 2001). In Appendix 4 an overview of Bayes' Theorem is given, using both discrete and continuous examples. The general form of Bayes' Theorem can be written

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}, \tag{4.3}$$

where $P(A)$ = the unconditional probability of event A, $P(B)$ = the unconditional probability of event B $P(A \mid B)$ = the conditional probability of A given B, and $P(B \mid A)$ = the conditional probability of B given A. When we apply Bayes' Theorem to the classification problem defined above, we find that the posterior probability $P(C_k \mid \boldsymbol{x})$ for the k^{th} cluster can be written as:

$$P(C_k \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid C_k)P(C_k)}{p(\boldsymbol{x})}, \tag{4.4}$$

where $p(\boldsymbol{x}) = \sum_{k=1}^{K} P(\boldsymbol{x} \mid C_k)P(C_k)$, $P(\boldsymbol{x} \mid C_k)$ is the likelihood of $\boldsymbol{x}$ given c_k, $P(C_k)$ is the

prior probability of class C_k, and $p(\boldsymbol{x})$ is a normalization factor, often called the evidence. A common form of y_j is given by taking the logarithm of equation (4.4), or

$$y_k(\boldsymbol{x}) = \ln[P(C_k \mid \boldsymbol{x})] = \ln[P(\boldsymbol{x} \mid C_k)] + \ln[P(C_k)], \tag{4.5}$$

where I have dropped the normalization factor since it is common to each function.

To implement Bayes' Theorem, we must assume some probability distribution for the likelihood function. A common approach is to use the multivariate normal distribution, which was discussed at length in Chapter 3. However, I will now assume that each class has its own mean and covariance function. That is

$$p(\boldsymbol{x} \mid C_k) = \frac{1}{(2\pi)^{M/2}\left|\Sigma_k\right|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_k})^T \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu_k})\right], \tag{4.6}$$

where $\boldsymbol{\mu_k}$ is the M-dimensional vector of means of the k^{th} cluster, given by

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{j=1}^{N_k} \boldsymbol{x}_j = \begin{bmatrix} \mu_{1k} \\ \mu_{2k} \\ \vdots \\ \mu_{Mk} \end{bmatrix},$$

and Σ_k is the $M \times M$ dimensional covariance matrix given by

$$\Sigma_k = \frac{1}{N_k}\left[\boldsymbol{x_1}-\boldsymbol{\mu}_k, \boldsymbol{x_2}-\boldsymbol{\mu}_k, \ldots, \boldsymbol{x}_{N_k}-\boldsymbol{\mu}_k\right]\begin{bmatrix} \boldsymbol{x_1}-\boldsymbol{\mu}_k \\ \boldsymbol{x_2}-\boldsymbol{\mu}_k \\ \vdots \\ \boldsymbol{x_N}-\boldsymbol{\mu}_k \end{bmatrix} = \begin{bmatrix} \sigma_{11}^k & \sigma_{12}^k & \cdots & \sigma_{1M}^k \\ \sigma_{21}^k & \sigma_{22}^k & \cdots & \sigma_{2M}^k \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{M1}^k & \sigma_{M2}^k & \cdots & \sigma_{MM}^k \end{bmatrix}.$$

Substituting equation (4.6) into equation (4.5) leads to

$$y_k(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_k})^T \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu_k}) - \frac{M}{2}\ln(2\pi) - \frac{1}{2}\ln\left|\Sigma_k\right| + \ln P(C_k), \tag{4.7}$$

which can be rewritten as the general quadratic form:

$$y_k(\boldsymbol{x}) = \boldsymbol{x}^T W_k \boldsymbol{x} + \boldsymbol{w}_k^T \boldsymbol{x} + w_{0k}, \tag{4.8}$$

where $W_k = -\frac{1}{2}\Sigma_k^{-1}$, $\boldsymbol{w}_k = \Sigma_k^{-1}\boldsymbol{\mu}_k$, and $w_{0k} = -\frac{1}{2}\boldsymbol{\mu}_k^T\Sigma_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\ln\left|\Sigma_k\right| + \ln P(C_k)$.

The discriminant function between two arbitrary clusters, *k* and *j*, can therefore be written (Duda et al., 2001):

$$y(\boldsymbol{x}) = y_k(\boldsymbol{x}) - y_j(\boldsymbol{x}) = \boldsymbol{x}^T W \boldsymbol{x} + \boldsymbol{w}^T \boldsymbol{x} + w_0, \tag{4.9}$$

where $W = W_k - W_j$, $\boldsymbol{w} = \boldsymbol{w}_k - \boldsymbol{w}_j$, *and* $w_0 = w_{0k} - w_{0j}$.

For the general case in which each cluster of points has a different covariance matrix, the separation between clusters would be defined by quadratic functions (Duda et al., 2001), which can take the shape of hyperplanes, hyperspheres, hyperellipsoids, etc. However, there is a simpler case that will lead to a linear discriminant function. If we assume that the covariance matrices for each cluster are identical (or $\Sigma_k = \Sigma$), we find that matrix *W* in equation (4.8) becomes independent of *k*. The discriminant function between two arbitrary clusters, *k* and *j*, can therefore be written

$$y(\boldsymbol{x}) = y_k(\boldsymbol{x}) - y_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0, \tag{4.10}$$

where $\boldsymbol{w} = \Sigma^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_j)$, and $w_0 = -\frac{1}{2}\boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \frac{1}{2}\boldsymbol{\mu}_j^T \Sigma^{-1} \boldsymbol{\mu}_j + \ln \frac{P(C_k)}{P(C_j)}$. Equation (4.10) is called the linear discriminant function and is identical in form to the linear regression equation (3.49) discussed in the previous chapter. However, as explained in Chapter 1, the linear discriminant function represents the boundary between two sets of points, not the regression of the target against an attribute.

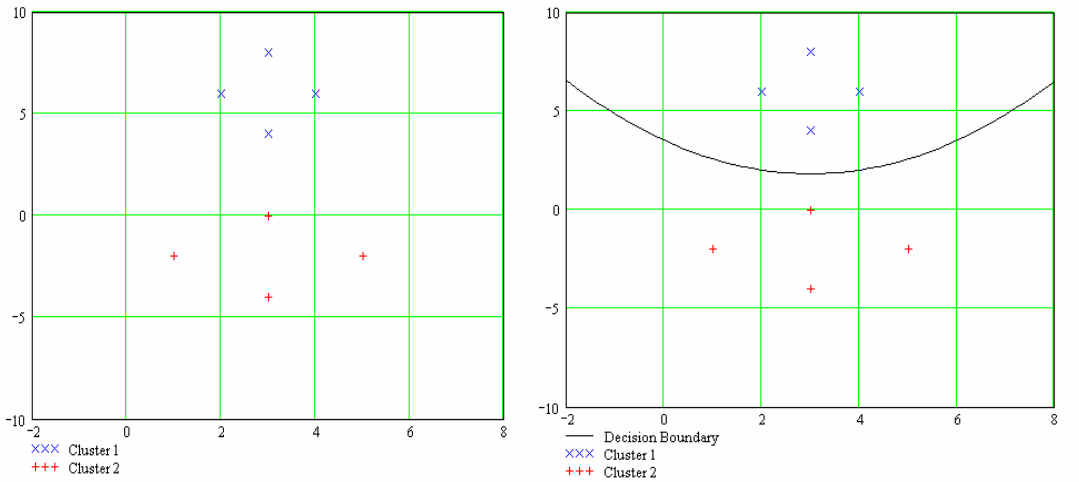
4.2.2 Two cluster example

Let us now apply the theory from the last section to a simple two-dimensional, two-cluster example. In this case, equation (4.9) gives us

$$\begin{aligned} y(\boldsymbol{x}) &= y_2(\boldsymbol{x}) - y_1(\boldsymbol{x}) = \boldsymbol{x}^T W \boldsymbol{x} + \boldsymbol{w}^T \boldsymbol{x} + w_0 \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} \\ w_{12} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + w_0, \\ &= w_{11}x_1^2 + 2w_{12}x_1x_2 + w_{22}x_2^2 + w_1x_1 + w_2x_2 + w_0 \end{aligned} \tag{4.11}$$

where $W = W_2 - W_1$, $\boldsymbol{w} = \boldsymbol{w}_2 - \boldsymbol{w}_1$, *and* $w_0 = w_{02} - w_{01}$. Setting this function to zero will give us the boundary between the two clusters.

Let us now consider a straightforward numerical example, from Duda et al. (2001), to illustrate the theory. Figure 4.1(a) shows two two-dimensional clusters, each containing four points.



(a) (b)

Figure 4.1: A simple classification example, where (a) shows two four-point clusters in two dimensions, and (b) shows the calculated decision boundary shown. The units on the axes are arbitrary.

To compute the decision boundary shown in Figure 4.2(b), notice that the first cluster contains the points $\boldsymbol{x}_1^{(1)} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $\boldsymbol{x}_2^{(1)} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$, $\boldsymbol{x}_3^{(1)} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$ and $\boldsymbol{x}_4^{(1)} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$, with a mean of $\boldsymbol{\mu}_1 = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$. The second cluster contains the points $\boldsymbol{x}_1^{(2)} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$, $\boldsymbol{x}_2^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $\boldsymbol{x}_3^{(2)} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$ and $\boldsymbol{x}_4^{(2)} = \begin{bmatrix} 5 \\ -2 \end{bmatrix}$, with a mean of $\boldsymbol{\mu}_2 = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$. The covariance matrices for the clusters are therefore given by

$$\Sigma_1 = \frac{1}{4}\begin{bmatrix} 0 & -1 & 0 & 1 \\ -2 & 0 & 2 & 0 \end{bmatrix}\begin{bmatrix} 0 & -2 \\ -1 & 0 \\ 0 & 2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix}$$

and

$$\Sigma_2 = \frac{1}{4}\begin{bmatrix} 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \end{bmatrix}\begin{bmatrix} 0 & 2 \\ -2 & 0 \\ 0 & -2 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$

where we have subtracted the mean from each point. The inverses of the two covariance matrices are given by $\Sigma_1^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$, and $\Sigma_2^{-1} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$, and the determinants by $|\Sigma_1| = 1$, and $|\Sigma_2| = 4$. Computing W and $\boldsymbol{w}$ using equation (4.11) gives us $W = \begin{bmatrix} 0.75 & 0 \\ 0 & 0 \end{bmatrix}$, and $\boldsymbol{w} = \begin{bmatrix} -4.5 \\ -1.5 \end{bmatrix}$. To compute the w_{0k} terms, we need to know the probabilities of clusters 1 and 2. Assigning equal probabilities ($P(C_1) = 0.5$ and $P(C_2) = 0.5$) means that the probabilities will cancel in w_0, leaving

$$w_0 = w_{0(2)} - w_{0(1)} = -\frac{1}{2}\left(\begin{bmatrix} 3 & -2 \end{bmatrix}\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} 3 \\ -2 \end{bmatrix} + \ln(4) \right) + \frac{1}{2}\left(\begin{bmatrix} 3 & 6 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} 3 \\ 6 \end{bmatrix} + \ln(1) \right)$$
$$= 14.057$$

To find the equation for the line separating the two clusters note that both w_{12} and w_{22} are equal to zero, so that equation (4.11) simplifies to

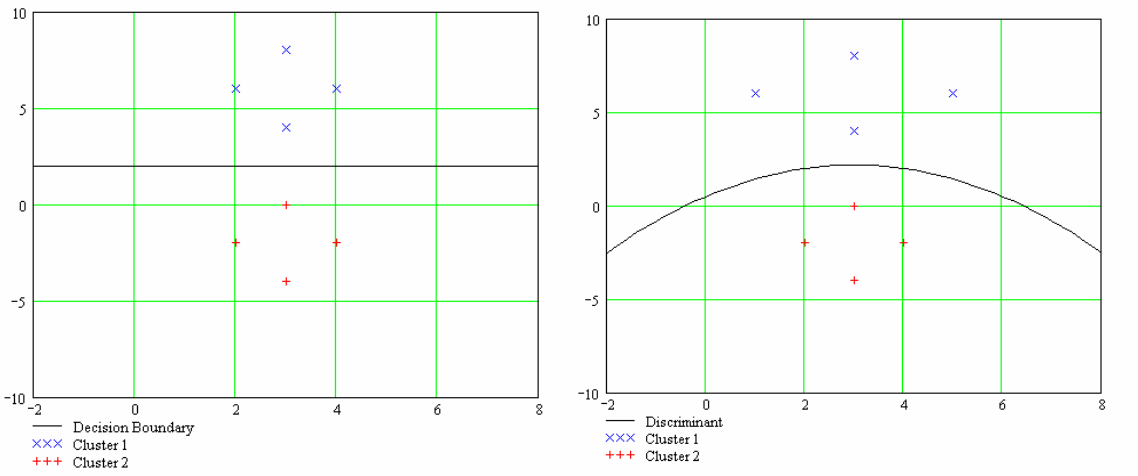
$$x_2 = -(w_{11}/w_2)x_1^2 - (w_1/w_2)x_1 - (w_0/w_2),$$

which, for the case we have just considered, gives us

$$x_2 = 0.1875x_1^2 - 1.125x_1 + 3.514,$$

an upward trending parabola with vertex equal to *[3, 1.83]*. This parabola, which is the Bayes decision boundary for this case, is shown in Figure 4.1(b). The key thing to note about this decision boundary is that it curls towards the cluster with the more compact shape. Mathematically, this is telling us that the covariance matrix of cluster 1 contains a smaller value for the auto-covariance in the x direction, which was clear for the computations.

Two further examples are shown in Figure 4.2, which were not shown by Duda et al. (2001). In the first example, shown in Figure 4.2(a), the covariances of the two clusters are equal, as shown by their identical shapes. As indicated in equation (4.10), the discriminant function for Figure 4.2(a) is linear. In the second example, shown in Figure 4.2(b), the covariance of cluster one is now is larger in the *x* direction than the covariance of cluster 2. Thus, the discriminant function is now a downward trending parabola.



(a) (b)

Figure 4.2: Two further classification examples, where (a) shows two clusters with equal variance, and (b) shows two clusters where cluster 1 has larger variance than cluster 2.

4.3 Linear discriminant functions

As shown by equation (4.10), a linear discriminant function between two clusters of points is the simplification of the general Bayesian formula in which the covariance matrices of the two clusters are equal. The linear discriminant function is given by the formula:

$$f(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T \boldsymbol{x}\,, \tag{4.12}$$

where $\boldsymbol{x}^T = [x_1 \quad x_2 \quad \cdots \quad x_M]$ is an *M*-dimensional input vector of seismic attributes, $\boldsymbol{w}^T = [w_1 \quad w_2 \quad \cdots \quad w_M]$ is an *M*-dimensional weight vector, and w_0 is a bias or threshold weight. Notice that equation (4.12) appears identical to equation (3.49) in the previous chapter, except that the threshold weight was absorbed into $\boldsymbol{w}$ in that equation

by setting the first value in $\boldsymbol{x}$ to one. But there is another important difference between the two equations. In equation (3.49), the dependent variable was the target value *t*, which was a measured reservoir parameter. This leads to the linear regression equation, in which we solve for the weights by using *N* separate equations. In equation (4.1), $f(\boldsymbol{x})$ is not a measured value but is instead used to classify $\boldsymbol{x}$ into one of *K* classes. In the simplest case where $K = 2$, this is done by assigning a value of 0 or 1 (or -1 and +1) to each of the $\boldsymbol{x}_j$ vectors, and solving for the weights that best separate the sets of values.

4.4 The Fisher linear discriminant

4.4.1 Theory of the Fisher linear discriminant

If we rewrite equation (4.12) without the bias term, we get

$$y = \boldsymbol{w}^T \boldsymbol{x}, \tag{4.13}$$

where $\boldsymbol{x}^T = [x_1, \quad x_2, \quad \cdots, \quad x_M]$ is an *M*-dimensional input sample vector of seismic attributes, and $\boldsymbol{w}^T = [w_1, \quad w_2, \quad \cdots, \quad w_M]$ is an *M*-dimensional weight vector. The interpretation of equation (4.2) is that we have projected the *M*-dimensional vector $\boldsymbol{x}$ onto the one-dimensional space of *y*. This is called "dimensionality reduction", and for multiple values of $\boldsymbol{x}$ the set of output values *y* creates a line called the discriminant line. If we divide these points into multiple clusters, the objective of discriminant analysis is to find the weight vector that maximizes the separation of these clusters along the line defined by the output values *y*.

The simplest case is that of two clusters, in which case we can assume that the *N* values of $\boldsymbol{x}$ are divided into two clusters, C_1 and C_2, containing N_1 and N_2 values, respectively. We can then define the means for the clusters as

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{j \in C_k} \boldsymbol{x}_j, \tag{4.14}$$

As a first attempt at finding the weight vector, we could try to maximize the distance between the projected means, or

$$m_2 - m_1 = \boldsymbol{w}^T(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1). \tag{4.15}$$

However, as discussed by both Bishop (1995) and Duda et al. (2001), this approach does not take into account the scatter of the clusters and therefore does not provide a very good way of discriminating between clusters. Fisher (1936) proposed that the problem could be solved by maximizing the difference between the means divided by the sum of the within-class scatter, or

$$J(\boldsymbol{w}) = \frac{(m_2 - m_1)^2}{s_2^2 + s_1^2}, \tag{4.16}$$

where $s_k^2 = \sum_{j \in C_k} (y_j - m_k)^2$ defines the within-class scatter.

As shown by Bishop (1995), equation (4.16) can be re-written in matrix form as

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}, \tag{4.17}$$

where $\boldsymbol{S_B}$ is the between-class covariance matrix defined by

$$\boldsymbol{S_B} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T, \tag{4.18}$$

and $\boldsymbol{S_W}$ is the within-class covariance matrix defined by

$$\boldsymbol{S_W} = \sum_{j \in C_1} (\boldsymbol{x}_j - \boldsymbol{\mu}_1)(\boldsymbol{x}_j - \boldsymbol{\mu}_1)^T + \sum_{j \in C_2} (\boldsymbol{x}_j - \boldsymbol{\mu}_2)(\boldsymbol{x}_j - \boldsymbol{\mu}_2)^T. \tag{4.19}$$

As also shown by Bishop (1995), equation (4.17) is maximized when

$$\boldsymbol{w} = \boldsymbol{S_W^{-1}}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1). \tag{4.20}$$

Equation (4.20) is known as Fisher's linear discriminant. As pointed out by Duda et al. (2001), this is almost exactly the same formulation that we found using Bayes' theorem for clusters with identical covariance matrices, or

$$\boldsymbol{w} = \boldsymbol{\Sigma^{-1}}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1). \tag{4.21}$$

This is to be expected since the covariance matrix and the scatter matrix are simply scaled versions of each other.

To generalize the above formulation for multiple clusters, we must maximize the equation

$$J(\boldsymbol{W}) = \frac{\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}}{\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}}, \tag{4.22}$$

where $\boldsymbol{W}$ is the matrix containing the weight vectors for each cluster $\boldsymbol{w_k}$, $k = 1, \ldots, K$. This is a more difficult problem than for the $K = 2$ case, but W can be maximized by selecting the columns of $\boldsymbol{W}$ as the generalized eigenvectors corresponding the largest eigenvectors in

$$\boldsymbol{S}_B \boldsymbol{w}_k = \lambda_k \boldsymbol{S}_W \boldsymbol{w}_k . \tag{4.23}$$

4.4.2 Applying the Fisher linear discriminant to porosity classification

Let us now apply the Fisher linear discriminant to a case study that involves the classification of density-derived porosity in a sandstone reservoir. The study area shown in Figure 4.3 is the Blackfoot oilfield in Alberta, the same as for the case study considered in section 3.7.

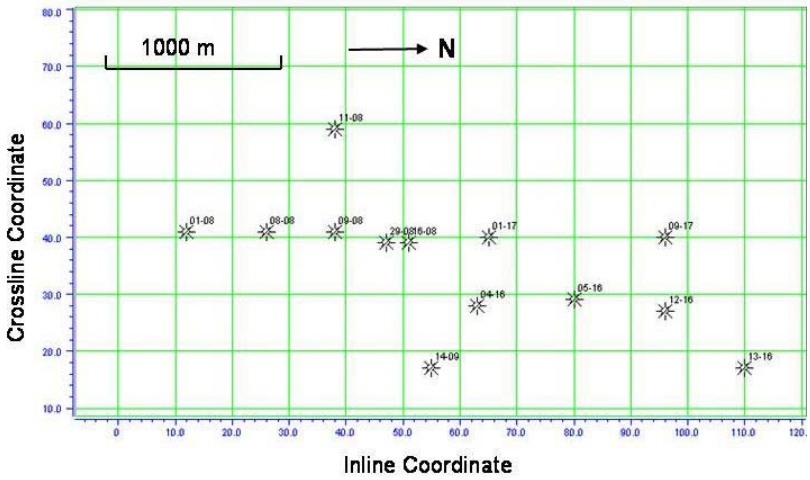


Figure 4.3: The distribution of wells used in this study. This map is from the Blackfoot oilfield of Alberta.

Figure 4.4 shows well 08-08 from the map in Figure 4.3. On this well are shown the check-shot-corrected sonic log, the density log, the porosity log, which has been

created from the density log, and the classified porosity logs. The porosity logs were classified using the following relationship:

$$C = \begin{cases} 1 \text{ if } \phi < 5\% \\ 2 \text{ if } 5\% \le \phi < 15\% \\ 3 \text{ if } \phi \ge 15\% \end{cases},$$

where ϕ is the porosity value. That is, I have labelled three classes of porosity: low, medium and high. Although it is hard to interpret the classified log at the scale shown in Figure 4.4, notice the high porosity zone at about 1570 m in well 08-08.

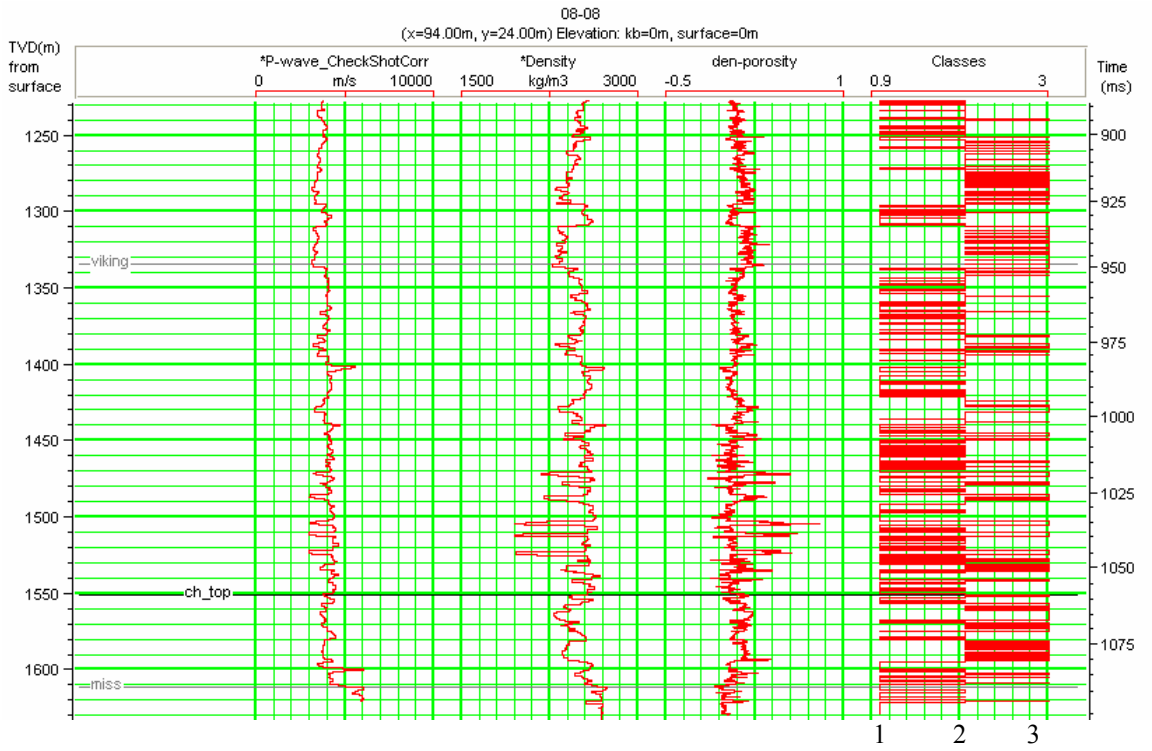


Figure 4.4: The well log curves from well 08-08, whose location is shown in Figure 4.3.

Figure 4.5 then shows line 95 from the 3D volume, with the integrated sonic log from well 08-08 superimposed on the section. The portion of the seismic line shown in Figure 4.5 is shown on the map in Figure 4.3 as a red line. The wiggle traces show the input seismic traces. The coloured amplitudes are the impedance values from a model-based inversion. The theory of model-based inversion was described in section 2.8.2. The colour bar for the inverted values is shown on the left of the section.

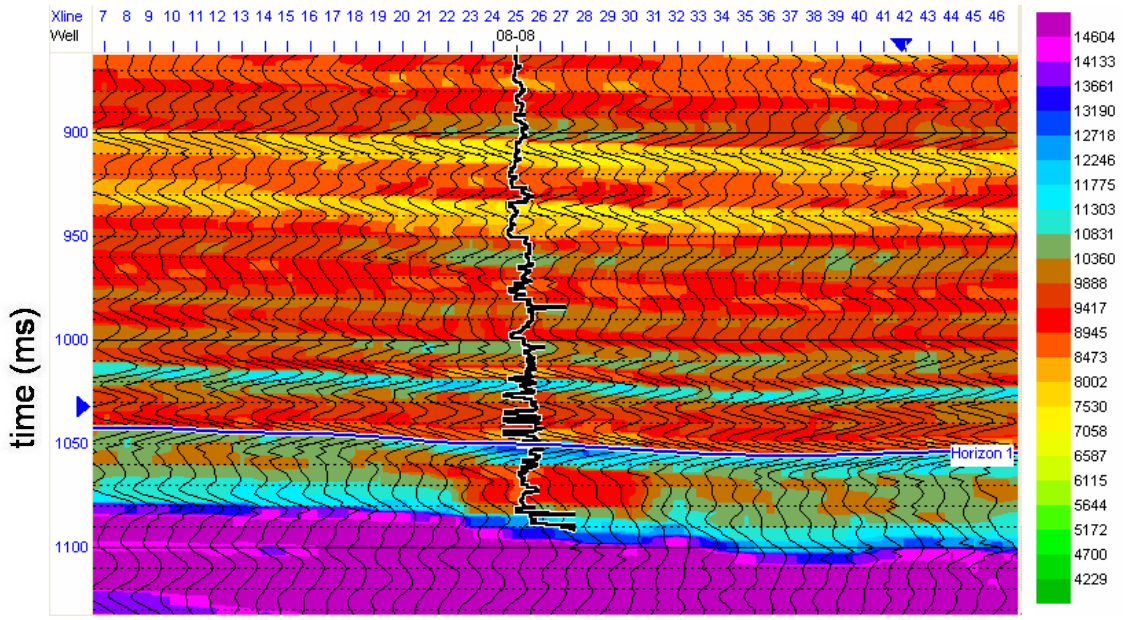


Figure 4.5: Line 95 from the 3D seismic volume shown in Figure 4.3.

In order to train the linear classification scheme, we first need to extract the composite seismic traces and impedance traces at the well locations. The result of this extraction is shown in Figure 4.6 for three of the well logs, including well 08-08.

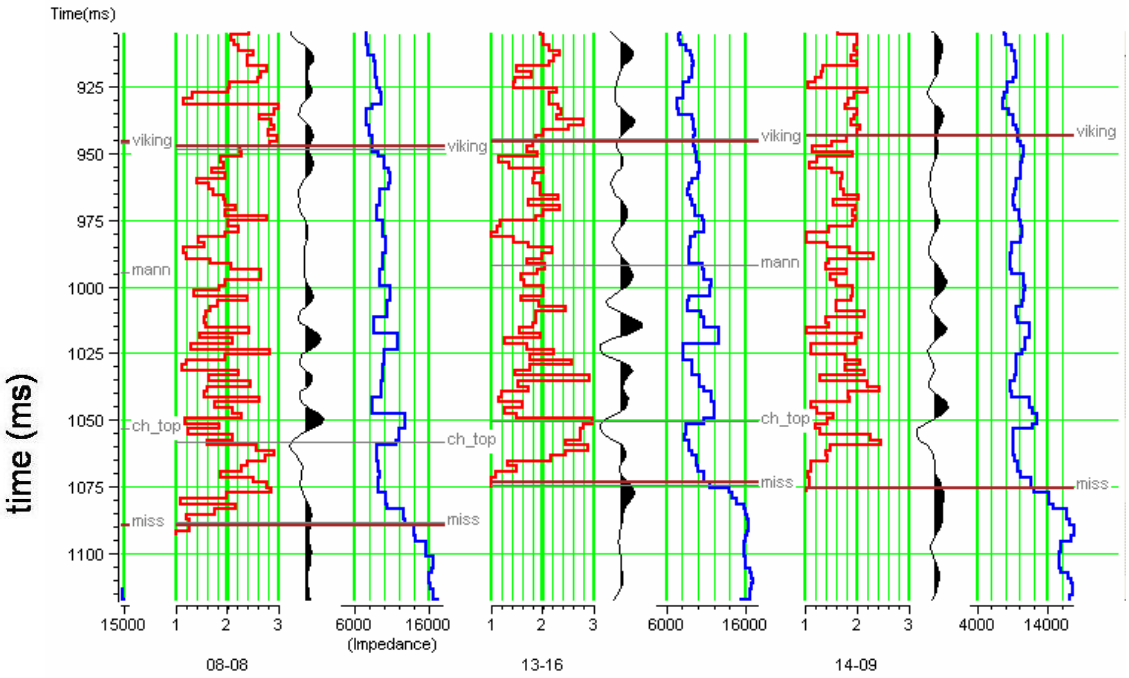


Figure 4.6: For three of the wells, the classified porosity log is shown on the left, the extracted seismic trace in the middle, and the extracted inverted trace on the left. The analysis zone is shown by the horizontal lines.

We next need to decide on which attributes to use for the classification of porosity using the seismic data, and will use the same procedure as described in detail in section 3.6. That is, we will find the best attributes using step-wise regression and decide which attributes to keep using cross-validation. Table 3.1 shows both the chosen attributes and the training and validation error. Only the first four attributes will be used, since the validation error starts to increase after that point.

.	Target	Final Attribute	Training Error	Validation Error
1	Classes	1 / (Impedance)	0.430196	0.442690
2	Classes	Filter 15/20-25/30	0.409517	0.426141
3	Classes	Amplitude Envelope	0.396259	0.418220
4	Classes	Filter 35/40-45/50	0.388704	0.415778
5	Classes	Raw Seismic	0.381104	0.417000
6	Classes	Cosine Instantaneous Phase	0.375897	0.414686
7	Classes	Integrated Absolute Amplitude	0.369723	0.416103

Table 4.1: The attributes used in the porosity classification, with their training and validation error.

Once the training has been completed, it can be applied to the full seismic volume using the attributes shown in Table 4.1. This fit at the wells is shown in Figure 4.7 for the three wells shown in Figure 4.6, where (a) shows the training result and (b) shows the validation result. The training result shows the effect of computing the weighting coefficients from all the wells and applying them to all the wells, whereas the validation result shows the effect of leaving the well shown from the computation and is thus a "blind" prediction of this result.

In Figure 4.7 the RMS error is shown at the top of each result. Note that the error is 0.3659 for the training result and 0.4657 for the classification result. This is as expected, since the validation result will always have a larger error.

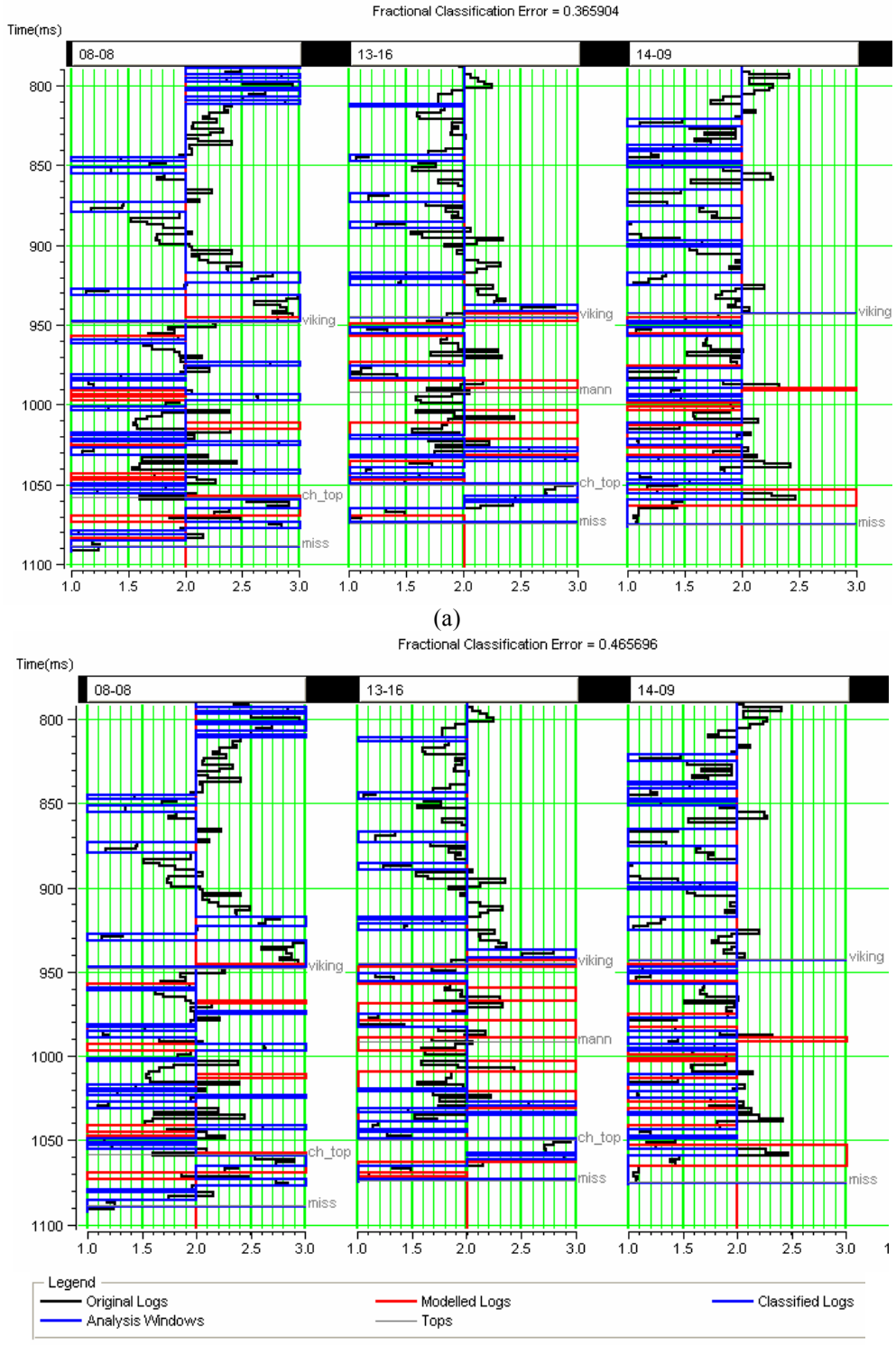


Figure 4.7: The application of the classification procedure to the three classified logs shown in Figure 4.6, where (a) shows the training result, and (b) shows the validation result.

Finally, I will apply the Fisher linear classification scheme to the seismic data. The result over the seismic line from Figure 4.5 is shown in Figure 4.8, where grey shows the low-porosity values, yellow shows the medium-porosity values and blue shows the high-porosity values. Notice the lateral definition of the high-porosity channel just below 1050 ms. This shows continuity away from the well-log-derived porosity. The inserted curve from well 08-08 is the density-porosity log, not the classified log. This log has been integrated to time, so lacks the detail shown on the depth-sampled log shown in Figure 4.6. It should also be pointed out that the apparent high-porosity zones in the shallow section between 860 and 950 ms are due to shale streaks rather than highly porous sands.

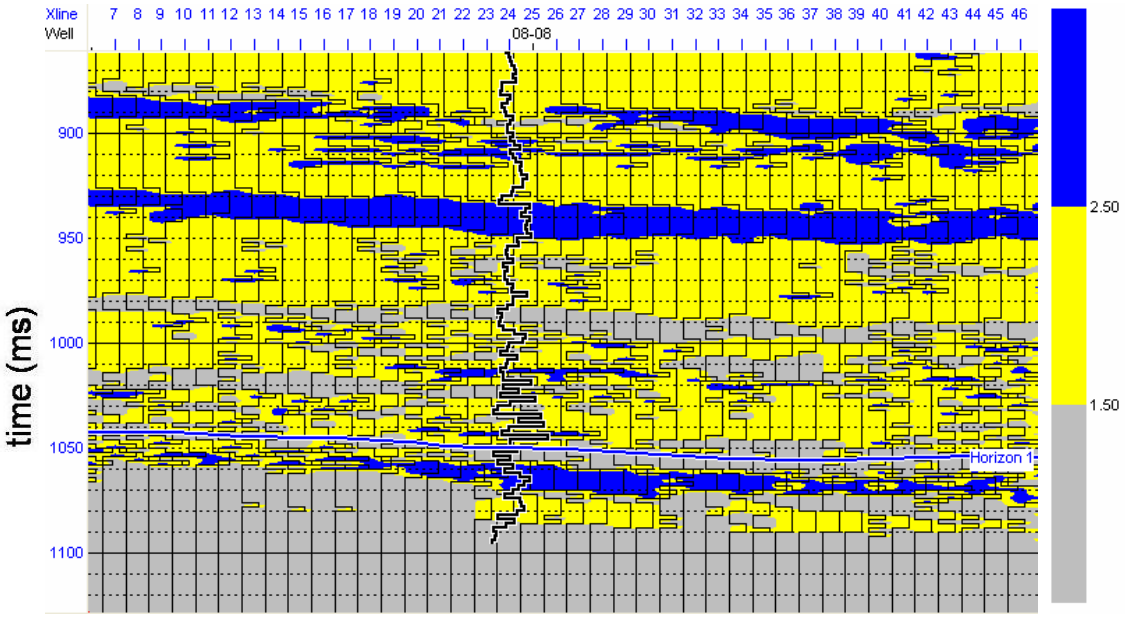


Figure 4.8: The classified porosity values on line 95 from Figure 4.5.

Figure 4.9 shows a slice of the classified porosity taken from a single sample window that was extracted 20 ms below the picked event that was labelled Horizon 1 in Figure 4.8. The wells have also been indicated in Figure 4.9. Notice that the continuity of the porosity is quite good in a lateral sense away from well 08-08. However, there are false indications of high porosity across the upper and lower left parts of the map. This is

partly due to the fact we are using a linear classification scheme on a nonlinear problem. As we shall see in Chapter 6, this can be improved with the probabilistic neural network.

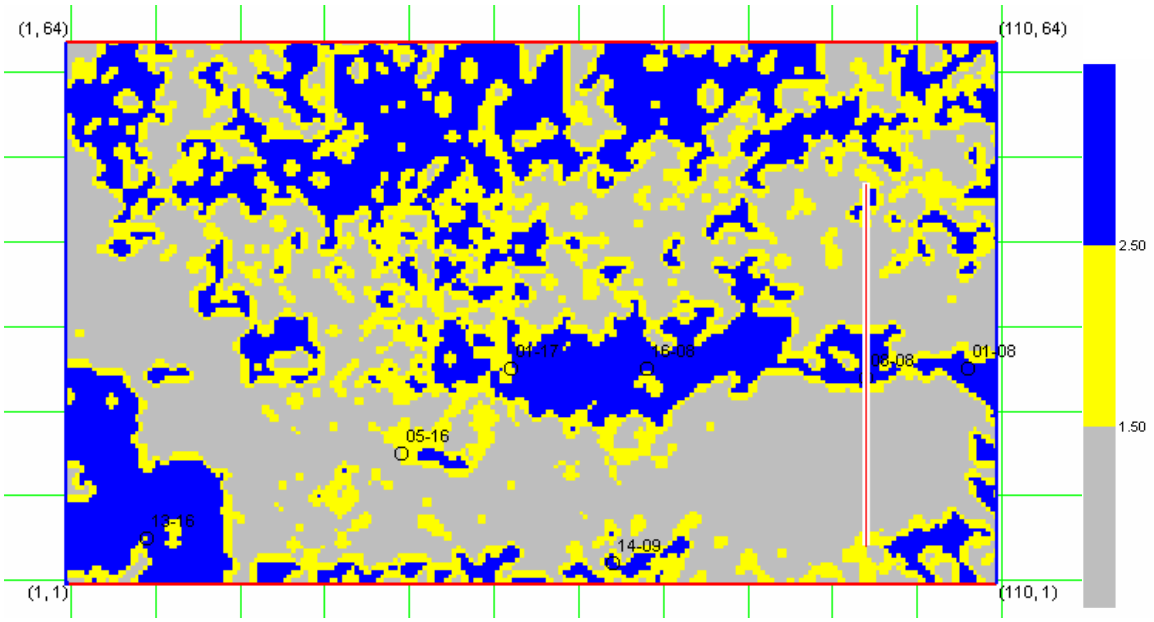


Figure 4.9: The classified porosity shown over a slice that was picked 20 ms below Horizon 1, shown on the section in Figure 4.8.

Now that I have discussed the Fisher linear discriminant function, which is essentially a tool developed by statisticians, I will look at the related concept of the single-layer perceptron, which started the new field of neural networks. Through this discussion, it will become obvious how closely related are the fields of statistics and neural networks.

4.5 The single-layer perceptron

The single-layer perceptron is a mathematical concept that is closely related to the linear discriminant function. It will also lay the basis for the multi-layer perceptron that will be discussed in the next chapter. In this section I will discuss the theory of the single-layer perceptron and then apply the theory to a geophysical example.

4.5.1 Basic theory of the single-layer perceptron

The classic model of the neuron is called the perceptron (McCulloch and Pitts, 1943) and is illustrated in Figure 4.10.

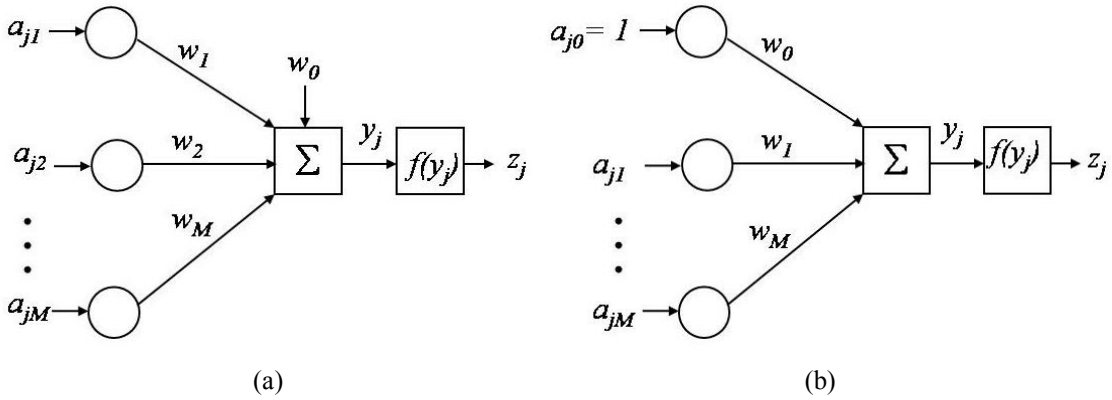


Figure 4.10: The figure above shows the perceptron neural network for (a) *M* inputs and a single output, with a bias weight fed directly into the summation, and (b) *M+1* inputs and a single output, where the bias weight is applied to a zeroth attribute which is equal to the unity vector **1**.

In Figure 4.10, notice that there are two ways to interpret the perceptron. In the first [Fig. 4.10(a)], the perceptron accepts *M* inputs $a_{j1}, a_{j2}, \ldots, a_{jM}$, and produces a single output. The inputs are then weighted and summed according to the equation:

$$y = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_M a_M , \tag{4.24}$$

where the first weight, w_0, is called the bias. Next, a threshold function, *f*, is applied to the intermediate output *x* to produce the final output *y*, or

$$z = f(y) \tag{4.25}$$

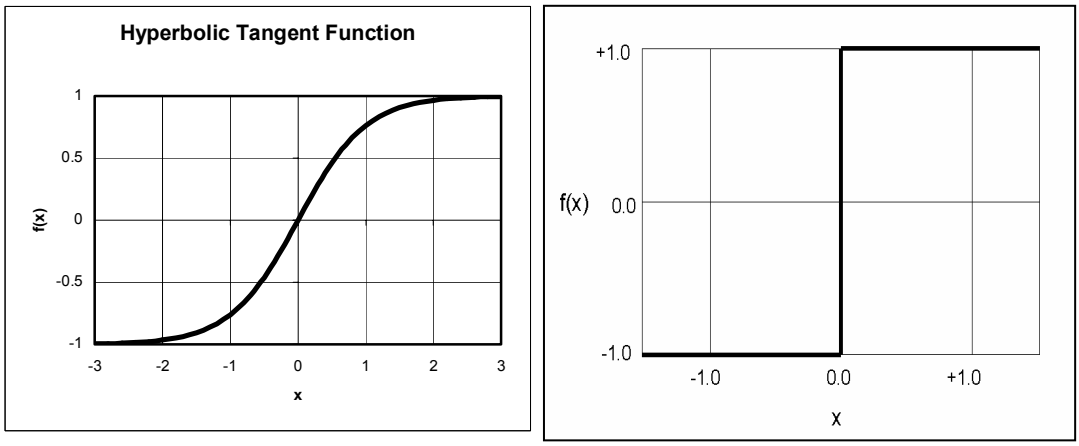
In the second interpretation (Figure 4.10(b)), the perceptron accepts *M+1* inputs $a_{j0}, a_{j1}, \ldots, a_{jM}$, where the a_{j0} term is equal to 1. Thus, the bias term now acts on the first attribute, and we can write

$$y = w_0 a_0 + w_1 a_1 + w_2 a_2 + \ldots + w_M a_M \tag{4.26}$$

In the Figure 4.10(b), the output of the summation is also transformed by a threshold function of equation (4.25). The choice of the threshold function *f* is important and depends on the problem being solved. If *f(y) = y*, the perceptron reduces to a linear sum of the inputs. This function is used in the linear associator, discussed below. In many

applications, *f(y)* is set to the smoothly varying sigmoidal function such as the hyperbolic tangent function [Fig. 4.11(a)], which is defined as:

$$f(y)=\frac{e^{y}-e^{-y}}{e^{y}+e^{-y}}. \tag{4.27}$$



(a) (b)

Figure 4.11: This figure shows a graph of (a) the hyperbolic tangent function of Equation (4.27), and (b) the symmetric step function of Equation (4.28).

For a two-class problem, such as the one we are discussing here, we often use the step function [Fig. 4.11(b)], which is given mathematically by the equation:

$$f(y)=\begin{cases}+1, & y\geq 0\\ -1, & y<0\end{cases} \tag{4.28}$$

4.5.2 An AVO classification problem

The basic AVO interpretation problem that I will study (Russell et al., 2002) is how to differentiate between the AVO responses of the two reservoirs shown in Figure 4.12. Figure 4.12(a) shows a wet sand encased between two shale layers, and Figure 4.12(b) shows a gas sand encased between the same two shales. The P-wave velocity (V_P), S-wave velocity (V_S), and density (ρ) for each layer are shown in each figure. I will assume that the far angle of incidence is small enough (i.e. approximately 30°) that we

can ignore the third term in the Aki-Richards equation and write the reflectivity as a function of angle of incidence θ as

$$R(\theta) = A + B\sin^2\theta, \tag{4.29}$$

where A is the intercept given by $A = \frac{1}{2}\left[\frac{\Delta V_P}{V_p} + \frac{\Delta\rho}{\rho}\right]$, and B is the gradient given by $B = \frac{1}{2}\frac{\Delta V_P}{V_p} - 4\frac{V_S^2}{V_P^2}\frac{\Delta V_S}{V_S} - 2\frac{V_S^2}{V_P^2}\frac{\Delta\rho}{\rho}$. The terms ΔV_P, ΔV_P, and $\Delta\rho$ are the differences across the layers and the terms V_P, V_P, and ρ are the averages.

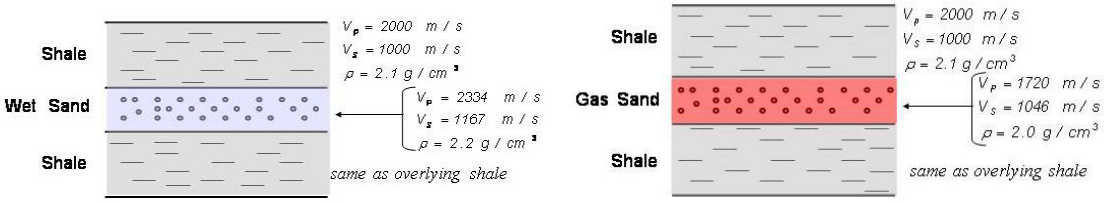


Figure 4.12: Two simple geological models where (a) shows a wet sand between two shale layers and (b) shows a gas sand between the same two shales.

Using the values for V_P, V_S, and ρ shown in Figure 4.12, we can now compute the values for the AVO intercept and gradient for the wet and gas sands. For the wet sand, note that V_P/V_S in both the sand and shale layers is equal to 2. This means that $\frac{\Delta V_P}{V_p} = \frac{\Delta V_S}{V_S}$ and, thus substituting this value into the expression for the gradient B given above leads to the simplification that $A = -B$ for both the top and base of the layer. (For a more complete derivation, see the Appendix to Russell et al., 2002). Using the parameters shown in Figure 4.6 gives $A_{TOP\_WET} = B_{BASE\_WET} = +0.1$ and $A_{BASE\_WET} = B_{TOP\_WET} = -0.1$. For the gas sand, the V_P/V_S ratio is equal to 1.65, and leads to $A_{TOP\_GAS} = B_{TOP\_GAS} = -0.1$ and $A_{BASE\_GAS} = B_{BASE\_GAS} = +0.1$.

Using the parameters for the gas case, we find that $A=B$ for both the top and base of the layer. The AVO curves for the wet and gas cases are shown in Figure 4.13, for an angular aperture of 0º to 30º. It is observed that the absolute values of the gas sand

curves show an increase in amplitude, whereas the absolute values of the wet sand curves show a decrease in amplitude.

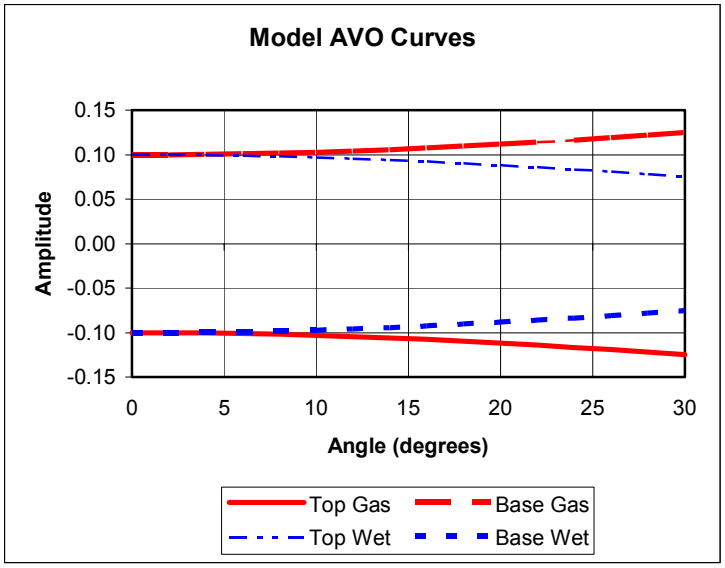
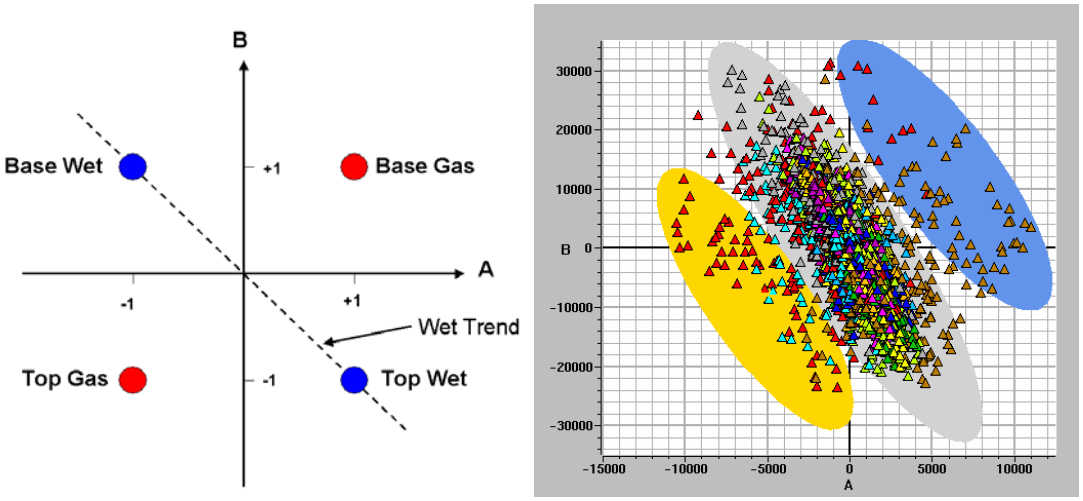


Figure 4.13: This figure shows the AVO responses from the top and base interfaces of the wet and gas sands shown in Figure 4.12.

Although the parameters used here simplify the problem, these values do fall within a reasonable petrophysical range. After scaling each of the values of A and B by a factor of 10 to give values of +1 and -1, they have been put on an A-B crossplot, as shown in Figure 4.14(a). In our example, the wet points (shown as solid blue circles) establish the wet sand-shale trend, and the top and base gas (shown as solid red circles) plot in the other two quadrants of the A-B crossplot. This is a typical class 3 AVO anomaly (Rutherford and Williams, 1989), caused by the reduction of the impedance and the V_p/V_s ratio of the sand by gas saturation.

Despite the simplicity of the models shown in Figure 4.12, the plot in Figure 4.14(a) shows us what is expected in a noise-free AVO crossplot. For comparison, Figure 4.14(b) is an interpreted AVO A-B crossplot for a class 3 AVO response in the Gulf of Mexico (Ross, 2000). The centre grey ellipse encompasses all of the wet sand-shale AVO points while the gold and blue ellipses outlying the grey "wet trend" points are associated with the top and base of the pay sand, respectively.



(a) (b)

Figure 4.14: Intercept versus gradient crossplots, where (a) shows the crossplot of the A and B values from the wet and gas models of Figure 4.12, crossplotted after being scaled by a factor of 10, and (b) shows a Gulf of Mexico real data example.

Identifying the wet trend and the outlying two points in Figure 4.14(a) is a trivial problem for the eye to interpret. Let us see if the single-layer perceptron is able to solve this problem. The application of the single-layer perceptron to our AVO problem is shown by the neural network graph in Figure 4.15. We now have two inputs, the intercept (*A*) and gradient (*B*), and will use the symmetric step function to compute the final output. For the output, a value of $+1$ will indicate the presence of a gas sand and a value of -1 will indicate the presence of a wet sand.

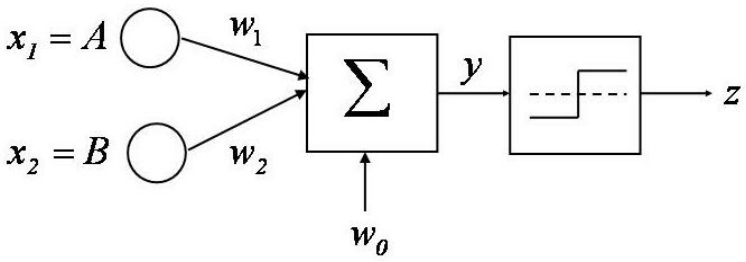


Figure 4.15: The perceptron adapted to the AVO problem of Figure 4.14(a), where the inputs are the intercept (*A*) and gradient (*B*) and the function is the symmetrical step function.

Notice that the equation for intermediate output y is now given as

$$y = w_0 + w_1 A + w_2 B \tag{4.30}$$

We now want to determine the weights w_0, w_1, and w_2. From equation (4.30), it is obvious we are interested in the separation between $y < 0$ and $y > 0$, which occurs when $y = 0$. This is called the decision boundary, and is illustrated in Figure 4.16. From this figure it is clear that the boundary crosses the A and B axes at $A = \frac{-w_0}{w_1}$ and $B = \frac{-w_0}{w_2}$.

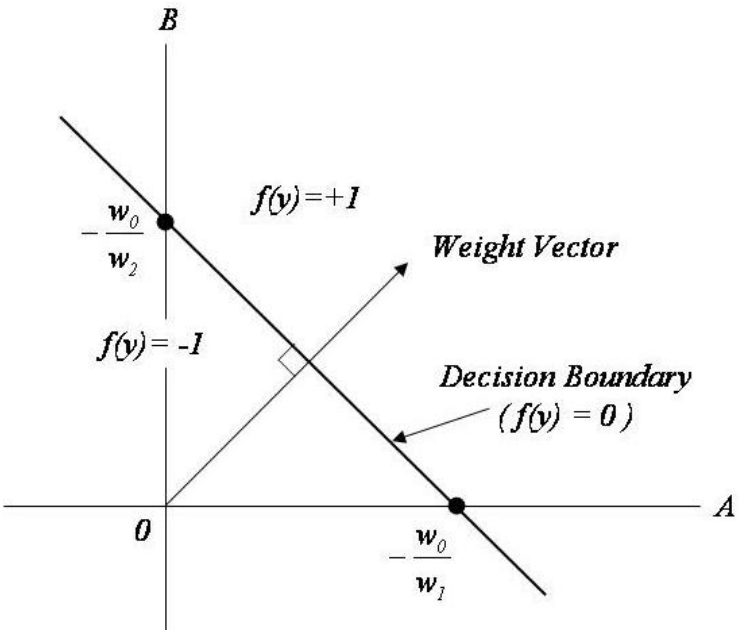
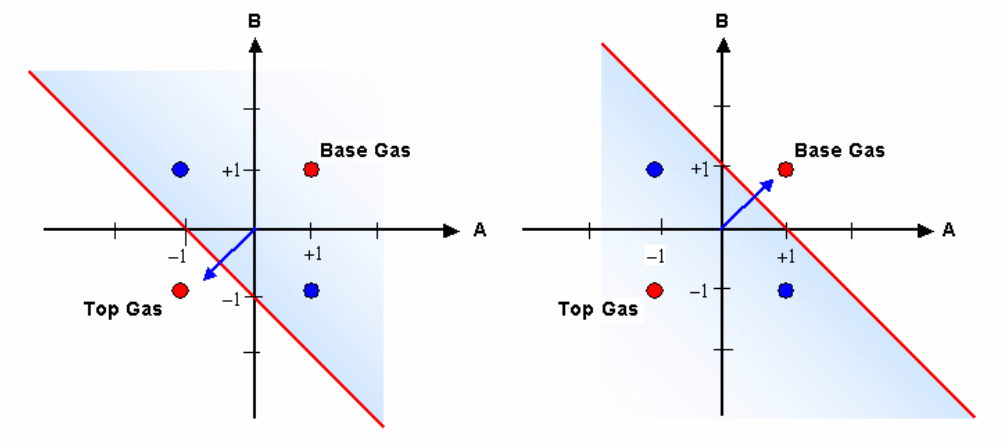


Figure 4.16: The perceptron decision boundary. Note that either w_0 or w_1 and w_2 must be negative to make the resulting intercept value on the A and B axes positive.

Figure 4.16 also shows the weight vector $\boldsymbol{w}^T = [w_1, w_2]$, which is normal to the decision boundary and points in the direction of $f(x) = +1$. This will give us the signs of w_1 and w_2. It is important to note from Figure 4.16 that the perceptron can only separate points that are *linearly* separable. That is, for the two dimensional case we can draw a line between the points, and for the three-dimensional case we can draw a plane. (For higher-dimensional inputs we use hyperplanes to separate the points). This limitation means that the perceptron can not solve a simple Boolean algebra problem, the exclusive OR, or XOR (Haykin, 1999). This problem is similar to our AVO problem.

The AVO problem of Figure 4.12(a) has been redrawn in Figure 4.17. Notice that we are not able to separate both the top and base of the gas-sand from the wet trend with a single decision boundary. We can separate either the top of the gas-sand, as shown in Figure 4.17(a), or the base of the gas-sand, as shown in Figure 4.17(b).



(a) (b)

Figure 4.17: The AVO problem from Figure 4.12(b) with decision boundaries, where (a) shows separation of the base of the gas sand and (b) shows separation of the top of the gas sand.

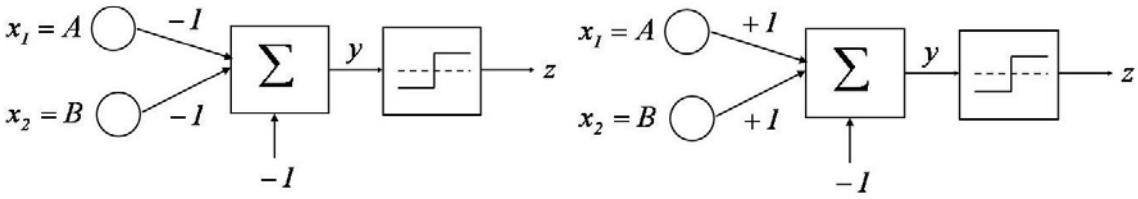
For the top of the gas-sand, we can compute the weights from

$$A = B = -\frac{w_0}{w_1} = -\frac{w_0}{w_2} = -1 .$$

Although w_0, w_1 and w_2 can be scaled by any value, we usually choose the simplest values of $w_1 = w_2 = -1$, and therefore $w_0 = -1$. The perceptron diagram for this is shown in Figure 4.18(a). The weights for the base of gas sand can be computed as

$$A = B = -\frac{w_0}{w_1} = -\frac{w_0}{w_2} = +1 ,$$

so that the weights are $w_1 = w_2 = +1$ and $w_0 = -1$. This is shown in Figure 4.18(b).



(a) (b)

Figure 4.18: Perceptron implementations for separating the (a) top of gas, and (b) base of gas.

Table 4.2 shows that these values do indeed solve the problem for the four possible cases.

	Inputs		*Perceptron 1*		*Perceptron 2*	
SAND	***A***	***B***	x_1	y_1	x_2	y_2
Top Gas	*-1*	*-1*	*+1*	*+1*	*-3*	*-1*
Base Wet	*-1*	*+1*	*-1*	*-1*	*-1*	*-1*
Top Wet	*+1*	*-1*	*-1*	*-1*	*-1*	*-1*
Base Gas	*+1*	*+1*	*-3*	*-1*	*+1*	*+1*

Table 4.2: The outputs from the perceptron models of Figure 4.18.

Although we have solved for the top and bottom of the gas-sand individually we have still not solved the complete problem, which is to separate the gas-sand responses from the wet-sand responses. This requires a multi-layer perceptron, or MLP, which will be discussed in the next chapter.

4.6 Computing the neural network weights

4.6.1 The perceptron learning rule

In the last section, I applied the perceptron model to an AVO classification problem and used intuition to solve for the weights. McCulloch and Pitts (1943) devised the first analytical approach to solve for the weights and called it the perceptron learning rule. I will apply this rule to the AVO example of the previous section.

Figure 4.19 shows an illustration of the perceptron neural network as applied to our *A-B* crossplot example. In Figure 4.19, and in the following theory, note that we are conforming to the notation set out in Appendix 1. That is, we have M attribute vectors, $\boldsymbol{a}_i$, and N input sample vectors $\boldsymbol{x}_j$. The perceptron leaning rule can be stated as follows. Given N input/training pairs $\{\boldsymbol{x}_j, t_j\}$, present these pairs sequentially to the algorithm and modify the weights so as to reduce the error between the actual output, z_j, and the target value, t_j.

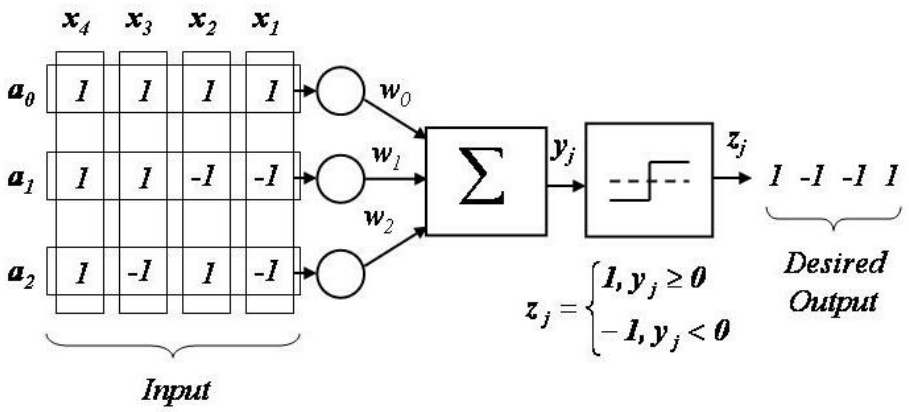


Figure 4.19: A conceptual illustration of the neural network used to solve for the A-B crossplot example.

The perceptron learning rule can written mathematically as follows

$$\begin{aligned} \boldsymbol{w}_{new} &= \boldsymbol{w}_{old} + e\boldsymbol{x}_j, \\ w_{0new} &= w_{0old} + e, \end{aligned} \tag{4.31}$$

where $\boldsymbol{x_j}$ is the input sample vector, $e = (t_j - z_j)/M$, and M is the number of input attributes. In the perceptron model, the output z_i is computed as follows:

$$z_j = \begin{cases} 1, & y_j \geq 0, \\ -1, & y_j < 0, \end{cases}$$

where $y_j = \boldsymbol{w}^T \boldsymbol{x}_j + w_0$.

The algorithm is initialized by setting the weights to small random values. Let us see how this algorithm will work in finding the gas zones from the previous example. Starting with the base of the gas zone, our three input vectors and targets are

$$\boldsymbol{x}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ t_1 = 1,\ \boldsymbol{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix},\ t_2 = -1, \text{ and } \boldsymbol{x}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ t_3 = -1.$$

If we initialize $\boldsymbol{w}$ and w_0 as $\boldsymbol{w}_{(0)} = \begin{bmatrix} -0.1 \\ 0.1 \end{bmatrix}$, and $w_{0(0)} = 0$, the computations proceed as follows:

$$(1)\ \text{Iteration}\ \ 1: y_1 = \begin{bmatrix} -0.1 & 0.1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 0 = 0 \Rightarrow z_1 = 1 \Rightarrow e = \frac{t_1 - z_1}{2} = 0$$

$$\Rightarrow \boldsymbol{w}_{(1)} = \boldsymbol{w}_{(0)} + 0 = \begin{bmatrix} -0.1 \\ 0.1 \end{bmatrix}, w_{0(1)} = 0.$$

Applying these weights, we find that $z_1 = 1 = t_1$, $z_2 = 1 \neq t_2$, and $z_3 = -1 = t_3$.

$$(2)\ \text{Iteration}\ 2: y_2 = \begin{bmatrix} -0.1 & 0.1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0 = 0.2 \Rightarrow z_2 = 1 \Rightarrow e = \frac{t_1 - z_1}{2} = -1$$

$$\Rightarrow \boldsymbol{w}_{(2)} = \boldsymbol{w}_{(1)} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.9 \\ -0.9 \end{bmatrix}, w_{0(2)} = -1.$$

Applying these weights, we find that $z_1 = -1 \neq t_1$, $z_2 = -1 = t_2$, and $z_3 = 1 \neq t_3$.

$$(3)\ \text{Iteration}\ 3: \ y_1 = \begin{bmatrix} 0.9 & -0.9 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} - 1 = 0.8 \Rightarrow z_1 = 1 \Rightarrow e = \frac{t_1 - z_1}{2} = -1$$

$$\Rightarrow \boldsymbol{w}_{(3)} = \boldsymbol{w}_{(2)} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.1 \end{bmatrix}, w_{0(3)} = -1.$$

Applying these weights, we find that $z_1 = -1 \neq t_1$, $z_2 = -1 = t_2$, and $z_3 = -1 = t_3$.

$$(4)\ \text{Iteration}\ 4: y_1 = \begin{bmatrix} -0.1 & 0.1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} - 1 = 0.8 \Rightarrow z_1 = 1 \Rightarrow e = \frac{t_1 - z_1}{2} = -1$$

$$\Rightarrow \boldsymbol{w}_{(4)} = \boldsymbol{w}_{(3)} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1.1 \\ -0.9 \end{bmatrix}, w_{0(4)} = 1.$$

Applying these weights, we find that $z_1 = 1 = t_1$, $z_2 = 1 \neq t_2$, and $z_3 = 1 \neq t_3$.

$$(5)\ \text{Iteration}\ 5: \ y_2 = \begin{bmatrix} -1.1 & -0.9 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} - 1 = 1.2 \Rightarrow z_2 = 1 \Rightarrow e = \frac{t_2 - z_2}{2} = -1$$

$$\Rightarrow \boldsymbol{w}_{(5)} = \boldsymbol{w}_{(4)} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -1.9 \end{bmatrix}, w_{0(5)} = -1.$$

Applying these weights, we find that $z_1 = 1 = t_1$, $z_2 = -1 = t_2$, and $z_3 = 1 \neq t_3$.

$$(6)\ \text{Iteration}\ 6: y_3 = \begin{bmatrix} -0.1 & -1.9 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} - 1 = 0.8 \Rightarrow z_3 = 1 \Rightarrow e = \frac{t_1 - z_1}{2} = -1$$

$$\Rightarrow \boldsymbol{w}_{(6)} = \boldsymbol{w}_{(5)} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1.1 \\ -0.9 \end{bmatrix}, w_{0(6)} = -1.$$

Applying these last weights, we find that $z_1 = 1 = t_1$, $z_2 = -1 = t_2$, and $z_3 = -1 = t_3$, and now all three outputs are correct. Since the correct weights are both equal to -1, the first weight has been overestimated by -0.1 and the second weight underestimated by 0.1. The bias term is equal to *-1*, the correct value. A 3D scatter plot of the convergence of the points after each iteration is shown in Figure 4.20. Note that the points move around quite randomly in 3D space until convergence is obtained.

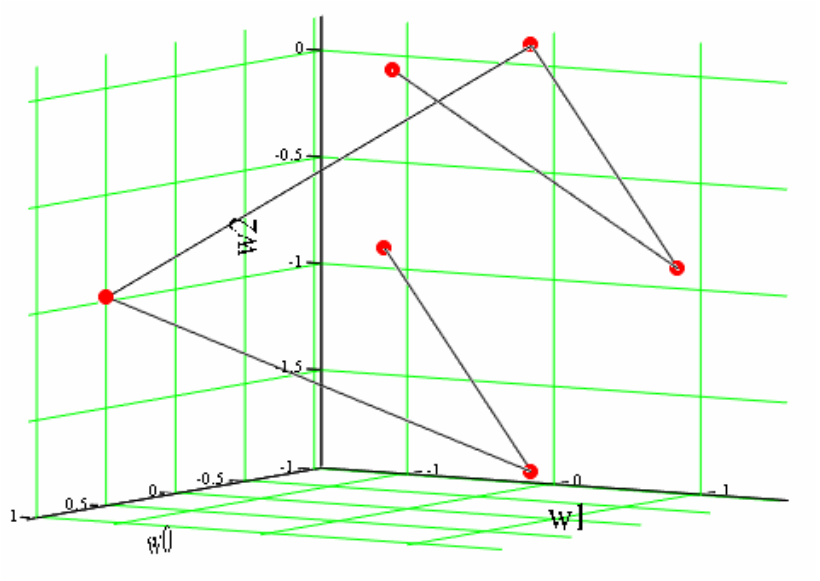


Figure 4.20: A 3D scatter plot of the perceptron weights after each iteration.

We next proceed to the problem of finding the weights for the separation of the wet sand from the base of the gas layer, where the inputs and target values are

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ t_1 = 1,\ \boldsymbol{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix},\ t_2 = -1, \text{ and } \boldsymbol{x}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ t_3 = -1,$$

and we find that, given the same starting guess, the solution again converges to

$$\boldsymbol{w}_{(6)} = \begin{bmatrix} 0.9 \\ 1.1 \end{bmatrix},\ w_{0(6)} = -1.$$

Remember that the right answer is that both weights are equal to *1* and the bias is equal to *-1*, so the algorithm has again done a good job. However, there are two major problems with the perceptron learning rule. First, since the algorithm is iterative, we do

not know in advance when it will stop, or even if it will converge to the right answer. Second, we do not really know how to initialize the weights. In the case considered above, we converged to the right answer using a starting guess of [-0.1, 0.1]. However, a starting guess of [1, 0.5] does not converge to the right answer. The problem of the initial guess in neural network weight computations will be discussed in more detail in the next chapter.

4.6.2 Hebb's rule and associative memory

Based on our conclusions about the perceptron learning rule in the last section, it would be preferable to find an algorithm that could be expressed in closed form, as was the solution to the regression problem given in the previous chapter. Such a method was introduced independently by Anderson (1972) and Kohonen (1972) and was based on the work of Hebb (1949). This method is called the linear associator, and results in a neural network called an associative memory (Haykin, 1998). A full description of this type of neural network is given in Appendix 5. In this section, I will simply apply the linear associator to the AVO classification problem that we have been studying.

The linear associator is shown in Figure 4.21. As the perceptron learning rule, the input to the linear associator is the N input/training pairs $\{\boldsymbol{x}_j, t_j\}$. However, unlike the perceptron shown in Figure 4.19, the output function is the linear function given by

$$z_j = y_j, \tag{4.32}$$

where $y_j = \boldsymbol{w}^T \boldsymbol{x}_j + w_0$. To simplify this equation, we will add the zero weight to the vector and add the value $x_0 = 1$ as the first value in each vector $\boldsymbol{x}_j$. Notice that this implies that we have added a zeroth attribute containing N ones. The output values can now be more simply written as the scalar product of two $M+1$ length vectors, or

$$y_j = \begin{bmatrix} w_0 & w_1 & \cdots & w_M \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_M \end{bmatrix} = \boldsymbol{w}^T \boldsymbol{x}_j . \tag{4.33}$$

To determine the weights given in equation (4.33), we use Hebb's rule (Appendix 5), which can be written as

$$w_{i(new)} = w_{i(old)} + t_{ij} x_{ij}\,, \tag{4.34}$$

or, in vector form, as

$$\boldsymbol{w}^T_{(new)} = \boldsymbol{w}^T_{(old)} + t_j \boldsymbol{x}^T_j\,. \tag{4.35}$$

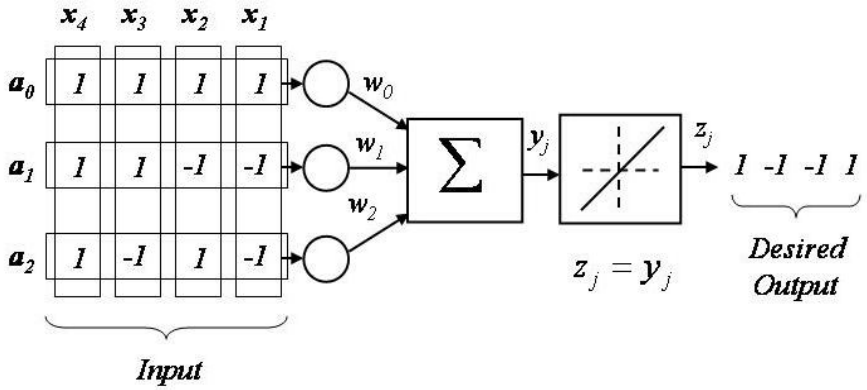


Figure 4.21: A conceptual illustration of the use of the linear associator to solve for the A-B crossplot example. The only difference with Figure 4.13 is that the applied function is linear.

If we set the initial weights to zero, equation (4.35) can be written as:

$$\boldsymbol{w}^T = \sum_j^N t_j \boldsymbol{x}^T_j = t_1 \boldsymbol{x}^T_1 + t_2 \boldsymbol{x}^T_2 + \ldots + t_N \boldsymbol{x}^T_N = \begin{bmatrix} t_1 & t_2 & \ldots & t_N \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^T_1 \\ \boldsymbol{x}^T_2 \\ \vdots \\ \boldsymbol{x}^T_N \end{bmatrix} = \boldsymbol{t}^T X^T = \boldsymbol{t}^T A\,. \tag{4.36}$$

In other words, the weight vector is given by the outer product of the vector of training values and matrix of input values. Alternately, we can write the transpose of equation (4.36), giving

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_N \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} = X\,\boldsymbol{t} = A^T \boldsymbol{t}\,. \tag{4.37}$$

As shown in Appendix 5, the associative memory works well if a copy of every possible input vector is presented to the algorithm. If some of the vectors are missing, a

better solution is obtained with the pseudo-inverse rule, which is written for equation (4.36) as

$$\boldsymbol{w}^{\boldsymbol{T}} = \boldsymbol{t}^{\boldsymbol{T}} X^{+} = \boldsymbol{t}^{\boldsymbol{T}} A^{T+}, \tag{4.38}$$

where $X^{+} = (X^{T}X + \lambda I)^{-1}X^{T}$, and $A^{T+} = (AA^{T} + \lambda I)^{-1}A$, λ is a prewhitening term and I is the M x M identity matrix. Using equation (4.37) as the starting point, we get

$$\boldsymbol{w} = X^{T+}\boldsymbol{t} = A^{+}\boldsymbol{t}. \tag{4.39}$$

where $X^{T+} = (XX^{T} + \lambda I)^{-1}X$, and $A^{+} = (A^{T}A + \lambda I)^{-1}A^{T}$.

I will now apply the preceding theory to our AVO classification problem. Let us start with the complete problem of four inputs, as shown in Figure 4.8(a), or

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix},\ t_1 = 1,\ \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix},\ t_2 = -1,\ \boldsymbol{x}_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix},\ t_3 = -1, \text{ and } \boldsymbol{x}_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},\ t_3 = 1,$$

where I have added the value 1 in each case to allow for a zeroth weight, as described above. Recall that we determined intuitively in the previous section that this problem has no solution.

From equation (4.36) the solution to the weights can be given as:

$$\boldsymbol{w}^{\boldsymbol{T}} = \begin{bmatrix} t_1 & t_2 & t_2 & t_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1^{\boldsymbol{T}} \\ \boldsymbol{x}_2^{\boldsymbol{T}} \\ \boldsymbol{x}_3^{\boldsymbol{T}} \\ \boldsymbol{x}_4^{\boldsymbol{T}} \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}. \tag{4.40}$$

The weights computed in equation (4.40) are all equal to zero because, as discussed earlier, a single layer perceptron cannot solve this problem. The input points will therefore be transformed by this weight vector and placed at the position *(0, 0)*. Using equation (4.37) we also get the same answer, or

$$\boldsymbol{w} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.41}$$

Next, we will apply the pseudo-inverse method. Using equation (4.38), we find that

$$\begin{aligned} \boldsymbol{w}^T &= \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} \left\{ \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \right\}^{-1} \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} \left\{ \begin{bmatrix} 3 & 1 & 1 & -1 \\ 1 & 3 & -1 & 1 \\ 1 & -1 & 3 & 1 \\ -1 & 1 & 1 & 3 \end{bmatrix} \right\}^{-1} \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \end{aligned} \tag{4.42}$$

However, the 4 x 4 matrix shown in equation (4.42) has a zero determinant, so has no inverse. Adding a small amount of prewhitening of *0.1*, we find

$$\boldsymbol{w}^T = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} \left\{ \begin{bmatrix} 3.1 & 1 & 1 & -1 \\ 1 & 3.1 & -1 & 1 \\ 1 & -1 & 3.1 & 1 \\ 1 & 1 & 1 & 3.1 \end{bmatrix} \right\}^{-1} \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}. \tag{4.43}$$

To solve equation (4.38), we needed to add pre-whitening, since the problem was under-constrained. Using equation (4.39), we find that:

$$\boldsymbol{w} = \left\{ \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \right\}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

$$= \left\{ \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} \right\}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.44}$$

Thus, equation (4.39) produces a result that does not require pre-whitening, since the inverse matrix is of the same size as the number of weights. For this reason, equations (4.37) and (4.39) are the preferred approach to writing the associative memory equation.

We will now apply this approach to the top of the gas zone, which is solvable. In this case, we have the inputs

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix},\ t_1 = 1,\ \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix},\ t_2 = -1,\ and\ \boldsymbol{x}_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix},\ t_3 = -1,$$

which, by applying equation (4.37), gives the result:

$$\boldsymbol{w} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}. \tag{4.45}$$

Thus, we have found the weights exactly, without using the perceptron training rule and having to estimate the initial weights. Since the matrix X is square, we could also solve the problem using the matrix inverse rather than the pseudo-inverse. This is given by:

$$\boldsymbol{w} = \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 0.5 \\ -0.5 & 0 & 0.5 \\ -0.5 & 0.5 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}. \tag{4.46}$$

Again, we get the right answer. Notice that the matrix being inverted is the transpose of the matrix in the previous expression. This would be less obvious if we solved for the top of the gas sand, since that matrix is symmetric. That is

$$X_{top} = X_{top}^{T} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}. \tag{4.47}$$

Throughout this section and the previous section, I have discussed the single layer perceptron and methods that can be used to solve for the perceptron weighting coefficients. I illustrated all of these methods with a straightforward AVO classification problem. Despite the large array of techniques discussed, we were never able to solve this simple problem completely, since it is nonlinearly separable. In the next chapter, we will show how the multi-layer perceptron, which is composed of interconnected layers of perceptrons, is able to solve a nonlinearly separable problem. Before doing this, the last section in this chapter will briefly discuss the generalized linear discriminant, which will lead us to the radial basis function neural networks that will be covered in Chapters 6 and 7.

4.7 The generalized linear discriminant

In this chapter, I have discussed a number of approaches to linear discrimination, all of which used the straightforward model

$$y = f(\boldsymbol{w}^T \boldsymbol{x}), \tag{4.48}$$

where $\boldsymbol{x}^T = [x_0 \quad x_1 \quad \cdots \quad x_M]$ is an $M+1$-dimensional input sample vector of seismic attributes, and $\boldsymbol{w}^T = [w_0 \quad w_1 \quad \cdots \quad w_M]$ is an $M+1$-dimensional weight vector. If we let

$f(x) = x$, equation (4.48) reduces to the linear case. We found that the limitation of this approach is its inability to solve problems that are not linearly separable. In the next chapter I will show how the multi-layer perceptron is able to get around this problem. However, another approach is the generalized discriminant function, written

$$y = \boldsymbol{w}^T \phi(\boldsymbol{x}), \tag{4.49}$$

where $\phi(\boldsymbol{x})^T = [\phi(x_0) \quad \phi(x_1) \quad \cdots \quad \phi(x_M)]$ is a nonlinear function of the vectors $\boldsymbol{x}$, often called a basis function. In chapter 6, we will see how a suitable choice of the basis function will allow us to generalize the problem in such a way that we will be able to solve nonlinear problems.

CHAPTER 5 : THE MULTI-LAYER PERCEPTRON

5.1 Introduction

In the previous two chapters, I have discussed linear approaches to the regression and classification problems. As might be expected, we found that linear methods do a good job of solving linear problems, but are unable to solve nonlinear problems. In particular, I discussed the single-layer perceptron and found that we could only solve the classification problem if the classes were linearly separable. This was illustrated by applying the perceptron method to an AVO classification problem and finding that we could not separate both the top and base of a gas sand from a wet sand.

The perceptron model was introduced by McCulloch and Pitts (1943). Supervised learning using the perceptron model was first discussed by Rosenblatt (1958). At this time there was much enthusiasm about the use of perceptrons to solve a wide variety of problems. Unfortunately, Minsky and Papert (1969) showed mathematically that the fact that a perceptron can only solve linearly separable problems is a fundamental limitation of the method. This dealt a severe blow to the development of neural network algorithms based on the perceptron model. As shown by McClelland and Rumelhart (1981), the solution to this problem is to add a second layer of perceptrons. Their algorithm, called the multi-layer perceptron (MLP), will be discussed in this chapter. I will apply the MLP to both classification and regression problems. In the first section of this chapter, I will revisit the AVO classification problem and show how to solve it intuitively using the MLP. In the following section, I will discuss the mathematical approach to solving for the weights, and show that the limitation of the MLP is the need to make an initial guess of the weights, which can severely affect the final result. Finally, I will apply the MLP to a real data example.

5.2 The multi-layer perceptron

5.2.1 The general multi-layer perceptron model

The multi-layer perceptron is a generalization of the single-layer perceptron that was discussed in section 4.5, in which we add multiple perceptrons to which the inputs are fed with interconnected weights. In addition, we add an extra layer of weights in order to connect the outputs of the perceptrons themselves. Figure 5.1 shows a flowchart of a multi-layer perceptron with *M* inputs and *K* perceptrons. The first layer in the multi-layer perceptron is referred to as the input layer, the second layer as the "hidden" layer, and the output is referred to as the output layer. Although we can add any number of "hidden" layers, it has been shown that most problems can be solved with a single "hidden" layer. In all of our applications we will use only a single "hidden" layer.

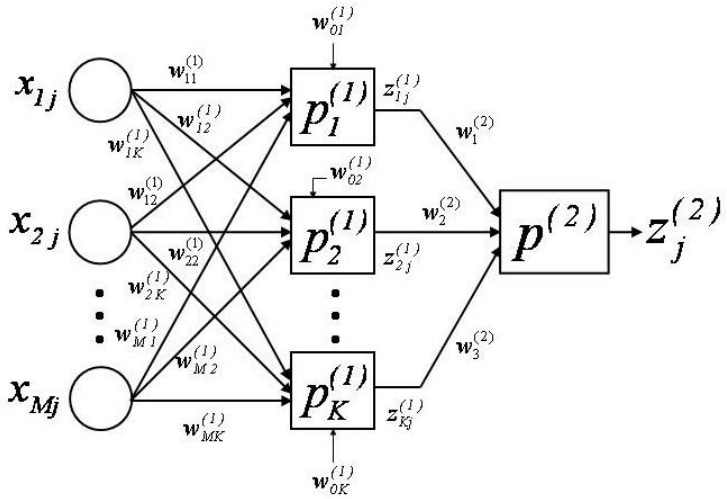


Figure 5.1: A multi-layer perceptron with *M* inputs, *K* perceptrons, and a single output.

To relate the multi-layer perceptron of Figure 5.1 to the reservoir prediction problem that we are considering in this study, the input to the multi-layer perceptron is a vector of *M* attribute values $\boldsymbol{x}_j^T = \left[x_{1j}, \quad x_{2j}, \quad \cdots, \quad x_{Mj}\right]$, where j = 1, ... , N, is the number of seismic samples. Because each input stream of values is connected to each perceptron and there are several layers of weights, the first set of weights are written as

$w^{(q)}_{ik}$, where i represents the input attribute number, k represents the perceptron number and the superscript q in brackets indicates the layer number.

As in the single layer algorithm, each perceptron consists of a set of weights, a summation step, and a nonlinear function step. The output of the weighting and summation in the first layer, the "hidden" layer, can be written:

$$y^{(1)}_{kj} = \sum_{i=0}^{M} w^{(1)}_{ki} x_{ij}, \; k = 1,2,\ldots,K, \; j = 1,2,\ldots,N. \tag{5.1}$$

In equation (5.1) the bias term has been included by letting $x_{0j} = 1$. In matrix and vector notation we can write

$$\boldsymbol{y}^{(1)}_j = W^{(1)T} \boldsymbol{x}_j, \tag{5.2}$$

where $\boldsymbol{y}^{(1)}_j = \begin{bmatrix} y^{(1)}_{1j} \\ y^{(1)}_{2j} \\ \vdots \\ y^{(1)}_{Kj} \end{bmatrix}$, $\boldsymbol{x}_j = \begin{bmatrix} 1 \\ x_{1j} \\ \vdots \\ x_{Mj} \end{bmatrix}$, *and* $W^{(1)T} = \begin{bmatrix} w^{(1)}_{10} & w^{(1)}_{20} & \cdots & w^{(1)}_{M0} \\ w^{(1)}_{11} & w^{(1)}_{21} & \cdots & w^{(1)}_{M1} \\ \vdots & \vdots & \ddots & \vdots \\ w^{(1)}_{1K} & w^{(1)}_{2K} & \cdots & w^{(1)}_{MK} \end{bmatrix}$.

Notice that the zeroth weight in equation (5.1) and (5.2) does not represent a new neuron, but rather a weight associated with a zeroth attribute, which consists of all ones. Also note that my notation convention differs from many neural network textbooks (e.g. Hagan et al., 1996) in that I have preserved the transpose operation for the weight matrix and the output vector, and write $w^{(q)}_{ik}$ rather that $w^{(q)}_{ki}$. This clarifies the connection between the multi-layer perceptron and the linear methods described in the previous two chapters.

The nonlinear function can be written as

$$z^{(1)}_{kj} = f(y^{(1)}_{kj}) \tag{5.3}$$

or, in vector form as

$$\boldsymbol{z}^{(1)}_j = f(\boldsymbol{y}^{(1)}_j). \tag{5.4}$$

The output from layer 1 is then fed into layer 2. Again, we can incorporate the bias term by letting $z_{0j}^{(1)} = 1$. The input to the single perceptron in layer 2 will thus contain $K+1$ weights, and can be written

$$y_j^{(2)} = \sum_{k=0}^{K} w_{kj}^{(2)} z_{kj}^{(1)} = \boldsymbol{w}_j^{(2)T} \boldsymbol{z}_j^{(1)}, \; j = 1,2,\ldots,N, \tag{5.5}$$

where $\boldsymbol{w}^{(2)T} = \begin{bmatrix} w_0^{(2)} & w_1^{(2)} & \cdots & w_K^{(2)} \end{bmatrix}$, and $\boldsymbol{z}_j^{(1)} = \begin{bmatrix} 1 \\ z_{1j}^{(1)} \\ \vdots \\ z_{Kj}^{(1)} \end{bmatrix}$.

The output of the second layer can then be written as

$$z_j^{(2)} = f^{(2)}(y_j^{(2)}), \tag{5.6}$$

Combining the above equations, note that the two-layer perceptron shown in Figure 5.1 can be also written in nested form as

$$z_j^{(2)} = f^{(2)}(W^{(2)T} f^{(1)}(W^{(1)T} \boldsymbol{x}_j)). \tag{5.7}$$

Now, let us discuss the functions themselves. One of the most commonly used functions in the multi-layer perceptron is the logistic function, illustrated in Figure 5.2. The logistic function is given by

$$f(x) = \text{logist}(x) = \frac{1}{1+\exp(-x)}. \tag{5.8}$$

Notice that the values of the logistic function are constrained between 0 and $+1$. In Figure 4.11 of Chapter 4 I showed two other common functions used in neural network design, the step function and the hyperbolic tangent function, in which the output is constrained between -1 and +1. Recall that the step function was used in our AVO classification example. This function will also be used in the next section.

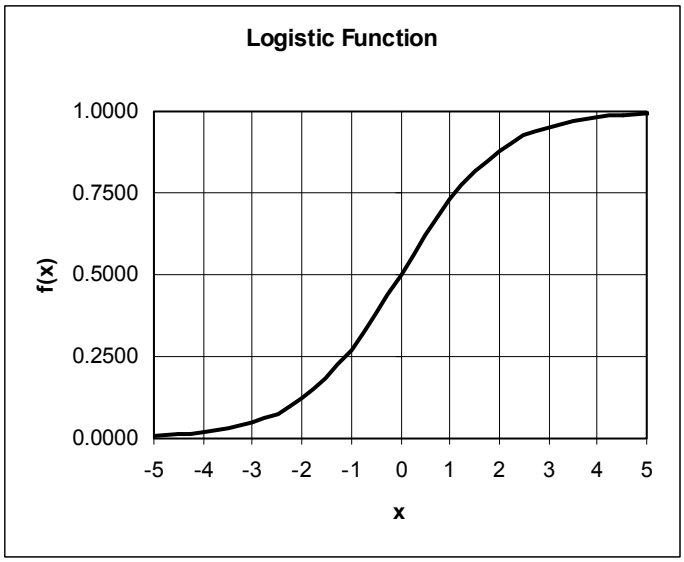


Figure 5.2: A graph of the logistic function.

The hyperbolic tangent function can be shown to be related to the logistic function in the following way

$$f(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = 2\,\text{logist(x)} - 1 \tag{5.9}$$

Finally, note that if we apply the multi-layer perceptron model to all of our input values simultaneously, equation (5.2) can be written as the matrix equation:

$$Y^{(1)} = W^{(1)T} X\,, \tag{5.10}$$

where:

$$Y^{(1)} = \left[\boldsymbol{y}_1^{(1)} \quad \boldsymbol{y}_2^{(1)} \quad \cdots \quad \boldsymbol{y}_N^{(1)} \right] = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1N} \\ y_{21} & y_{22} & \cdots & y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{K1} & y_{K2} & \cdots & y_{KN} \end{bmatrix},$$

and

$$X = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_N] = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix}.$$

This is referred to as batch processing, rather than sequential processing.

5.2.2 The multi-layer perceptron applied to AVO classification

To apply this formulation to the AVO classification problem of the last chapter, we need to recast the problem as a multi-layer perceptron. This is shown in Figure 5.3, where the inputs are still the intercept *(A)* and gradient *(B)*, but we have now interconnected the two perceptrons using the "hidden" layer concept.

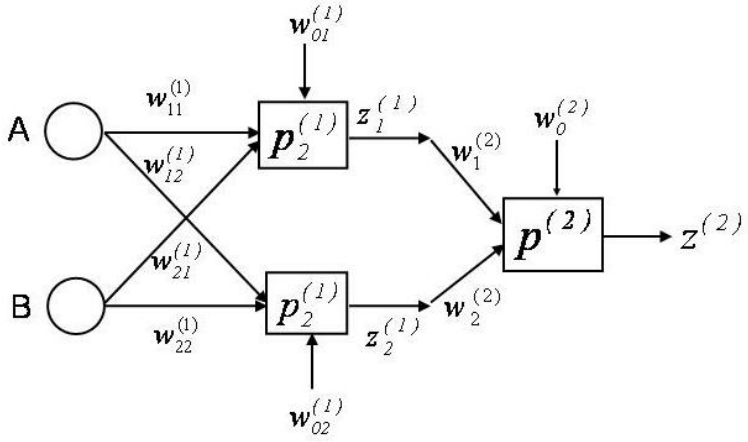


Figure 5.3: The multi-layer perceptron for the gas-water sand model of Figure 4.12.

In mathematical notation, the output from the first layer is given by

$$\boldsymbol{z}_j^{(1)} = f(W^{(1)T}\boldsymbol{x}_j), \tag{5.11}$$

where $\boldsymbol{x}_j = \begin{bmatrix} 1 \\ A_j \\ B_j \end{bmatrix}$, *and* $W^{(1)T} = \begin{bmatrix} w_{01}^{(1)} & w_{11}^{(1)} & w_{21}^{(1)} \\ w_{02}^{(1)} & w_{12}^{(1)} & w_{22}^{(1)} \end{bmatrix}$ and $f(\cdot)$ is the function given in equation (4.28).

The output from the second layer is given by

$$z_j^{(2)} = f(\boldsymbol{w}^{(2)T} \boldsymbol{z}_j^{(1)'}), \tag{5.12}$$

where $\boldsymbol{z}_j^{(1)'} = \begin{bmatrix} 1 \\ z_{1j}^{(1)} \\ z_{2j}^{(1)} \end{bmatrix}$, *and* $\boldsymbol{w}^{(2)T} = \begin{bmatrix} w_0^{(2)} & w_1^{(2)} & w_2^{(2)} \end{bmatrix}$.

Before discussing a mathematical approach to solving for the perceptron weights, we will solve the weight values intuitively. What we have done is simply connect two perceptrons to produce two separate outputs. These outputs now represent the input to a new perceptron. If we use the two perceptrons from Figure 4.12 in the previous chapter as our two first layer perceptrons, we can crossplot their outputs $z_1^{(1)}$ and $z_2^{(1)}$, and design a new perceptron, $p^{(2)}$, to separate these outputs. This is shown in Figure 5.4, using the output values from Table 4.1.

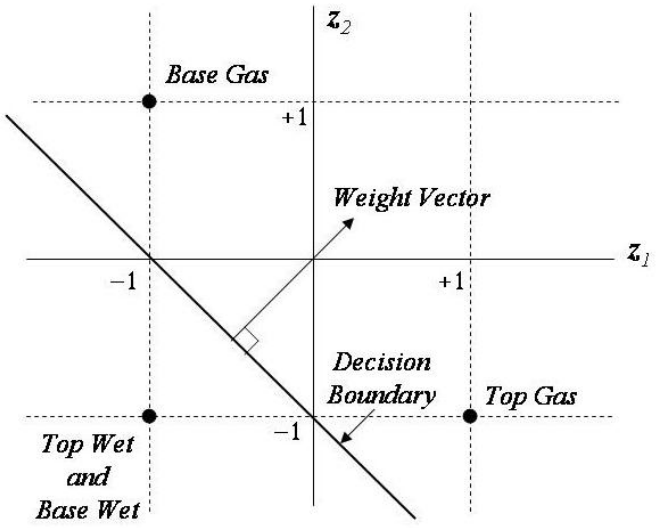


Figure 5.4: A crossplot of the outputs of the two perceptrons of Figure 4.18.

The weights for the top of gas can thus be written $w_{11}^{(1)} = w_{21}^{(1)} = -1$, and $w_{01}^{(1)} = -1$, and for the base of gas can be written $w_{12}^{(1)} = w_{22}^{(1)} = +1$ and $w_{02}^{(1)} = -1$. The top of gas output is now at the point *(-1, +1)*, the base of gas output is at the point *(+1, -1)*, and both of the wet sands have moved to the point *(-1, -1)*. The output is now linearly separable,

as is graphically shown by the decision boundary on Figure 5.4. We can therefore derive the weights for perceptron $p^{(2)}$ in Figure 5.3, which are:

$$-\frac{w_0^{(2)}}{w_1^{(2)}} = -\frac{w_0^{(2)}}{w_2^{(2)}} = -1, \text{ or } w_1^{(2)} = w_2^{(2)} = w_0^{(2)} = +1 .$$

The final perceptron, with its weights annotated, is shown in Figure 5.5. Notice that the weights are all equal to *+1* or *–1*, which are the simplest set of weights. This solution is not unique, and there are also many other sets of weights that would solve the problem.

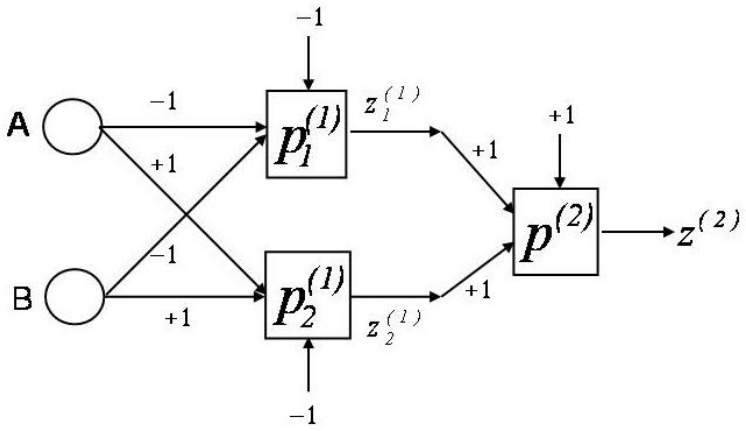


Figure 5.5: The final multi-layer perceptron weights.

To verify that this is the solution, Table 5.1 shows the outputs for all inputs.

	Inputs		**Perceptron** $p_1^{(1)}$	**Perceptron** $p_2^{(1)}$	**Perceptron** $p^{(2)}$
Sand	**A**	**B**	$z_1^{(1)}$	$z_2^{(1)}$	$z^{(2)}$
Top Gas	-1	-1	+1	-1	+1
Base Wet	-1	+1	-1	-1	-1
Top Wet	+1	-1	-1	-1	-1
Base Gas	+1	+1	-1	+1	+1

Table 5.1: The computed output values from the multi-layer perceptron in Figure 5.5 for the gas and wet models of Figure 4.12, where $y^{(2)}$ shows that the gas sand values (*+1*) have been separated from the wet sand values (*-1*).

As we can see in Table 5.1, the correct output is given for all four input cases. Thus, we have solved the problem of how to separate a Class 3 gas sand from its equivalent wet sand.

Although we have solved the weights intuitively in the preceding discussion, we need a more rigorous approach to finding the weights. This can be done using a technique called *backpropagation*, in which we reduce the error between the known output and the actual output by backward propagation of the errors. This will be described in the next section.

5.3 Computing the weights for the multi-layer perceptron

5.3.1 The backpropagation algorithm

In Chapter 4, I showed that the weights for a single layer perceptron could be solved using a single matrix inversion, since the problem was linear. However, when we add a second layer of weights, the problem becomes nonlinear, and we cannot solve for the weights using a straightforward matrix inversion. To solve for these weights in the multilayer case I use a generalization of the LMS algorithm which was invented by Rummelhart et al. (1986). This technique is referred to as error backpropagation, since the errors are backpropagated through the network and the weights are updated to reduce the error. This has led to some confusion, because the network is often referred to as a backpropagation network, although the calculations themselves proceed in a forward direction. As mentioned in the last section, although this technique can be generalized to any number of hidden layers, we will work with a network with a single hidden layer, referred to as a two-layer network.

In section 5.2 I described the forward model for the multi-layer perceptron. As shown in Figure 5.1, the input values for a given input $\boldsymbol{x_j}$ are transformed to an output value $z^{(2)}$, where the superscript *2* represents the output of the second layer. The basic idea in the computation of the weights is that this output is compared to the training value

t, and the weights are updated based on the error between the two values. This is called error backpropagation, because the errors are backpropagated through the network and used to improve the fit between the actual output and the training value. To start the process, the weights are initialized to small random values. We then forward propagate the solution using these initial weights. Once the output value has been computed, we minimize the squared error, which can be written as a function of the weights as

$$E(\boldsymbol{w}) = \frac{1}{2}(t_j - z_j^{(2)})^2 . \tag{5.13}$$

In equation (5.13) there is no superscript on the weights since we will use this equation to compute the values for both sets of weights. The difference in the weight values can be evaluated using the gradient descent method (Duda et al., 2001), in which we differentiate the error term with respect to the weights, giving

$$\Delta \boldsymbol{w} = -\eta \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}, \tag{5.14}$$

where η is a scaling value between 0 and 1. The weights are updated iteratively by the equation

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \Delta \boldsymbol{w}(n) . \tag{5.15}$$

To implement this procedure, we start with the output layer weights. Using the chain rule, and computing the derivative for each element in the weight vector, we get

$$\frac{\partial E_j}{\partial w_{kj}^{(2)}} = \frac{\partial E_j}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial w_{kj}^{(2)}} = -\delta_j^{(2)} \frac{\partial y_j^{(2)}}{\partial w_{kj}^{(2)}}, \tag{5.16}$$

where $\delta_j^{(2)}$ is the error term for the second layer. To evaluate the individual derivatives in equation (5.16), note that

$$\delta_j^{(2)} = \frac{\partial E_j}{\partial y_j^{(2)}} = \frac{\partial E_j}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial y_j^{(2)}} = (t_j - z_j^{(2)}) f'(y_j^{(2)}) , \tag{5.17}$$

where the prime denotes differentiation. From equation (5.11) we see that the second term in equation (5.17) can be computed as

$$\frac{\partial y_j^{(2)}}{\partial w_{kj}^{(2)}} = z_{kj}^{(1)}. \tag{5.18}$$

Combining the preceding equations, we find that

$$\Delta w_{kj}^{(2)} = \eta \delta_j^{(2)} z_{kj}^{(1)} = \eta (t_j - z_j^{(2)}) f'(y_j^{(2)}) z_{kj}^{(1)}. \tag{5.19}$$

We next turn our attention to the first layer weights. Using the same approach, we find that

$$\frac{\partial E_j}{\partial w_{ki}^{(1)}} = \frac{\partial E_j}{\partial z_{kj}^{(1)}} \frac{\partial z_{kj}^{(1)}}{\partial y_{kj}^{(1)}} \frac{\partial y_{kj}^{(1)}}{\partial w_{ki}^{(1)}}. \tag{5.20}$$

Computing the first term on the right hand side of equation (5.20) using the chain rule and equation (5.18), we get

$$\begin{aligned} \frac{\partial E_j}{\partial z_{kj}^{(1)}} &= \frac{\partial}{\partial z_{kj}^{(1)}} \left[\frac{1}{2} (t_j - z_j^{(2)})^2 \right] = -(t_j - z_j^{(2)}) \frac{\partial z_j^{(2)}}{\partial z_{kj}^{(1)}} \\ &= -(t_j - z_j^{(2)}) \frac{\partial z_j^{(2)}}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial z_{kj}^{(1)}} \\ &= -(t_j - z_j^{(2)}) f'(y_j^{(2)}) w_{kj}^{(2)} \\ &= -\delta_j^{(2)} w_{kj}^{(2)}. \end{aligned} \tag{5.21}$$

After a second application of the chain rule we find that the derivative of the error with respect to the first layer outputs is equal to the second layer error term multiplied by the second layer weights. From equation (5.16), next note that the second term on the right hand side of equation (5.21) is given by

$$\frac{\partial z_{kj}^{(1)}}{\partial y_{kj}^{(1)}} = f'(y_{kj}^{(1)}). \tag{5.22}$$

Finally, by equation (5.13), note that the third term in equation (5.22) can be written

$$\frac{\partial y_{kj}^{(1)}}{\partial w_{ki}^{(1)}} = x_{ij} . \tag{5.23}$$

Combining equation (5.20) through (5.23) along with equation (5.13), we get

$$\Delta w_{ki}^{(1)} = \eta \delta_j^{(2)} w_{kj}^{(2)} f'(y_{kj}^{(1)}) x_{ij} = \eta \delta_j^{(1)} x_{ij} . \tag{5.24}$$

where $\delta_j^{(2)}$ is the first layer error term.

To implement the backpropagation algorithm, all that remains is to compute the derivatives of the functions used in the neural network. This explains why the logistic and tanh functions are used so often, since their derivatives are easy to compute. The derivative of the logistic function is written

$$f'(x) = f(x)[1 - f(x)] = \frac{\exp(-x)}{1 + \exp(-x)} , \tag{5.25}$$

and the derivative of the hyperbolic tangent function is written

$$f'(x) = 1 - f(x)^2 = \frac{1}{[\exp(x) + \exp(-x)]^2} . \tag{5.26}$$

5.3.2 An AVO classification example

To understand the practical aspects of backpropagation, I will first apply the method to two simple model examples. The first example is the AVO classification problem that was considered in the last section. In that discussion, we computed the weights intuitively. In this section, I will use the backpropagation technique to find the weights.

The flow chart for the implementation of the MLFN network used to solve for the AVO classification process is shown in Figure 5.6. To solve this problem, I will reformulate the equations of section 5.3.1 in matrix form. Before letting the computer take over, I will manually compute a single iteration through the process.

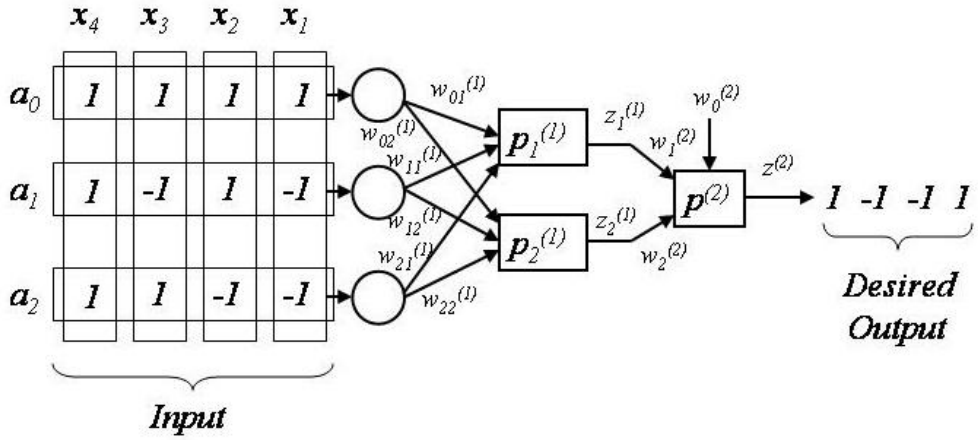


Figure 5.6: The inputs and desired outputs for the AVO classification problem.

Let us start with the forward modelling. Our input, as seen in Figure 5.6, can be written

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}.$$

A critical step is the initialization of the weights. These are set to random numbers between *0* and *1*. The choice of these random numbers will determine the convergence of the algorithm. Using a random number generator, the first layer weights are initially set to

$$W_{(0)}^{(1)} = \begin{bmatrix} w_{01}^{(1)} & w_{02}^{(1)} \\ w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.9501 & 0.4860 \\ 0.2311 & 0.8913 \\ 0.6068 & 0.7621 \end{bmatrix},$$

where the value in the subscripted bracket refers to the 0^{th} iteration. Applying these weights in the forward process gives the following output from the first layer summation

$$\mathbf{y}_{(0)}^{(1)} = W_{(0)}^{(1)T} X = \begin{bmatrix} 0.1121 & 0.5744 & 1.3258 & 1.7881 \\ -1.1674 & 0.6152 & 0.3568 & 2.1394 \end{bmatrix}.$$

Taking the hyperbolic tangent of each value in the above matrix, and adding a row of ones for the bias values in the second layer, we get

$$\boldsymbol{z}_{(0)}^{(1)} = \tanh(\boldsymbol{y}_{(0)}^{(1)}) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.1121 & 0.5744 & 1.3258 & 1.7881 \\ -1.1674 & 0.6152 & 0.3568 & 2.1394 \end{bmatrix}.$$

The second layer weights are then initialized to

$$\boldsymbol{w}_{(0)}^{(2)} = \begin{bmatrix} w_0^{(2)} \\ w_1^{(2)} \\ w_2^{(2)} \end{bmatrix} = \begin{bmatrix} 0.4565 \\ 0.0185 \\ 0.8214 \end{bmatrix},$$

and the output of the second layer summation is

$$\boldsymbol{y}_{(0)}^{(2)} = \boldsymbol{w}_{(0)}^{(1)T} \boldsymbol{z}_{(0)}^{(1)} = \begin{bmatrix} -0.2178 & 0.9160 & 0.7538 & 1.2729 \end{bmatrix}.$$

Taking the hyperbolic tangent of each value in the above matrix, we get

$$\boldsymbol{z}_{(0)}^{(2)} = \tanh(\boldsymbol{y}_{(0)}^{(2)}) = \begin{bmatrix} -0.2145 & 0.7240 & 0.6374 & 0.8546 \end{bmatrix}.$$

We will now compute the difference between this result and the training values of

$$\boldsymbol{t} = \begin{bmatrix} 1.0 & -1.0 & -1.0 & 1.0 \end{bmatrix},$$

which is given by

$$\boldsymbol{e}_{(0)}^{(2)} = \boldsymbol{t} - \boldsymbol{z}_{(0)}^{(2)} = \begin{bmatrix} 1.2145 & -1.7240 & -1.6374 & 0.1454 \end{bmatrix},$$

and the RMS error is given by $e_{RMS} = 2.6738$.

This is a large error, so we will now use the backpropagation method to improve the estimate of the weights. The two key parameters in the backpropagation algorithm are the learning constant η and the number of iterations. In our test, we will set η to 0.2 and use 1000 iterations. We will do the first iteration manually, and then let the computer take over. First we calculate $\boldsymbol{\delta}^{2)}$ which, using the derivative form shown in equation (5.26), is given by

$$\boldsymbol{\delta}^{(2)} = f'(\boldsymbol{y}^{(2)}) \cdot \boldsymbol{e}_{(0)}^{(2)} = (1 - \boldsymbol{z}^{(2)2}) \cdot \boldsymbol{e}_{(0)}^{(2)} = \begin{bmatrix} 1.1586 & -0.8203 & -0.9722 & 0.0392 \end{bmatrix},$$

where the square and multiplication operations are applied element by element. The update for the second layer weights is thus given by

$$\boldsymbol{\Delta}^{(2)} = \eta \cdot \boldsymbol{z}^{(1)} \cdot \boldsymbol{\delta}^{(2)T} = \begin{bmatrix} -0.1189 \\ -0.2206 \\ -0.3396 \end{bmatrix},$$

and the updated weights are

$$\boldsymbol{w}_{(1)}^{(2)} = \boldsymbol{w}_{(0)}^{(2)} + \Delta^{(2)} = \begin{bmatrix} 0.3375 \\ -0.2021 \\ 0.4818 \end{bmatrix}.$$

Next we calculate $\boldsymbol{\delta}^{(1)}$, which is given by

$$\boldsymbol{\delta}^{(1)} = (1 - \boldsymbol{z}^{(1)2}) \cdot (\boldsymbol{w}_{(0)}^{(2)} \boldsymbol{\delta}^{(2)}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.0212 & -0.0111 & -0.0044 & 0.0001 \\ 0.3064 & -0.4716 & -0.7049 & 0.0017 \end{bmatrix}.$$

The weight update for the first layer weights is then given by

$$\boldsymbol{\Delta}^{(1)} = \eta X \boldsymbol{\delta}^{(1)T} = \begin{bmatrix} 0 & 0.0011 & -0.1737 \\ 0 & -0.0056 & -0.0143 \\ 0 & -0.0029 & -0.1076 \end{bmatrix}.$$

The zeros in the first column are superfluous and can be dropped, leaving a *3x2* matrix which, when added to the initial weights gives

$$W_{(1)}^{(1)} = W_{(0)}^{(1)} + \Delta^{(1)} = \begin{bmatrix} 0.9513 & 0.3123 \\ 0.2256 & 0.8770 \\ 0.6040 & 0.6545 \end{bmatrix}.$$

Now that we have the updated weights, we can repeat the forward calculation, giving a new output of

$$\boldsymbol{z}_{(1)}^{(2)} = \begin{bmatrix} -0.0911 & 0.4370 & 0.2022 & 0.5405 \end{bmatrix},$$

with an RMS error of $e_{RMS} = 2.2163$. The error is going down, but very slowly. We will therefore let the computer compute the next 999 iterations. Figure 5.7 shows the RMS error after each of the 1000 iterations. Notice in Figure 5.7 that the error decays smoothly nearly to zero, indicating that after 1000 iterations we have converged to a reasonable answer. The final computed values are

$$z^{(2)}_{(1000)} = \begin{bmatrix} 0.9736 & -0.9806 & -0.9806 & 0.9734 \end{bmatrix},$$

and the error has decreased to $e_{RMS} = 0.0464$. The weights after 1000 iterations are given by

$$W^{(1)}_{(1000)} = \begin{bmatrix} 1.5024 & -1.3779 \\ 1.6564 & 1.5539 \\ 1.6556 & 1.5534 \end{bmatrix}, \text{ and } \boldsymbol{w}^{(2)}_{(1000)} = \begin{bmatrix} 2.3006 \\ -2.5788 \\ 2.5885 \end{bmatrix}.$$

Notice that these weights are somewhat different than the weights we derived intuitively in the previous section.

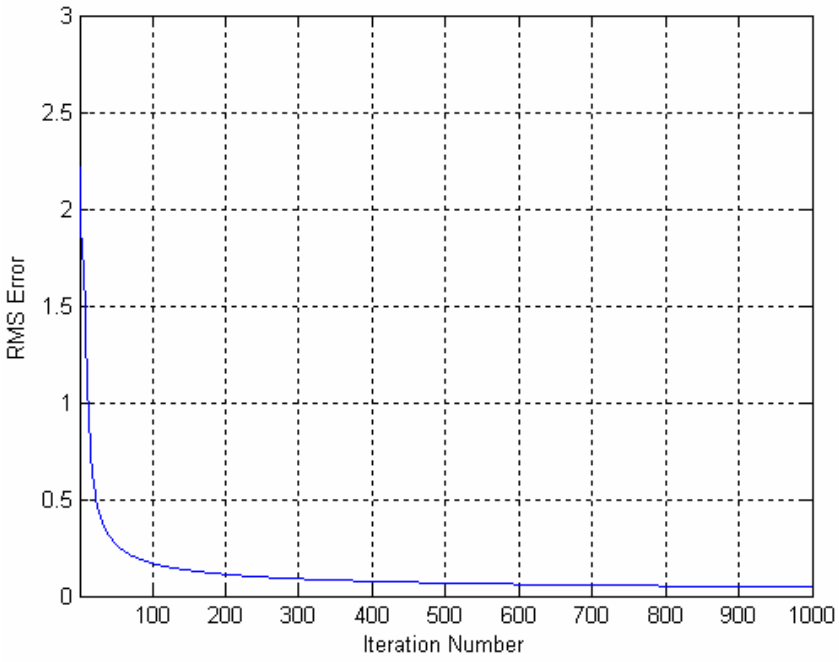


Figure 5.7: Convergence of the backpropagation algorithm for the AVO classification problem using the first example given above.

From this first example it would appear that, with enough iterations, we can approximate the right answer to any desired level of accuracy. However, let us see what happens if we choose a different set of initial weights. In this second case, we let the initial weights be

$$W_{(0)}^{(1)} = \begin{bmatrix} 0.4103 & 0.3529 \\ 0.8936 & 0.8132 \\ 0.0579 & 0.0099 \end{bmatrix}, \text{ and } \boldsymbol{w}_{(0)}^{(2)} = \begin{bmatrix} 0.1389 \\ 0.2028 \\ 0.1987 \end{bmatrix},$$

leading to an initial output of

$$\boldsymbol{z}_{(0)}^{(2)} = \begin{bmatrix} -0.0483 & 0.4411 & -0.0264 & 0.4469 \end{bmatrix},$$

with RMS error of 2.0072. Figure 5.8 show the RMS error convergence for 1000 iterations using this set of weights. Notice that the algorithm now gets trapped in a local minimum, and does not to converge to an acceptable answer.

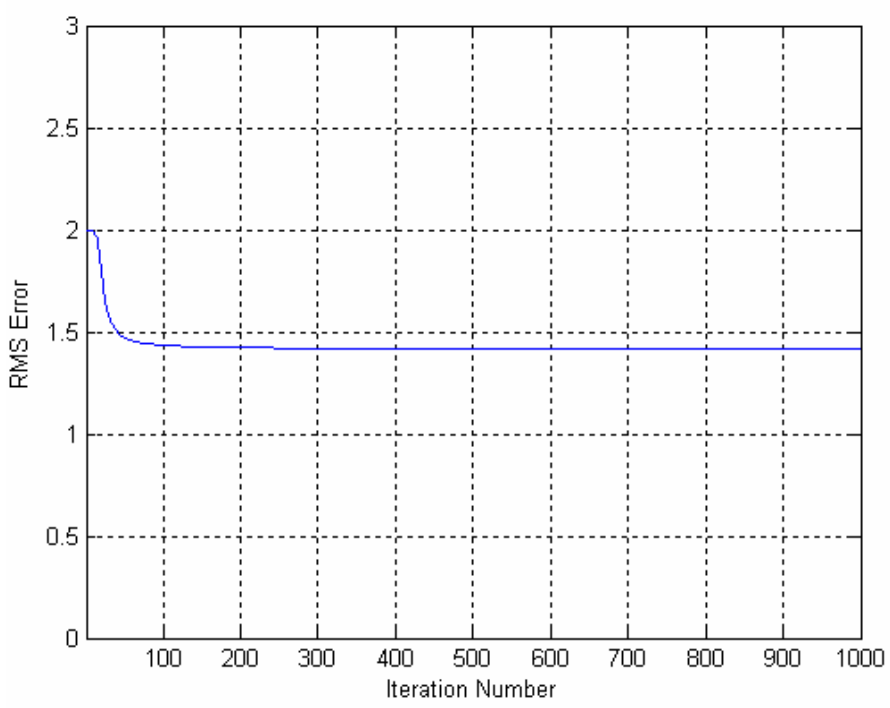


Figure 5.8: Convergence of the backpropagation algorithm for the AVO classification problem using the second example given above.

The final weights are for the convergence shown in Figure 5.8 are

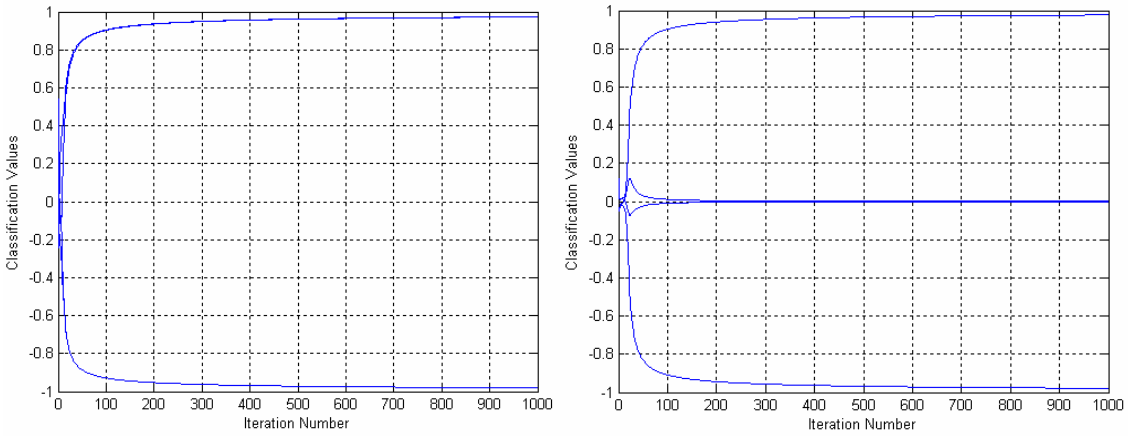
$$W_{(1000)}^{(1)} = \begin{bmatrix} 2.4303 & 2.5525 \\ 2.4744 & 2.5148 \\ 0.7535 & -0.8651 \end{bmatrix}, \text{ and } \boldsymbol{w}_{(1000)}^{(2)} = \begin{bmatrix} -0.0888 \\ -1.6353 \\ 1.7242 \end{bmatrix},$$

and the final output is

$$\boldsymbol{z}_{(0)}^{(2)} = \begin{bmatrix} 0.9772 & 0.0008 & -0.9784 & -0.0007 \end{bmatrix},$$

with RMS error of 1.4156. Comparing these results with the first results, it can be seen that the last row of the first weight matrix is much too low, as is the bias value in the second set of weights. The result is an output that finds the first and third value correctly, but not the second and fourth.

As a final display, let us look at the estimated classification values after each iteration. This is shown in Figure 5.9, where Figure 5.9(a) shows the estimates for the case shown in Figure 5.7, and Figure 5.9(b) shows the estimates for the case shown in Figure 5.8. In Figure 5.9(a) the positive and negative pairs fall on top of each other and converge to the right answer in much the same way as the error plot. In Figure 5.9(b), the four cases are distinct and show that two of the classification values refused to converge. This shows how sensitive the process is to the initial estimate of the weights.



(a) (b)

Figure 5.9: Convergence of the classification values in backpropagation algorithm, where (a) shows the case from Figure 5.7, and (b) shows the case from Figure 5.8.

5.3.3 A sine wave example

As mentioned in Chapter 1, neural networks can be used for either classification or function prediction, that is an example of nonlinear regression where we know which function we are trying to interpolate. Our first example was a classification problem, in which we classified a sand reservoir as being either gas-charged or wet. In our second example, we will use the backpropagation network to interpolate the function $\sin(\pi t)$ using a training set consisting of four measured points from the sine wave. To perform this interpolation, we will use a different type of neural network than was used in the first example, called a 1-K-1 neural network (Hagan et al., 1996). This type of network accepts a scalar input (rather than the two-dimensional vector used in our classification example), passes it through K neurons, then combines the output from these K neurons using a single neuron, and outputs a scalar value. We will use the simplest case of $K = 2$, and will use different functions for the neurons in each layer. For the first layer, the hidden layer, we will use the logistic function given by equation (5.9), with derivative given by equation (5.25). For the second layer, the output layer, we will use the linear function given by $f(x) = x$, with a derivative of 1. Figure 5.10 shows the flowchart for the 1-2-1 neural network.

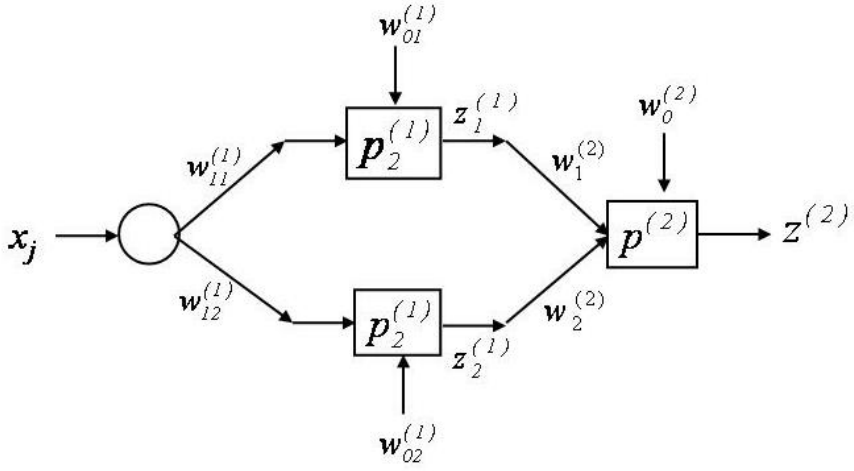


Figure 5.10: The 1-2-1 neural network used in this section to interpolate a sine wave.

In our test, I will use four input values at x values of 1/8, 3/8, 5/8, and 7/8. This means that our X matrix can be written:

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1/8 & 3/8 & 5/8 & 7/8 \end{bmatrix}$$

The training values are therefore at angles of $\pi/4$, $3\pi/4$, $5\pi/4$, and $7\pi/4$ and are given by

$$\begin{aligned} \boldsymbol{t} &= \begin{bmatrix} \sin(\pi/4) & \sin(3\pi/4) & \sin(5\pi/4) & \sin(7\pi/4) \end{bmatrix} \\ &= \begin{bmatrix} \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \end{bmatrix} \end{aligned}$$

Using a random number generator, the first-layer weights for the zeroth iteration are set to

$$W_{(0)}^{(1)} = \begin{bmatrix} w_{01}^{(1)} & w_{02}^{(1)} \\ w_{11}^{(1)} & w_{12}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.9501 & 0.6068 \\ 0.2311 & 0.4860 \end{bmatrix}.$$

Applying these weights in the forward process gives the following output from the first layer summation

$$\boldsymbol{y}_{(0)}^{(1)} = W_{(0)}^{(1)T} X = \begin{bmatrix} 0.9790 & 1.0368 & 1.0946 & 1.1524 \\ 0.6676 & 0.7891 & 0.9106 & 1.0321 \end{bmatrix}.$$

Applying the logistic function to each value in the above matrix, and adding a row of ones for the bias values in the second layer, we get

$$\boldsymbol{z}_{(0)}^{(1)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.7269 & 0.7382 & 0.7492 & 0.7599 \\ 0.6610 & 0.6876 & 0.7131 & 0.7373 \end{bmatrix}.$$

The second layer weights are then initialized to

$$\boldsymbol{w}_{(0)}^{(2)} = \begin{bmatrix} w_0^{(2)} \\ w_1^{(2)} \\ w_2^{(2)} \end{bmatrix} = \begin{bmatrix} 0.8913 \\ 0.7621 \\ 0.4565 \end{bmatrix},$$

and the output of the second layer summation and function application, since the function is linear, is

$$\boldsymbol{z}_{(0)}^{(2)} = \boldsymbol{y}_{(0)}^{(2)} = \boldsymbol{w}_{(0)}^{(1)T} \boldsymbol{z}_{(0)}^{(1)} = \begin{bmatrix} 1.7470 & 1.7678 & 1.7878 & 1.8070 \end{bmatrix}.$$

We will now compute the difference between this result and the training values, that is given by

$$\boldsymbol{e}_{(0)}^{(2)} = \boldsymbol{t} - \boldsymbol{z}_{(0)}^{(2)} = \begin{bmatrix} -1.0399 & -1.0607 & -2.4949 & -2.5141 \end{bmatrix},$$

and the RMS error is given by $e_{RMS} = 3.8408$.

As in the previous case, the two key parameters in the backpropagation algorithm are the learning constant η and the number of iterations. In our test, I will set η to *0.1* and use 12,000 iterations. (I chose 12,000 iterations by trial and error. Another approach to finding the number of iterations is to stop iterating when some convergence criterion is met.) Again, I will do the first iteration manually, since this problem is quite different than the first one, and then I will let the computer take over. First we calculate $\boldsymbol{\delta}^{(2)}$ which, since the derivative of a linear function is equal to one, is given by

$$\boldsymbol{\delta}^{(2)} = f'(\boldsymbol{y}^{(2)}) \cdot \boldsymbol{e}_{(0)}^{(2)} = \boldsymbol{e}_{(0)}^{(2)}.$$

This is simply equal to the error between the initial calculation and the training values, given above. The update for the second layer weights is given by

$$\boldsymbol{\Delta}^{(2)} = \eta \cdot \boldsymbol{z}^{(1)} \cdot \boldsymbol{\delta}^{(2)T} = \begin{bmatrix} -0.7110 \\ -0.5319 \\ -0.5050 \end{bmatrix},$$

and the updated weights are

$$\boldsymbol{w}_{(1)}^{(2)} = \boldsymbol{w}_{(0)}^{(2)} + \boldsymbol{\Delta}^{(2)} = \begin{bmatrix} 0.1803 \\ 0.2302 \\ -0.0485 \end{bmatrix}.$$

Next we calculate $\boldsymbol{\delta}^{(1)}$, in which we need the derivative of the logistic function, given in equation (5.25). This gives us

$$\boldsymbol{\delta}^{(1)} = (1 - z^{(1)2}) \cdot (\boldsymbol{w}_{(0)}^{(2)} \boldsymbol{\delta}^{(2)}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -0.1573 & -0.1562 & -0.3572 & -0.3495 \\ -0.1064 & -0.1040 & -0.2330 & -0.2223 \end{bmatrix}.$$

The weight update for the first layer weights is then given by

$$\Delta^{(1)} = \eta X \boldsymbol{\delta}^{(1)T} = \begin{bmatrix} 0 & -0.1020 & -0.666 \\ 0 & -0.607 & -0.0392 \end{bmatrix}.$$

The zeros in the first column are superfluous and can be dropped, leaving a *2x2* matrix that, when added to the initial weights gives

$$W_{(1)}^{(1)} = W_{(0)}^{(1)} + \Delta^{(1)} = \begin{bmatrix} 0.8481 & 0.5403 \\ 0.1704 & 0.4467 \end{bmatrix}.$$

Now that we have the updated weights, we can repeat the forward calculation, giving a new output of

$$z_{(1)}^{(2)} = \begin{bmatrix} 0.3113 & 0.3121 & 0.3129 & 0.3137 \end{bmatrix},$$

with an RMS error of $e_{RMS} = 1.5477$. The error is going down, and the values are all roughly the same size, but we know that the last two values should be the same absolute value but negative. We will therefore let the computer compute the next 11,999 iterations. Figure 5.11(a) shows the RMS error after each of the 12,000 iterations. (The reason that the first value on the plot does not seem to agree with the first error given above is that the error in the first ten iterations is very steep, and the points are not shown at this scale.) Notice that the error decays smoothly to close to zero, indicating that after 12,000 iterations we have converged to a reasonable answer.

The final computed values are

$$z_{(12000)}^{(2)} = \begin{bmatrix} 0.7085 & 0.7047 & -0.7045 & -0.7087 \end{bmatrix},$$

and the error has decreased to $e_{RMS} = 0.0041$. The weights after 12,000 iterations are given by

$$W^{(1)}_{(12000)} = \begin{bmatrix} -1.0483 & -6.0168 \\ 2.7847 & 12.3039 \end{bmatrix}, \text{ and } \boldsymbol{w}^{(2)}_{(12000)} = \begin{bmatrix} -0.3823 \\ 3.3917 \\ -3.0671 \end{bmatrix}.$$

In Figure 5.11(b) I show the results of applying the derived weights to interpolate values away from the training points. Notice that the approximation is quite good between the centre two training points, but underestimates the true value between the other pairs of training points and overestimates the true value at the ends of the function. The error has converged to close to zero, but is far from being uniform in convergence, showing a large change after about 300 iterations.

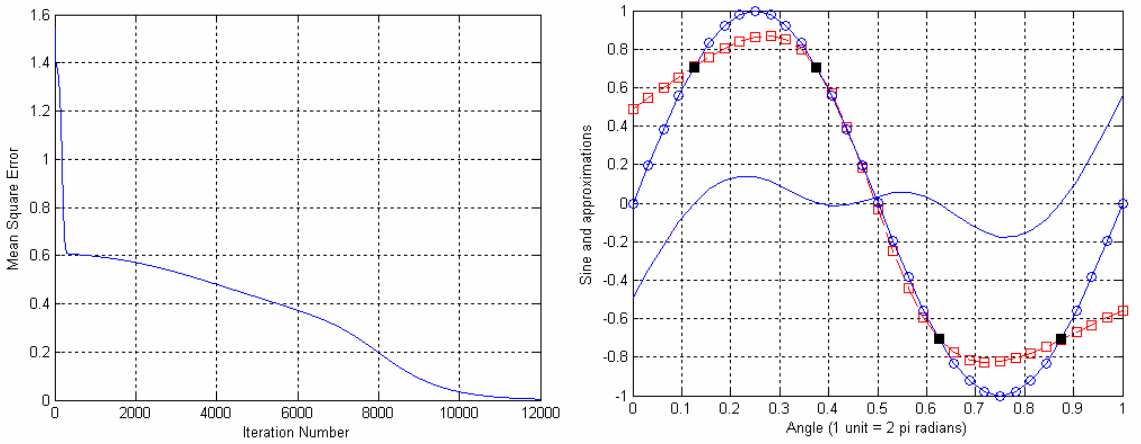


Figure 5.11. Interpolation of a sine wave using the backpropagation network shown in Figure 5.10, where (a) shows the error convergence after 10,000 iterations, and (b) shows the correct answer as open blue circles, the training points as four black squares, the predictions as open red squares, and the error between the predicted and true values as a solid blue line.

It is also useful to display the four estimated sine wave values for each iteration of the algorithm. This is shown in Figure 5.12. In this figure, note that the algorithm gets the sign of the values correct after very few iterations, but underestimates the size of one value and overestimates the other for both the positive and negative pairs. Also, there is almost perfect symmetry between the two pairs of values.

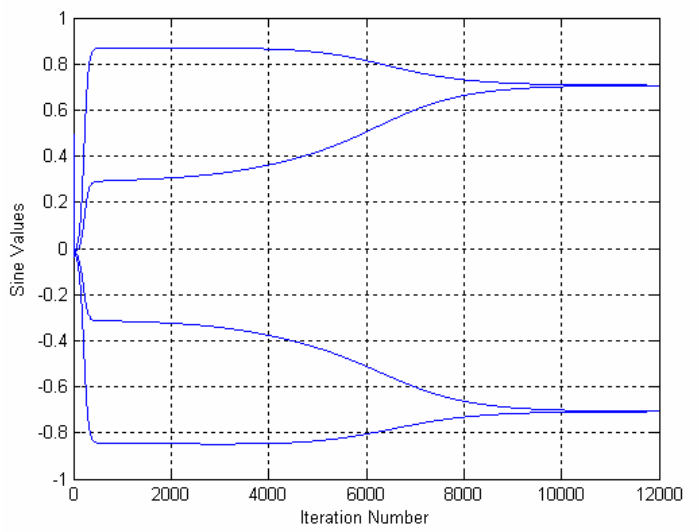
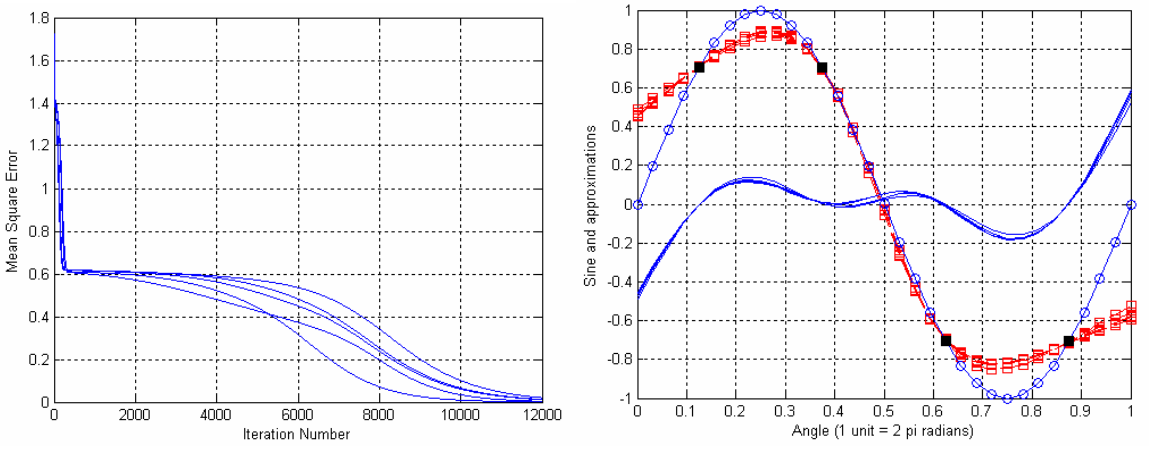


Figure 5.12. The estimate of each of the four sine wave values (the black squares in Figure 5.11(a)) after each iteration of the neural network.

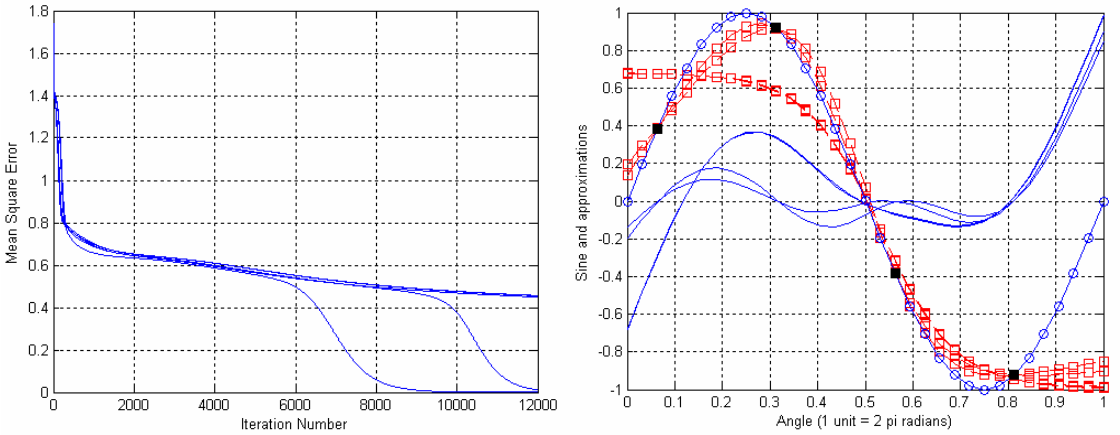
Next, let us see what happens if we change the random guess on the initial weights. Figure 5.13 shows five separate solutions of 12,000 iterations each, starting with a different set of random numbers each time. Notice that although each estimate is close to the estimate in Figure 5.12, they all differ slightly.



(a) (b)

Figure 5.13: Same as Figure 5.11, except that five separate interpolations have been done using different sets of random initial weights.

Next, let us vary the training points. Figure 5.14 shows five separate solutions of 12,000 iterations each, starting with a different set of random numbers each time. However, this time the training values are at π/8, 5π/8, 9π/8, and 13π/8.



(a) (b)

Figure 5.14. Same as Figure 5.13, except that the training points have been shifted by π/8.

In Figure 5.14, note that only two of the five initial starting estimates have converged to the right answer, and these do so after different numbers of iterations. In the best case, the one that converged after 9000 iterations, the training values and final answer are

$$\boldsymbol{t} = [0.3827 \quad 0.9239 \quad -0.3827 \quad -0.9239],$$

and

$$\boldsymbol{z}^{(2)}_{(12000)} = [0.3827 \quad 0.9238 \quad -0.3827 \quad -0.9239],$$

with an error of only 0.00047. So this first run has done an excellent job.

The convergence of the individual values is shown in Figure 5.15. Because the points are not symmetric, as in our first example, the convergence is no longer symmetric.

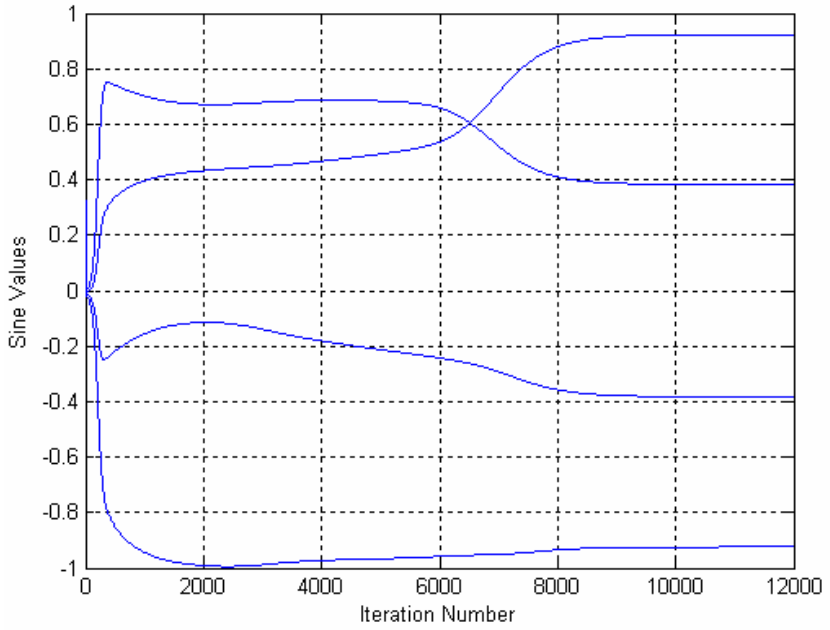


Figure 5.15. The estimated sine wave values after each iteration of the neural network.

What we can conclude from both this model study and the previous one is that the multi-layer perceptron is highly sensitive to both the initial guess of the weights and also the number of iterations used in the solution. If we are lucky, we will converge to a correct answer after a reasonable number of iterations. If we are not lucky, we will either require a very large number of iterations, making the process costly in time and computer usage, or may get locked in a local minimum and never converge at all. The next section will discuss several methods for trying to avoid these problems.

5.4 Advanced methods for backpropagation

5.4.1 Introduction

In the examples just given, I have used the traditional LMS backpropagation method to compute weights in our neural networks. However, this method has been shown to be quite slow and highly dependent on the initial guess. Several more advanced methods have been developed over the years to improve the performance of network training. The two methods that will be used in the next section are the conjugate gradient

method and the simulated annealing method. Before discussing these methods, I will quickly review the basics of function optimization (Gill et al., 1981, Hagan et al., 1996). These methods can be best understood by considering the vector form of the Taylor series expansion, that can be written

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \boldsymbol{g}^T \Delta \boldsymbol{x} + \Delta \boldsymbol{x}^T H \Delta \boldsymbol{x}, \tag{5.27}$$

where $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\boldsymbol{g} = \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$, $H = \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$, and

$\Delta \boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_o$. The vector of derivatives $\boldsymbol{g}$ is called the gradient, and the symmetric matrix H is called the Hessian. If $f(\boldsymbol{x})$ is the quadratic function given by

$$f(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}^T A \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c, \tag{5.28}$$

where A is also a symmetric matrix, we find that

$$\nabla f(\boldsymbol{x}) = A \boldsymbol{x} + b, \tag{5.29}$$

and

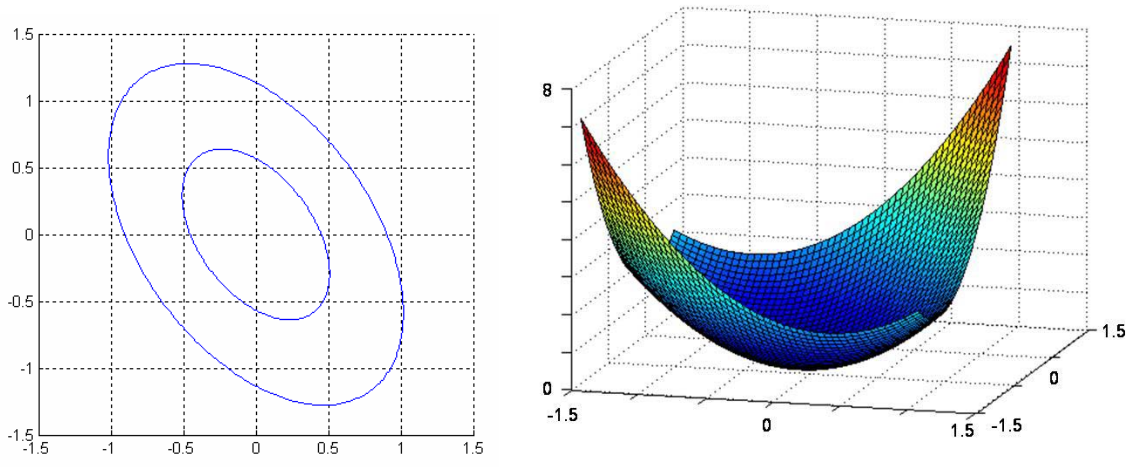
$$\nabla^2 f(\boldsymbol{x}) = A, \tag{5.30}$$

which shows that A is equivalent to the Hessian.

As a simple example of this theory, that will be used to illustrate the gradient search methods described in the next two sections, let $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$, and consider the quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}^T A \boldsymbol{x} = \frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x^2 + xy + y^2, \tag{5.31}$$

that is the ellipse shown in Figure 5.16, both in contour and perspective form.



(a) (b)

Figure 5.16. The elliptical function used for the explanation of optimization methods, showing (a) the contours of the function, and (b) a perspective plot of the function.

For this function, the gradient is

$$\boldsymbol{g} = \nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ 2y + x \end{bmatrix}, \tag{5.32}$$

and the Hessian is

$$H = \nabla^2 f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{5.33}$$

5.4.2 The gradient descent method

To optimize the function $f(\boldsymbol{x})$, we must find the value of $\boldsymbol{x}$ that gives us the minimum value of $f(\boldsymbol{x})$. This can be done iteratively using the equation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \, \boldsymbol{p}_k, \tag{5.34}$$

where k is the iteration number, $\boldsymbol{p}_k$ is a search direction, and α_k is the learning rate (Hagan et al., 1996).

Note that equation (5.34) can also be written

$$\Delta \boldsymbol{x}_k = \alpha \, \boldsymbol{p}_k, \tag{5.35}$$

where $\Delta \boldsymbol{x}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$. Dropping the third term of equation (5.27) and substituting $\boldsymbol{x}_{k+1}$ for $\boldsymbol{x}$ and $\boldsymbol{x}_k$ for $\boldsymbol{x}_0$, we get

$$f(\boldsymbol{x}_{k+1}) = f(\boldsymbol{x}_k) + \boldsymbol{g}^T \Delta \boldsymbol{x}_k . \tag{5.36}$$

Equation (5.36) can be rewritten as

$$\Delta f = \boldsymbol{g}_k^T \Delta \boldsymbol{x}_k , \tag{5.37}$$

that must be negative if the function *f(x)* is to decrease. Combining equations (5.35) and (5.37) therefore gives us

$$\boldsymbol{g}_k^T \Delta \boldsymbol{x}_k = \alpha_k \, \boldsymbol{g}_k^T \boldsymbol{p}_k , \tag{5.38}$$

where, for $\boldsymbol{p}_k$ to point in the direction of maximum change, or descent, we must have $\boldsymbol{p}_k = -\boldsymbol{g}_k$. This means that we can rewrite equation (5.34) as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \, \boldsymbol{g}_k , \tag{5.39}$$

which is called the steepest descent, or gradient descent method.

For an analytical function, $\boldsymbol{g}_k$ can be computed directly, as we saw in the last section. However, when we do not know the underlying analytic function, the gradient must be evaluated directly from the data. In the case where the vector $\boldsymbol{x}$ in equation (5.39) represents a set of unknown weights, as in both the single and multi-layer perceptron, we can replace the gradient with the error between the desired and predicted outputs. In this case, *the gradient descent method becomes identical to the LMS algorithm*, that was used in both chapter 4 and in this chapter to solve for the weights. A detailed discussion of the LMS algorithm is given in Appendix 5.

Another key decision is how to choose a value for α_k. Often, this value is set to a constant that does not vary from iteration to iteration. Choosing this value to be very small will ensure a smooth convergence but will lead to too many iterations. Choosing a value of alpha that is too large will result in fewer iterations, but the solution may become unstable.

As shown by Hagan et al. (1996) and discussed in section A5.6 of Appendix 5, the theoretical limit of α_k is

$$\alpha_k < \frac{2}{\lambda_{max}}, \tag{5.40}$$

where λ_{max} is the largest eigenvalue of the Hessian matrix.

If we are dealing with quadratic functions, we can compute an optimum value for α_k at each iteration by minimizing the function $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)$ with respect to α_k. As shown by Hagan et al. (1996) this leads to the vector result

$$\alpha_k = -\frac{\boldsymbol{g}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T H \boldsymbol{p}_k}. \tag{5.41}$$

As an example of the gradient descent method, let us use the function defined by equation (5.31), with gradient and Hessian defined in equations (5.32) and (5.33). Note that the eigenvalues for the matrix A are *3* and *1*, so that the largest allowable value for α is 0.667. We will choose $\boldsymbol{x}_0 = \begin{bmatrix} 1.0 \\ -0.4 \end{bmatrix}$, which means, from equation (5.32), that

$$\boldsymbol{g}_0 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1.0 \\ -0.4 \end{bmatrix} = \begin{bmatrix} 1.6 \\ 0.2 \end{bmatrix}.$$

If we choose $\alpha = 0.1$, then we find that

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - 0.1\,\boldsymbol{g}_0 = \begin{bmatrix} 0.84 \\ -0.42 \end{bmatrix},$$

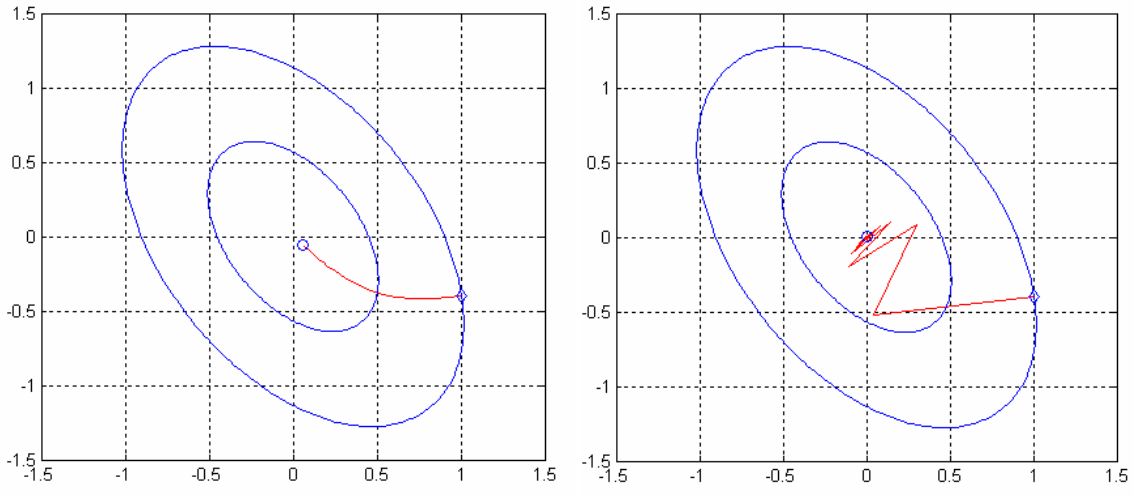
which is a small first step. If we choose $\alpha = 0.6$, then we find that

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - 0.6\,\boldsymbol{g}_0 = \begin{bmatrix} 0.04 \\ -0.52 \end{bmatrix},$$

which is a large first step.

Figure 5.17 shows the convergence of the solution for the two different values of α. As expected, using a small value of α requires more iterations than a large value of α.

In the cases shown in Figure 5.17, we needed *50* iterations for $\alpha = 0.1$ and *20* iterations for $\alpha = 0.6$. Note that the large value of α has created a result that is close to unstable.



(a) (b)

Figure 5.17. The convergence of the gradient descent method using α values of (a) 0.1 and (b) 0.6.

Next, we will let α_k vary for each iteration, using the line minimization procedure given in equation (5.41). For the first iteration, we find that

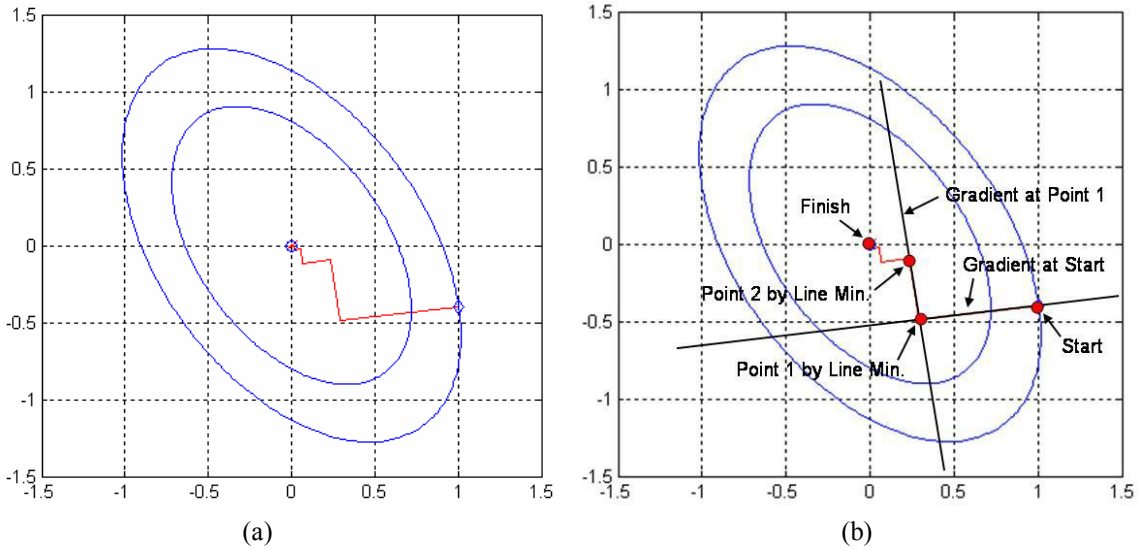
$$\alpha_0 = \frac{\boldsymbol{g}_0^T \boldsymbol{g}_0}{\boldsymbol{g}_0^T H \boldsymbol{g}_0} = \frac{\begin{bmatrix}1.6 & 0.2\end{bmatrix}\begin{bmatrix}1.6\\0.2\end{bmatrix}}{\begin{bmatrix}1.6 & 0.2\end{bmatrix}\begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}\begin{bmatrix}1.6\\0.2\end{bmatrix}} = 0.4452 .$$

This results in a first value equal to

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - 0.4452\,\boldsymbol{g}_0 = \begin{bmatrix}0.2877\\-0.489\end{bmatrix}$$

Figure 5.18 shows the first and subsequent values of the gradient descent algorithm with line minimization. Note that this method converges to the correct solution in an optimum fashion when compared to the results in Figure 5.18, in which the alpha values were both fixed and arbitrary. As shown in Figure 5.18(a), most of the convergence is in the first five steps, although ten steps are shown in the figure. Figure 5.18(b) shows a detailed interpretation of the first two iterations of Figure 5.18(a). Note

that at each step we move down the steepest slope (the gradient) from this step, but this direction is virtually never directly towards the actual minimum. Thus, the convergence to the solution is quite "jerky". The length of each step is controlled by the α factor. In this particular example, the values of α_k vary alternatively between *0.4452* and *0.5702*.



(a) (b)

Figure 5.18. The gradient descent algorithm with line minimization, where (a) shows the iterative solution and (b) explains the first two steps of (a) in more detail.

5.4.3 The conjugate gradient method

The conjugate gradient method allows us to converge to a solution much more rapidly than the gradient descent method, since it uses vectors that are mutually conjugate. Using the positive definite Hessian matrix defined in equation (5.27), we can state that a set of vectors $\boldsymbol{p_k}$ are mutually conjugate if

$$\boldsymbol{p}_k^T H \boldsymbol{p}_j = 0, \quad k \neq j . \tag{5.38}$$

If we then combine the definitions of the gradient and Hessian given for the quadratic function in equations (5.29) and (5.30), where we now write the Hessian as the specific matrix A, we can compute the change in the gradient as

$$\Delta \boldsymbol{g}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k = (A\boldsymbol{x}_{k+1} + \boldsymbol{b}) - (A\boldsymbol{x}_k + \boldsymbol{b}) = A\Delta \boldsymbol{x}_k . \tag{5.39}$$

If we next multiply equation (5.38) by α_k and substitute equation (5.33) for the term $\Delta \boldsymbol{x}_k$, we get

$$\alpha_k \boldsymbol{p}_k^T H \boldsymbol{p}_j = \Delta \boldsymbol{x}_k^T H \boldsymbol{p}_j = \Delta \boldsymbol{g}_k^T \boldsymbol{p}_j = 0, \quad k \neq j . \tag{5.40}$$

Equation (5.40) shows us that to compute the conjugate to the gradient we no longer need to know the Hessian, only the changes in the gradient from iteration to iteration. The procedure is done iteratively using the equation

$$\boldsymbol{p}_k = -\boldsymbol{g}_k + \beta_k \boldsymbol{p}_{k-1}, \tag{5.41}$$

where the first search direction uses the gradient descent method, so that $\boldsymbol{p}_0 = -\boldsymbol{g}_0$, and the β_k values are found using the equation

$$\beta_k = \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}} . \tag{5.42}$$

Equation (5.42) is due to Fletcher and Reeves (1964. The update in the x_k value is then given by equation (5.34).

Now I will illustrate the conjugate gradient method using the same dataset used to illustrate the gradient descent method. As just discussed, the first step of the conjugate gradient method is identical to the first step of the gradient descent method. Since I will again start at $\boldsymbol{x}_0 = \begin{bmatrix} 1.0 \\ -0.4 \end{bmatrix}$, the initial gradient is $\boldsymbol{g}_0 = \begin{bmatrix} 1.6 \\ 0.2 \end{bmatrix}$ and the first value is $\alpha_0 = 0.4452$. This means that the first step is exactly as shown in Figure 5.17, resulting in the position $\boldsymbol{x}_1 = \begin{bmatrix} 0.2877 \\ -0.489 \end{bmatrix}$.

The second step is the conjugate gradient step. For this, we need the second iteration of the gradient, that is

$$\boldsymbol{g}_1 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 0.2877 \\ -0.489 \end{bmatrix} = \begin{bmatrix} 0.0863 \\ -0.6904 \end{bmatrix}.$$

The first β term is then given by

$$\beta_1 = \frac{\boldsymbol{g}_1^T \boldsymbol{g}_1}{\boldsymbol{g}_0^T \boldsymbol{g}_0} = \frac{\begin{bmatrix} 0.0863 & -0.6904 \end{bmatrix} \begin{bmatrix} 0.0863 \\ -0.6904 \end{bmatrix}}{\begin{bmatrix} 1.6 & 0.2 \end{bmatrix} \begin{bmatrix} 1.6 \\ 0.2 \end{bmatrix}} = 0.1682,$$

Using equation (5.41) and recalling that $\boldsymbol{p}_0 = -\boldsymbol{g}_0$, this results in a conjugate gradient value of

$$\boldsymbol{p}_1 = \begin{bmatrix} -0.0863 \\ 0.6904 \end{bmatrix} + 0.1682 \begin{bmatrix} -1.6 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -0.3842 \\ 0.6532 \end{bmatrix}.$$

The new value of α is then given by line minimization as

$$\alpha_1 = -\frac{\boldsymbol{g}_1^T \boldsymbol{p}_1}{\boldsymbol{p}_1^T H \boldsymbol{p}_1} = -\frac{\begin{bmatrix} 0.0863 & -0.6904 \end{bmatrix} \begin{bmatrix} -0.3842 \\ 0.6532 \end{bmatrix}}{\begin{bmatrix} -0.3842 & 0.6532 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} -0.3842 \\ 0.6532 \end{bmatrix}} = 0.7487,$$

and the final result is

$$\boldsymbol{x}_2 = \boldsymbol{x}_1 + 0.7487\, \boldsymbol{p}_1 = \begin{bmatrix} 0.2877 \\ -0.489 \end{bmatrix} + \begin{bmatrix} -0.2877 \\ 0.489 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Thus, the conjugate gradient method has converged in just two iterations. This is illustrated in Figure 5.19, in which (a) shows the steps in the algorithm and (b) annotates these steps. Note that the conjugate to the gradient points directly at the correct solution and therefore the line minimization method allows us to arrive at the correct solution in just the second step. This can be contrasted with the gradient descent method of Figure 5.18, in which the gradient virtually never points directly at the correct solution and thus must tortuously arrive at the solution in multiple steps.

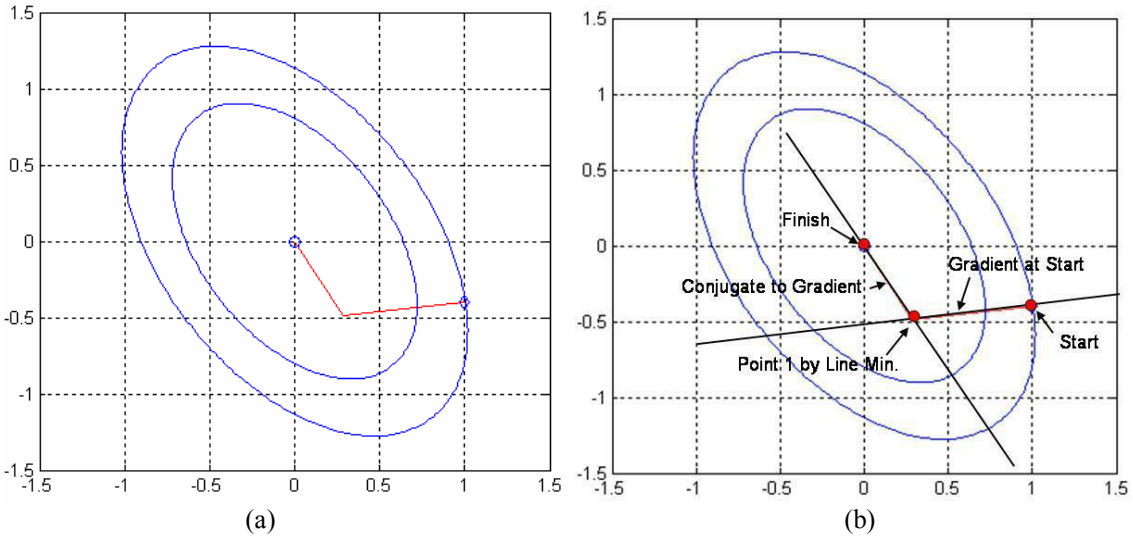


Figure 5.19. An illustration of the conjugate gradient algorithm with line minimization, where (a) shows the iterative solution and (b) shows an annotation of (a). Note that the algorithm converges in two steps.

The preceding analysis assumed that we are dealing with a quadratic function when performing minimization. In the case that will be considered in Section 5.5 we are dealing with real discrete data and the algorithm must be modified (Masters, 1993). The first difference is that the gradient term is not analytic and must be determined by the error between the predicted and desired result, as explained in the LMS algorithm. The second difference is that the distance step α cannot be determined by line minimization. In the real data case, this value is determined by searching for a minimum in the error term using a technique called the Golden Section search (Press et al., 1992). This technique will be also be used in finding the minimum for the radial basis function method, and will be discussed in the next chapter.

5.5 A multi-layer feedforward neural network case study

In this section, I will apply the theory of the multi-layer perceptron to the prediction of P-wave velocity in a channel sand from the Blackfoot oilfield of southern Alberta. This is the same area that I have discussed in sections 3.7 and 4.4.2. The base map showing both the wells in the area and the outline of the 3D seismic survey is shown in Figure 5.20.

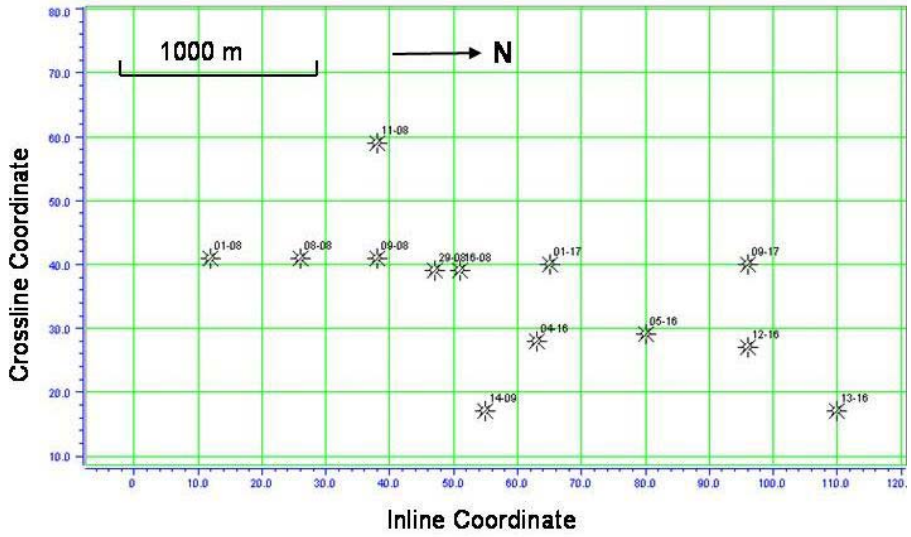


Figure 5.20. A base map showing the wells and limits of the 3D seismic data used in this study. The red vertical line shows line 95, that is displayed in the next figure.

Figure 5.21 shows seismic line 95, which was indicated as the vertical red line in Figure 5.20. The P-wave sonic log from line 95 has been superimposed on the section at its correct location, after correlation with the seismic data. The wiggle traces on the display represent seismic amplitude and the colour represents acoustic impedance derived using the model-based seismic inversion scheme discussed in section 2.8.2. Also shown is a picked seismic event (Horizon 1) just above the channel location. The low impedance red zone just below Horizon 1 shows the extent of the channel on this line. Notice that this is the same input data that was used in the case study of section 4.4.2. The two differences are that we will be predicting P-wave velocity rather than porosity, and performing regression rather than classification.

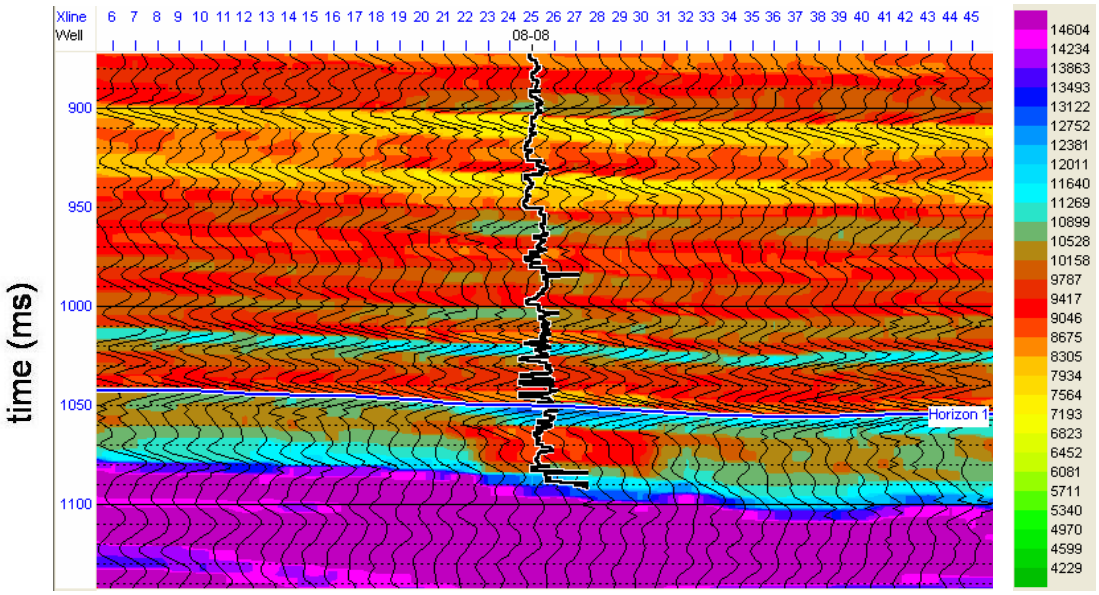


Figure 5.21. Line 95 from the 3D survey shown in Figure 5.23, where the wiggle traces show the seismic amplitudes, colour shows the inverted impedance, and the P-wave sonic log has been inserted at trace 25.

To see the full lateral extent of the channel, a dataslice was created over the complete 3D survey, and is shown in Figure 5.22. This slice was created by averaging over a 10 ms zone that was centered on a "phantom" event created by shifting Horizon 1 down by 20 ms. Notice the low impedances (yellow) that correspond to the channel in the lower part of the map, running from right to left.

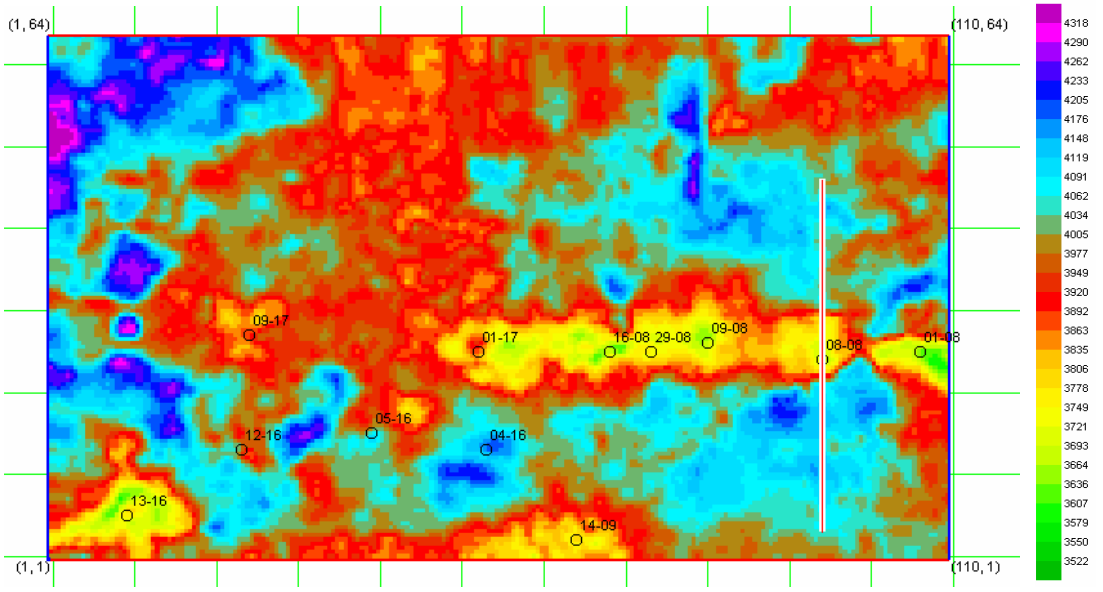


Figure 5.22. The average impedance from a 10 ms window that was shifted 20 ms below Horizon 1 in Figure 5.21.

We will next extract the seismic trace and inverted trace that correspond to each P-wave sonic log by averaging these traces from a one-trace radius around each well. Figure 5.23 shows the result at wells 08-08 and 09-08, with the sonic log on the right, the seismic trace in the middle, and the inverted trace on the right. The analysis window for each well is shown by the horizontal red lines.

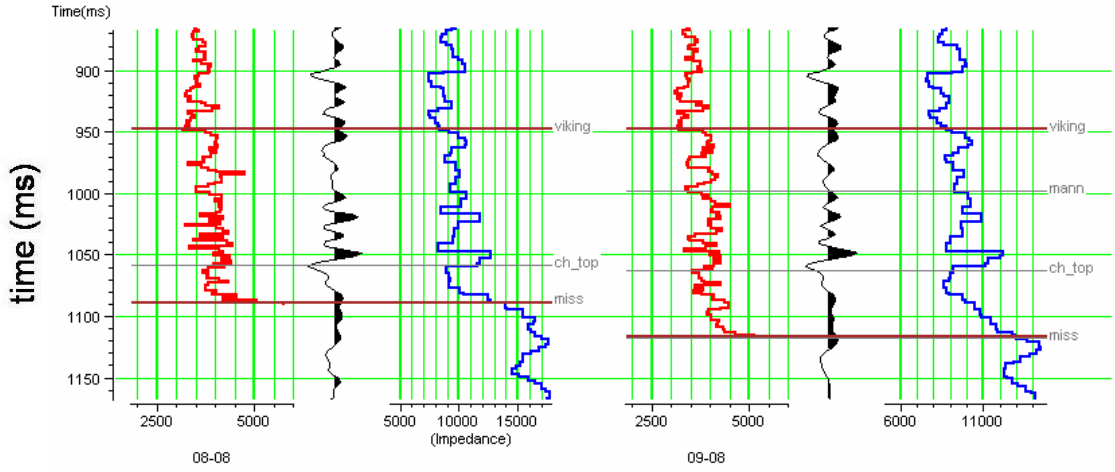
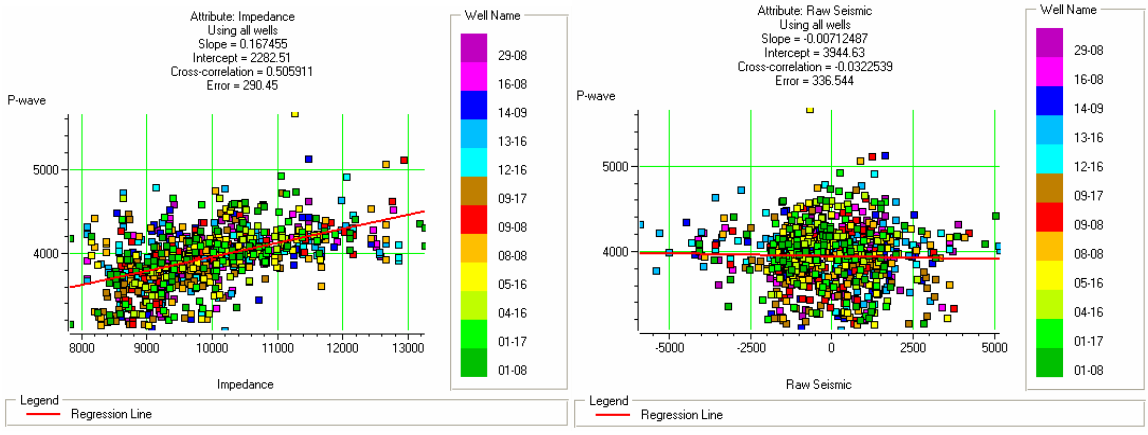


Figure 5.23. The P-wave sonic, extracted seismic trace and inverted impedance for wells 08-08 and 09-08. The zones of interest are marked by the horizontal red lines.

We can get an idea of how well the seismic trace and inverted trace match the sonic log over the zone of interest by cross-plotting them, as shown in Figure 5.24. It is obvious that the inverted trace fits better.



(a) (b)

Figure 5.24. The crossplots of (a) inverted impedance, and (b) seismic amplitude against the P-wave sonic logs over the zone of interest in all wells.

We will next find the optimum set of attributes using the method described in section 3.6. That is, we will find the best attributes using step-wise regression and decide which attributes to keep using cross-validation. Table 5.2 and Figure 5.25 show the result of this analysis, where the table shows the chosen attributes and the figure shows the training error in black and the validation error in red. Only the first six attributes will be used, since the validation error starts to increase after six. In this analysis, we also used an operator length of seven points, that was found to be the optimal length by trying a range of values.

.	Target	Final Attribute	Training Error	Validation Error
1	Sqrt(P-wave)	1 / (Impedance)	262.794115	265.392307
2	Sqrt(P-wave)	Filter 15/20-25/30	252.846073	256.424282
3	Sqrt(P-wave)	Filter 35/40-45/50	246.300437	251.932202
4	Sqrt(P-wave)	Amplitude Weighted Phase	243.163417	250.644314
5	Sqrt(P-wave)	Filter 55/60-65/70	240.339386	249.522596
6	Sqrt(P-wave)	Amplitude Weighted Frequency	237.576412	248.880681
7	Sqrt(P-wave)	Instantaneous Frequency	235.328710	249.518669

Table 5.2: The list of attributes used in the training for this problem.

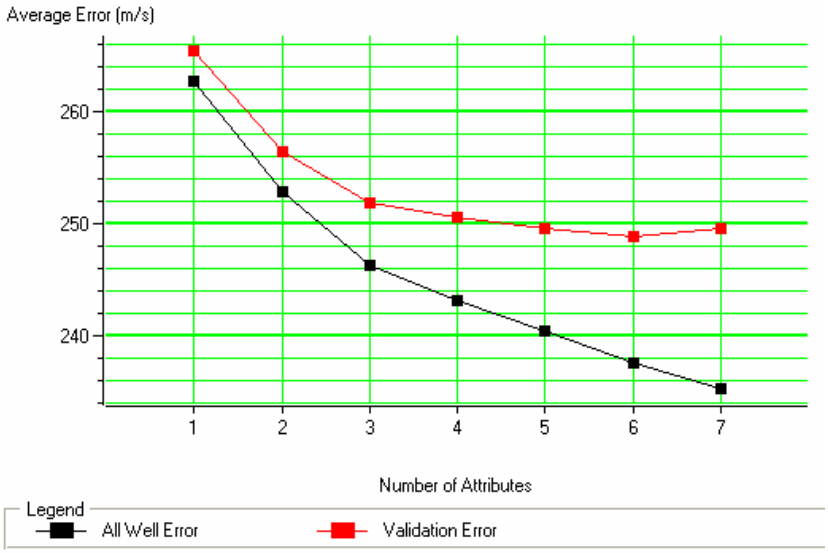
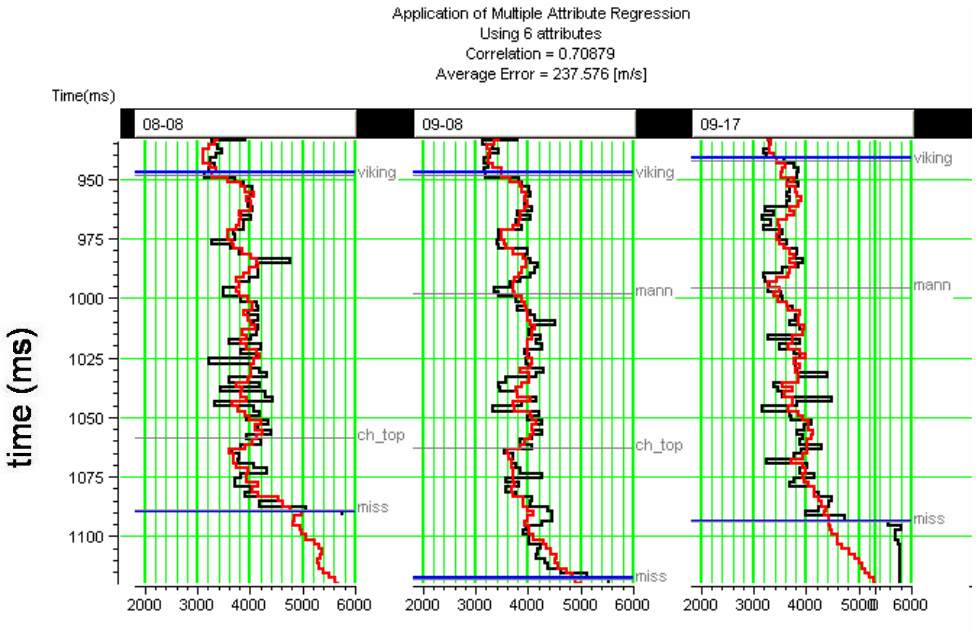
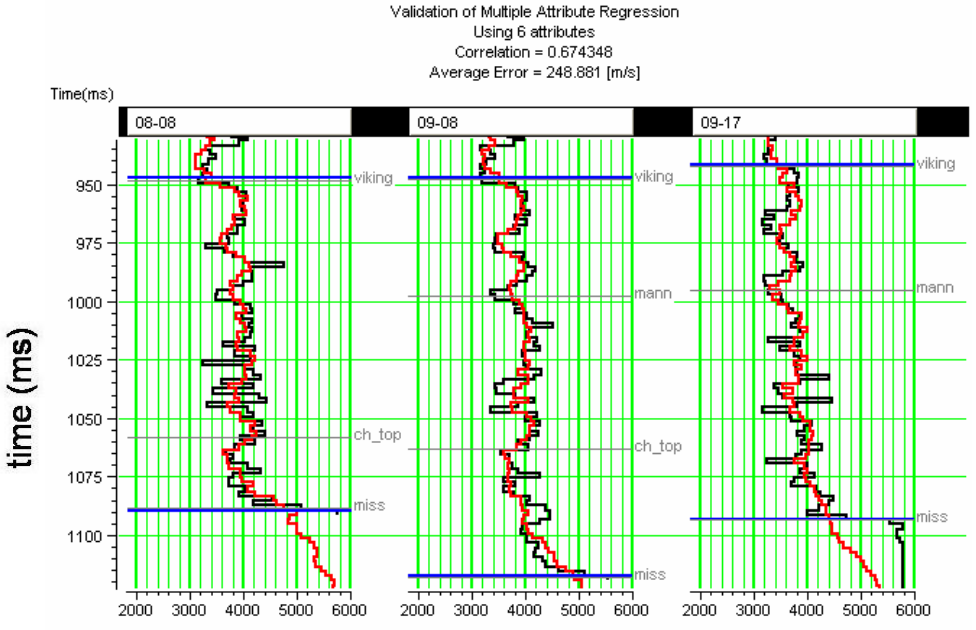


Figure 5.25. The graphical error for the attributes, where the red curve shows the validation error and the black curve shows the total error.

The comparison between the predicted and actual sonic log values at wells 08-08, 09-08 and 09-17 are shown in Figure 5.26.



(a)



(b)

Figure 5.26. A comparison of the predicted P-wave logs (red) to the actual P-wave logs (black), where (a) shows the training result and (b) shows the validation result.

In Figure 5.26, note that (a) shows the result of using all the wells and (b) shows the validation result, in which the well being predicted is left out of the training. Notice also that the correlation coefficient drops to 0.674 for the validation result, but this is still reasonably close to the training result of 0.709. Figure 5.27 shows a crossplot of the pseudo-sonic logs against the measured sonic logs at all well locations. When comparing this with Figure 5.24(a), notice that the scatter has gone down and the correlation coefficient has improved to 0.709.

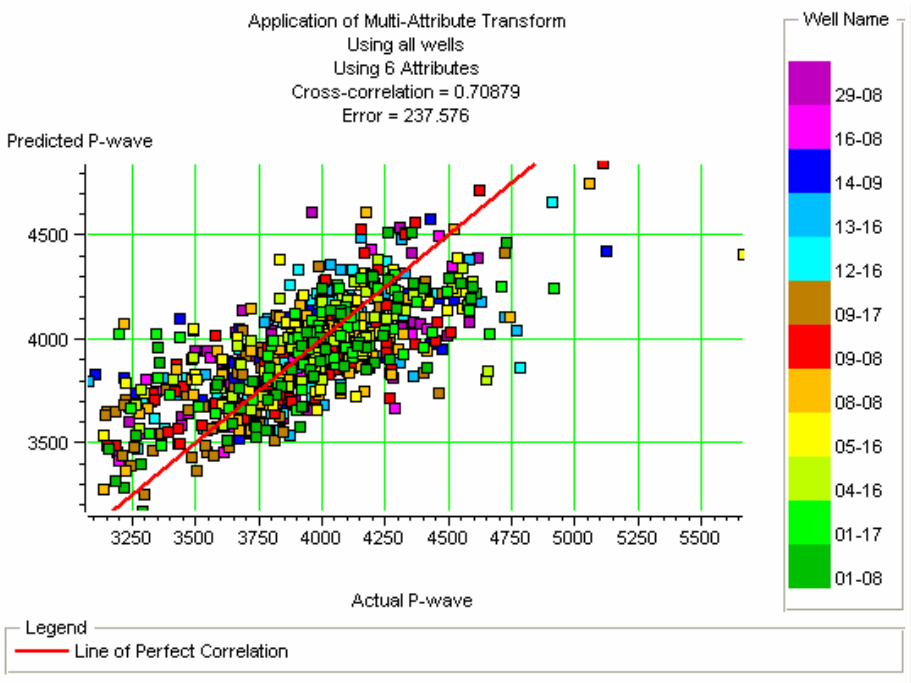


Figure 5.27. The crossplot of P-wave sonic log values predicted using the multilinear transform against true P-wave sonic log values, for all wells.

We will now apply the multilinear regression coefficients derived in the training to produce a full volume of pseudo-P-wave sonic logs. Figure 5.28 shows the predicted P-wave values for seismic line 95. Notice the extra detail that is seen in the channel that intersects the well log below Horizon 1 when compared to Figure 5.21. There is also better lateral continuity throughout the section.

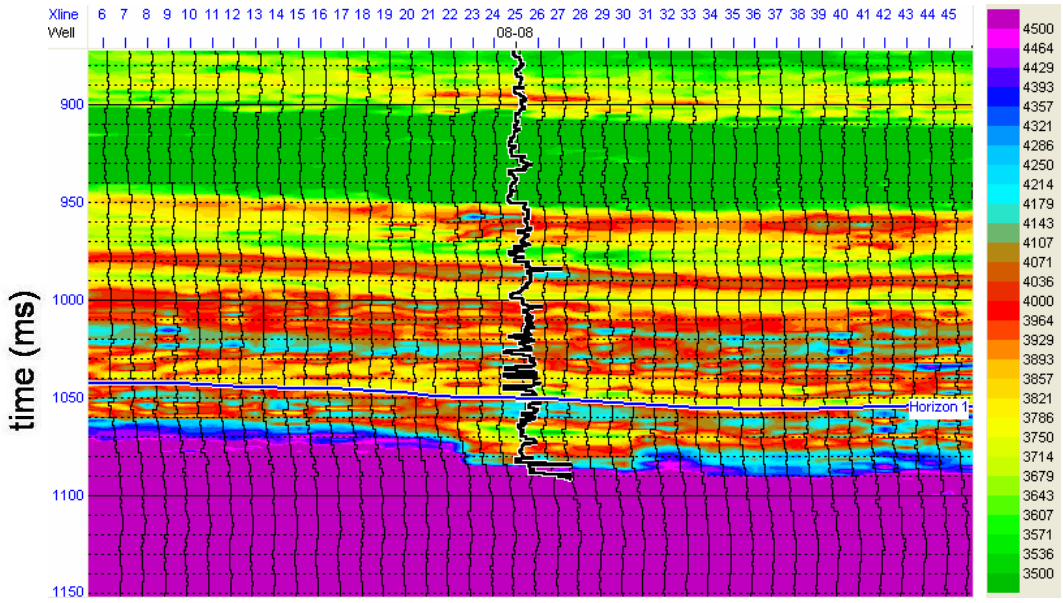


Figure 5.28. The predicted P-wave sonic logs from multilinear regression over line 95, taken from the predicted P-wave logs over the complete seismic volume.

Figure 5.29 shows a dataslice from the complete 3D survey, which is the mean average of a 10 ms zone that was centered on a "phantom" event created by shifting Horizon 1 down by 20 ms, as in Figure 5.22. As in the Figure 5.22, the grid cells were then interpolated by a factor of four and smoothed with a 3 x 3 running smoother. Notice that the channel is more clearly defined than the channel in Figure 5.22.

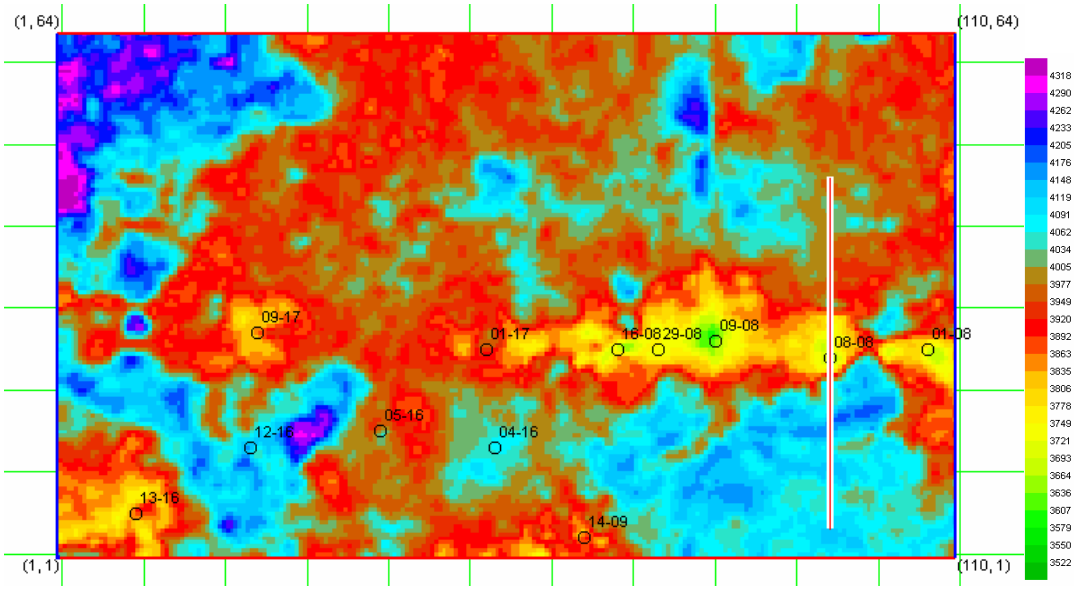


Figure 5.29. The extracted P-wave slice over the complete seismic survey, averaged over a 10 ms window that was shifted 20 ms below Horizon 1 in Figure 5.31.

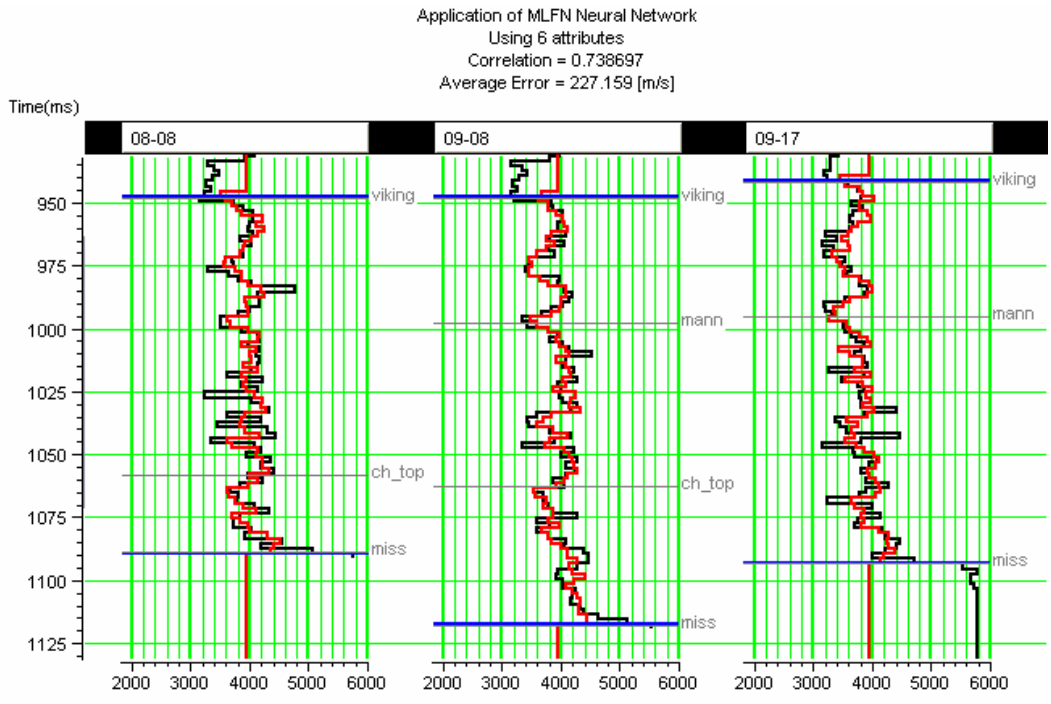
Finally, we will use the multi-layer perceptron (MLP) to try to improve the results of the multilinear regression. The theory of the MLP was discussed earlier in this chapter, but not the practical implementation. The three key questions are how to choose the attributes, how many nodes to include in the hidden layer and how many iterations to perform, both for the backward propagation step and the conjugate gradient step.

To answer the first question, I have found in this study that the multilinear regression approach gives a good estimate of the attribute order, so the attributes with the operator described above are the ones that will be used as input to the neural network. Note that the attributes were 6 in number with an operator length of 7 points. To answer the second question, a good rule-of-thumb for the number of nodes is to use *2/3* of the number of attributes (Hampson and Russell, 1998). Since the number of effective attributes is *6 x 7 = 42* (six attributes with a seven point operator), we will therefore use *28* nodes. Finally, for the number of iterations, we will use *10* full iterations and *100* conjugate gradient iterations within each full iteration. The simulated annealing steps are included if the error does not go down. In this case, no simulated annealing steps were needed.

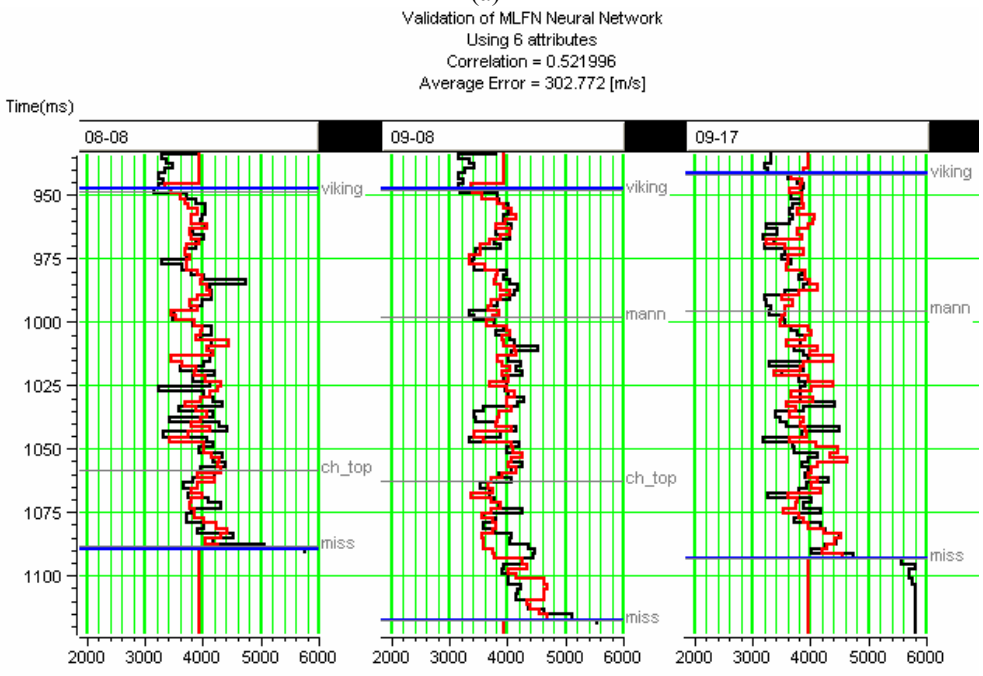
The comparison between the predicted and actual sonic log values after the MLP process, at wells 08-08, 09-08 and 09-17, are shown in Figure 5.30. Figure 5.30(a) shows the training result, in which all the wells are used in the calculation, and Figure 5.30(b) shows the validation result, in which the well being predicted is left out of the training calculation. Notice that the correlation coefficient for the training result is 0.739, which is better than multilinear regression, but that the correlation coefficient for the validation result drops to 0.522, which is worse. In other words, some of the improvement that we are seeing with the MLP may be due to overtraining at the well locations.

Figure 5.31 then shows a crossplot of the pseudo-sonic logs against the measured sonic logs at all well locations, after applying the MLP process. When comparing this with Figure 5.30, notice that the scatter has gone down even more and the correlation

coefficient has improved to 0.739. However, keep in mind that this is the result of full training, and not validation.



(a)



(b)

Figure 5.30. A comparison of the predicted P-wave logs (red) to the actual P-wave logs (black) from the MLP network, where (a) shows the training result and (b) shows the validation result.

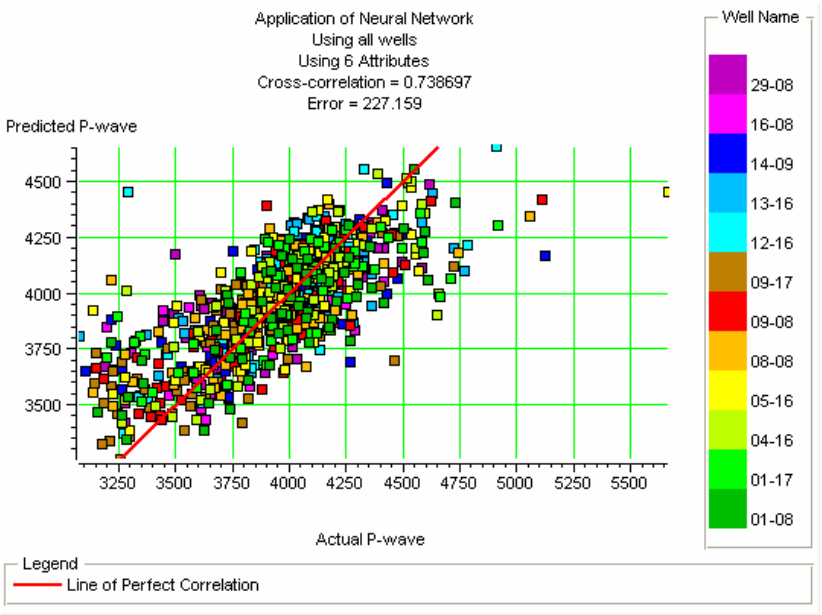


Figure 5.31. The crossplot of P-wave sonic log values predicted using the MLP network against true P-wave sonic log values, for all wells.

We will now apply the multilinear regression coefficients derived in the MLP training to produce a full volume of pseudo-P-wave sonic logs. Figure 5.32 shows the predicted P-wave velocity values for seismic line 95. Notice the extra detail that is seen in the channel that intersects the well log below Horizon 1, when compared to the multilinear regression result of Figure 5.28. There is also much higher frequency and good lateral continuity throughout the section. However, a strong "imprint" of the well log is clearly visible on the seismically-derived logs adjacent to the well, which may suggest that the result has been overtrained.

Figure 5.33 then shows a dataslice from the complete 3D survey, again showing the mean average over a 10 ms zone that was centered on a "phantom" event created by shifting Horizon 1 down by 20 ms. As in the Figures 5.22 and 5.29, the grid cells were then interpolated by a factor of four and smoothed with a 3 x 3 running smoother. Notice that the channel is less continuous than the channel in Figure 5.29, indicating again that

the network has possibly been overtrained. Thus, overtraining is a big concern for the multi-layer perceptron neural network, and appears in evidence here.

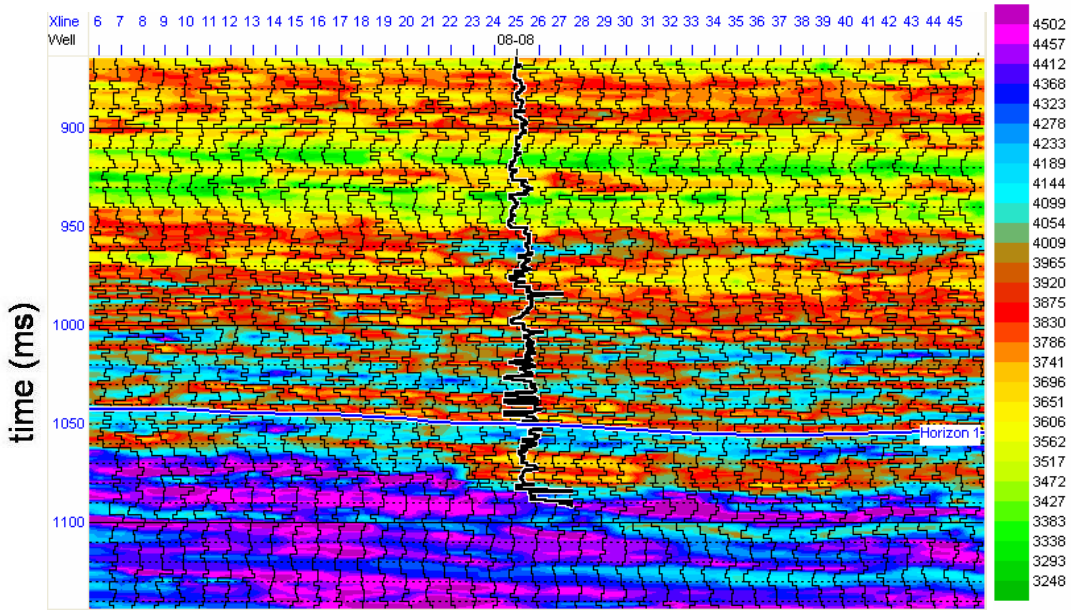


Figure 5.32. The predicted P-wave sonic logs from the MLP network over line 95, taken from the predicted P-wave logs over the complete seismic volume.

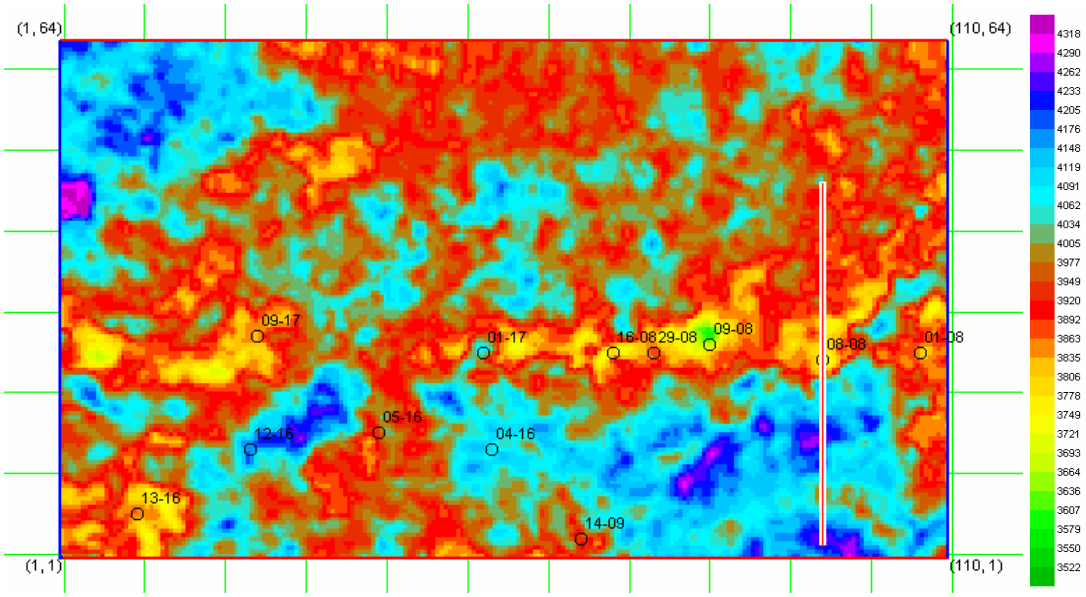


Figure 5.33. The extracted P-wave slice over the complete seismic survey, averaged over a 10 ms window that was shifted 20 ms below Horizon 1 in Figure 5.35.

This case study has shown that we can produce higher-frequency results with the multi-layer perceptron approach to reservoir prediction than with the multilinear regression approach. However, it has also shown us that the danger of overtraining has increased. A second problem, that was shown earlier in the chapter, is that if we had started the process with a different set of initial weights, the final answer could have been different. This is not the case with multilinear regression, which always converges to the same answer, given the same number of iterations. In the next chapter, we will look at a set of neural networks that combine the best features of both the multilinear regression approach and the multi-layer perceptron approach, called basis function neural networks. These methods will always converge to the same answer, but they are inherently nonlinear. Although the danger of overtraining will also be present using basis function networks, the danger is less severe than in the multi-layer perceptron.

5.6 Neural networks using a linear function

It is important to note that the multilayer network is equivalent to a single layer network if the linear function is applied to the output of the weighted sum at each layer. To understand this, consider a simple two-layer network with three inputs, two perceptrons in the first layer, and a single perceptron in the second layer, as illustrated in Figure 5.34.

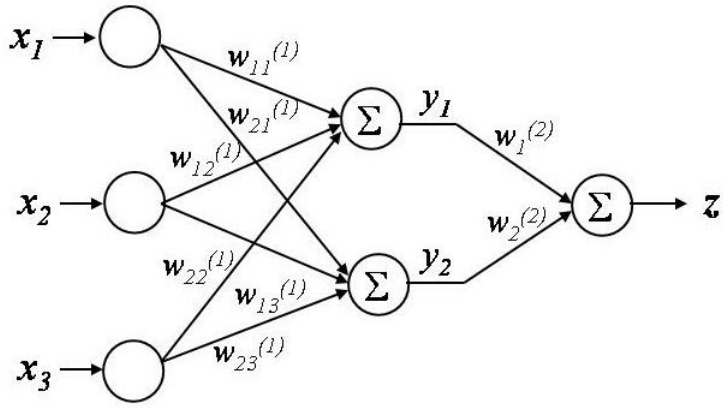


Figure 5.34. A multi-layer perceptron that uses a linear function.

If we let the bias equal zero, we can write

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{5.43}$$

for the output of the first layer, and

$$z = \begin{bmatrix} w_1^{(2)} & w_2^{(2)} & w_3^{(2)} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \tag{5.44}$$

for the output of the second layer, where we have assumed that $f(\mathbf{y}) = \mathbf{y}$ *and* $f(z) = z$. Combining equations (5.43) and (5.44) gives

$$z = \begin{bmatrix} w_1^* & w_2^* & w_3^* \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{5.45}$$

where

$$\begin{aligned} w_1^* &= w_1^{(2)} w_{11}^{(1)} + w_2^{(2)} w_{12}^{(1)}, \\ w_2^* &= w_1^{(2)} w_{21}^{(1)} + w_2^{(2)} w_{22}^{(1)}, \\ w_3^* &= w_1^{(2)} w_{31}^{(1)} + w_2^{(2)} w_{32}^{(1)}. \end{aligned}$$

Thus, the two layer network of Figure 5.37 reduces to the single layer network shown in Figure 5.35. This can be generalized to any number of layers and perceptrons.

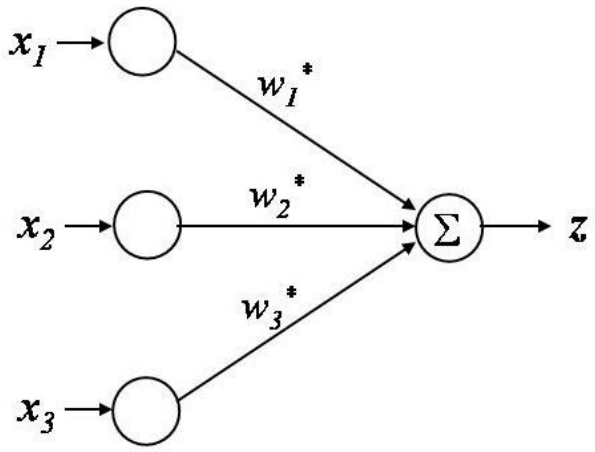


Figure 5.35. The equivalent single-layer perceptron to the multi-layer perceptron shown in Figure 5.34.

CHAPTER 6 : BASIS-FUNCTION NEURAL NETWORKS

6.1 Introduction

In all the approaches I have applied so far to the determination of reservoir parameters from seismic attributes, I have used some form of the scalar product of a weight vector and the input attribute vector. In the multilinear regression and linear discriminant methods, the output of this scalar product is the final result. In the multi-layer perceptron method, a nonlinear function is applied to the scalar product, and we are able to solve nonlinear problems. The disadvantage of linear methods is that they can solve only linear problems, whereas the disadvantage of the multi-layer perceptron approach is that the final answer is dependent on this initial guess of the weights.

In section 4.7 I introduced the generalized linear discriminant, in which we compute the scalar product of the weight vector and some nonlinear function of the input attribute vector. This equation is written

$$y = \mathbf{w}^T \phi(\mathbf{x}), \tag{6.1}$$

where $\phi(\mathbf{x})^T = [\phi(x_0) \quad \phi(x_1) \quad \cdots \quad \phi(x_M)]$ is a nonlinear function of the vectors $\mathbf{x}$, often called a basis function. In this chapter I will revisit the basis function and show how this approach combines the advantages of the methods we have discussed so far, but avoids the disadvantages. As with linear methods, the solution to the weights does not depend on an initial guess, but, unlike linear methods, the basis function approach can solve nonlinear problems. The neural network methods discussed in this chapter will include the probabilistic neural network, or PNN, the generalized regression neural network, or GRNN, and the radial basis function neural network, or RBFN.

6.2 Probability density estimation

6.2.1 Parametric statistics

In parametric statistics, we use a model that is based on several known parameters. This approach was extensively discussed in Chapter 3, in which we used the normal, or Gaussian, probability density function. In the general case for an *M*-dimensional vector $\boldsymbol{x}$, the multivariate normal distribution is given as

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{M/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right], \tag{6.2}$$

where $\boldsymbol{\mu}$ is the *M*-dimensional vector of means and $\boldsymbol{\Sigma}$ is and *MxM* dimensional covariance matrix. These parameters can be estimated by finding the mean, variance, and covariance of a representative set of data points. Parametric statistics give us an excellent way of visualizing statistical relationships within our data, and lead to such powerful techniques as linear discriminant analysis, Bayesian inference and maximum likelihood analysis. However, the drawback of parametric methods is that we "force" a model onto our data. In the next section, I will discuss non-parametric statistics, in which the form of the probability density function depends on the data itself.

6.2.2 Non-parametric statistics and the histogram

In non-parametric statistics we do not specify the parameters in advance, but instead try to fit a function based on the data itself. The simplest type of non-parametric approach, the histogram, was introduced in Chapter 3. In Figure 3.2, the histograms of four different attributes were shown. Implicit in the histograms shown in Figure 3.2 is the fact that each histogram is based on 10 uniform divisions, or bins, between the minimum and maximum amplitude of the attribute. If we change the bin size, the look of our histogram will change. This is shown in Figure 6.1, where I have re-computed the histogram of Figure 3.2(d) for 5, 10, 50, and 100 bins, respectively.

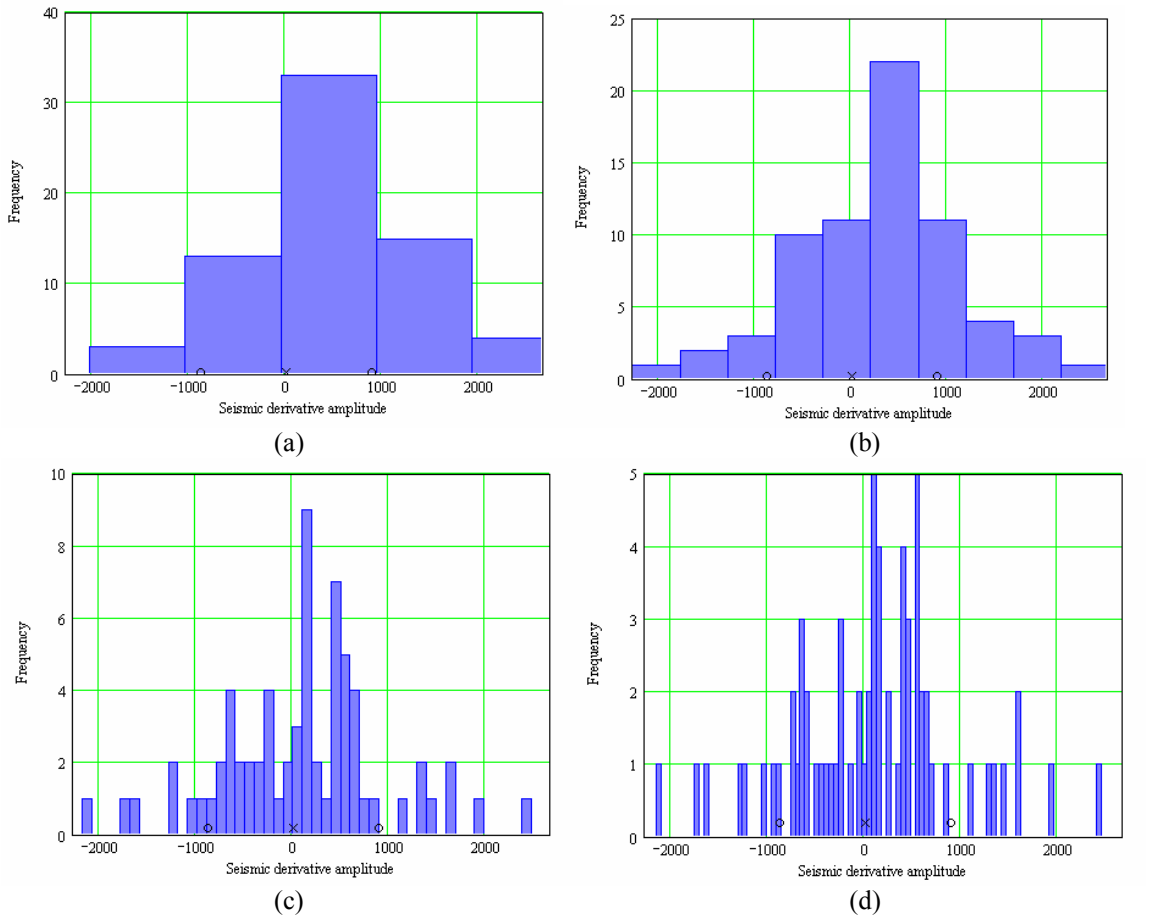


Figure 6.1. Histograms for Attribute 3 of Figure 3.2, for (a) 5, (b) 10, (c) 50, and (d) 100 bins, respectively.

Notice the extra detail that becomes apparent in Figure 6.1 as we increase the number of bins. Recall that this dataset contained 69 points, so a bin size of 100 is actually too large and contains a lot of zeros and ones.

Although the histogram gives us an easy way of visualizing the distribution of values within our dataset, it has several limitations (Bishop, 1995). First, the function is not smooth, but has discontinuities at the bin locations. Second, and more importantly, the histogram cannot be easily extended to the multi-dimensional case. In the next section, we will discuss a density estimation method that overcomes both of these limitations.

6.2.3 Kernel-based density estimation

In kernel-based density estimation we define a hypercube in M dimensions and then define a kernel function that will map the points which fall within this hypercube into a new space defined by the kernel function. In our application, I will again consider the case of our input attribute vectors $\boldsymbol{x}_j$, $j = 1, \ldots, N$, where each vector contains M attributes and can be written $\boldsymbol{x}_j^T = \begin{bmatrix} x_{1j} & x_{2j} & \cdots & x_{Mj} \end{bmatrix}$. These vectors can be thought of as N points in M-dimensional attribute space. If we consider an arbitrary point in this space, $\boldsymbol{x}$, we can define Parzen's estimator (Parzen, 1960) by the function

$$p(\boldsymbol{x}) = \frac{1}{N\sigma} \sum_{j=1}^{N} \phi\left(\frac{\boldsymbol{x} - \boldsymbol{x}_j}{\sigma} \right), \tag{6.3}$$

where $\phi(\boldsymbol{u})$ is referred to as either the kernel function or the Parzen window, and satisfies the properties

$$\phi(\boldsymbol{u}) \geq 0, \tag{6.4a}$$

and

$$\int \phi(\boldsymbol{u}) d\boldsymbol{u} = 1. \tag{6.4b}$$

Although there are a number of functions that conform to these criteria, the most commonly used function is the Gaussian kernel, given by

$$p(\boldsymbol{x}) = \frac{1}{N(2\pi)^{M/2}\sigma^M} \sum_{j=1}^{N} \exp\left(-\frac{\left| \boldsymbol{x} - \boldsymbol{x}_j \right|^2}{\sigma^2} \right), \tag{6.5}$$

I will exclusively use the Gaussian kernel as our Parzen estimator, and also for the neural networks based on this method.

The key parameter in equation (6.5) is the scaling factor σ, which can obviously be equated to the standard deviation in the parametric form of the Gaussian distribution. Let us first consider the one-dimensional case, where the Parzen window can be written

$$p(x) = \frac{1}{N\sigma\sqrt{2\pi}} \sum_{j=1}^{N} \exp\left(-\frac{(x - x_j)^2}{\sigma^2} \right), \tag{6.6}$$

where x is a scalar value. Figure 6.2 shows the result of applying the Parzen estimator to the dataset shown in the histograms of Figure 6.1. In this case, I have used σ values of 10, 100, 250, and 1000, respectively. The reason for the large values of σ is that I did not normalize the dataset first. Recall also that $N = 69$ for this dataset.

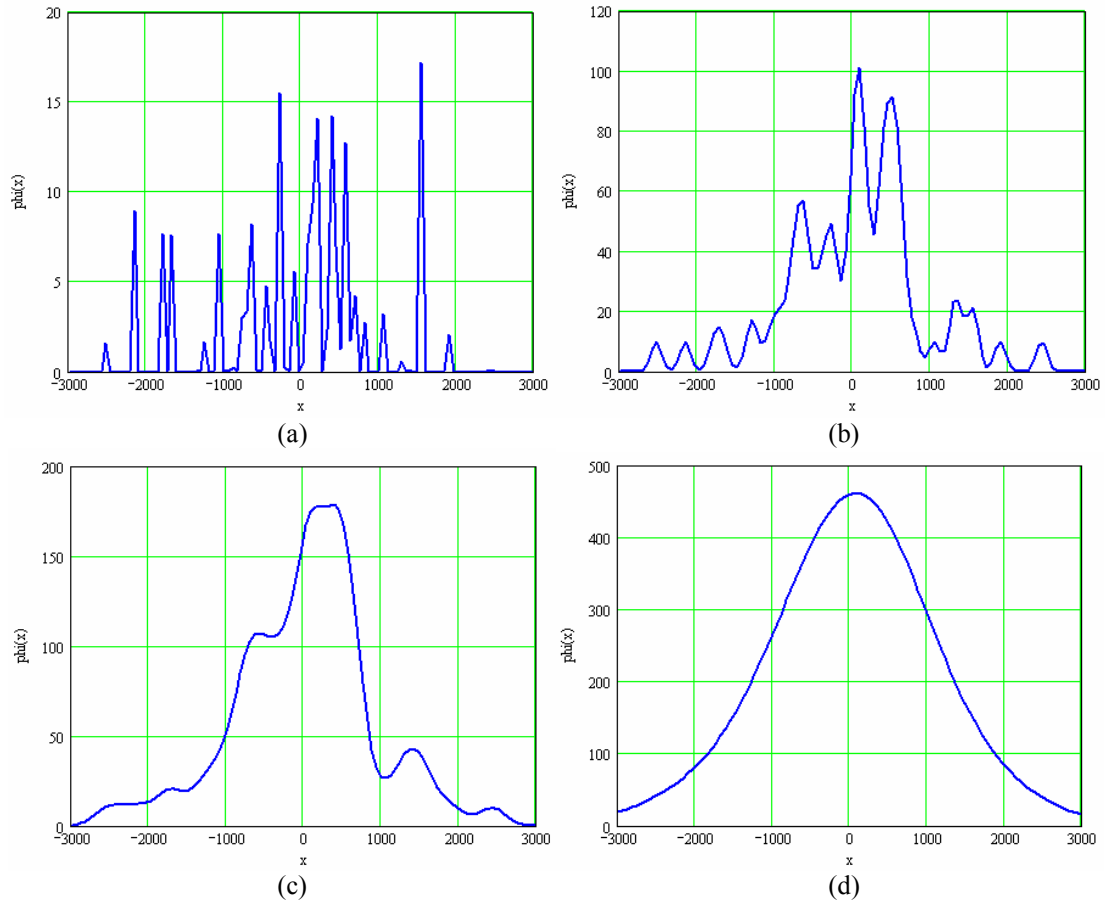


Figure 6.2. The Parzen estimator applied to the dataset shown in Figure 6.1, where I have used σ values of (a) 10, (b) 100, (c) 250, and (d) 1000.

From Figure 6.2 it is obvious that, as σ increases, the resulting function becomes smoother. It is also clear that a σ value of 10 is probably too small, and a σ value of 1000 is too large. To illustrate the effect of σ on the Parzen window even more clearly, I

will let $N = 6$, so that the individual Gaussian curves can be seen. I will let the six points be $x_1 = 20$, $x_2 = 30$, $x_3 = 35$, $x_4 = 60$, $x_5 = 70$, and $x_6 = 75$. The result of applying a Parzen window to these values is shown in Figure 6.3, where I have used σ values of *5, 10, 20,* and *40*, respectively. Again, it is the relative effect of these values that we are concerned with, not their absolute values.

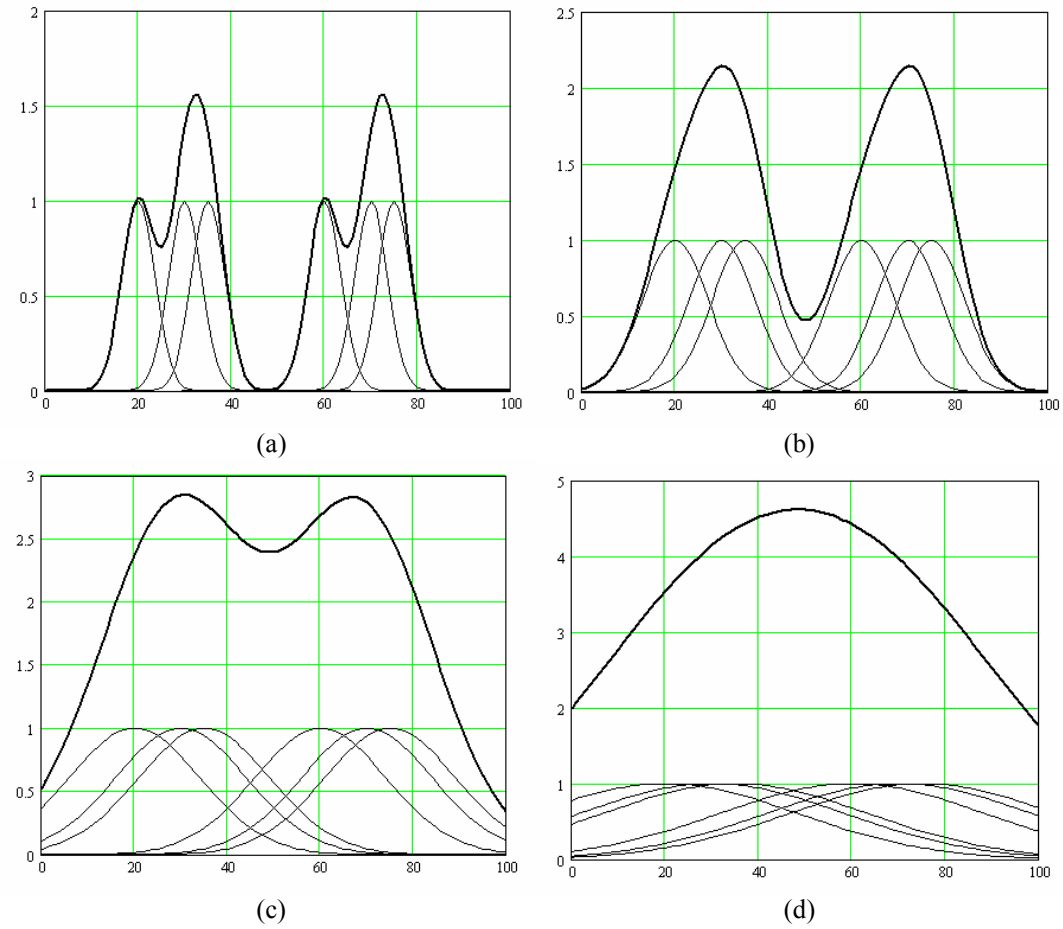


Figure 6.3: The 1D Parzen windows for the six points described in the text, where I have used σ values of (a) 5, (b) 10, (c) 20, and (d) 40.

In Figure 6.3, each Gaussian has been normalized to a value of one. Notice that as the value of σ increases, the resolution of the six points becomes less clear until, for a value of 40, we just see a single smooth Gaussian shape. Let us now consider the two-dimensional case, where the Parzen window can be written as

$$p(\boldsymbol{x}) = \frac{1}{12\pi\sigma^2} \sum_{j=1}^{6} \exp\left(-\frac{(x_1 - x_{1j})^2 + (x_2 - x_{2j})^2}{\sigma^2} \right). \qquad (6.7)$$

Figure 6.4 shows the Parzen window for the 2D case, where I have used the six points $\boldsymbol{x}_1 = \begin{bmatrix} 20 \\ 55 \end{bmatrix}$, $\boldsymbol{x}_2 = \begin{bmatrix} 30 \\ 75 \end{bmatrix}$, $\boldsymbol{x}_3 = \begin{bmatrix} 35 \\ 65 \end{bmatrix}$, $\boldsymbol{x}_4 = \begin{bmatrix} 60 \\ 40 \end{bmatrix}$, $\boldsymbol{x}_5 = \begin{bmatrix} 70 \\ 20 \end{bmatrix}$, $\boldsymbol{x}_6 = \begin{bmatrix} 75 \\ 30 \end{bmatrix}$, and have again used σ values of 5, 10, 20, and 40.

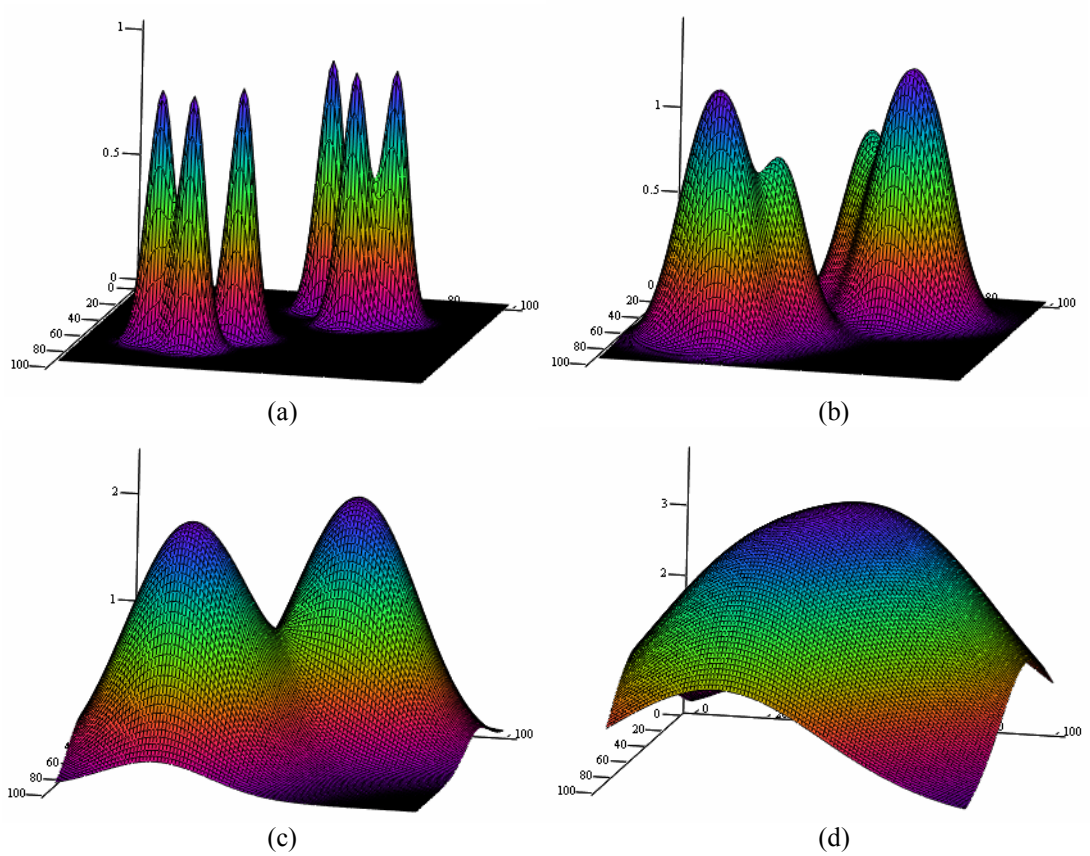


Figure 6.4: The 2D Parzen windows for the six points described in the text, where I have used σ values of (a) 5, (b) 10, (c) 20, and (d) 40.

Notice in Figure 6.4 that a σ value of 5 is too small, since the result is spiky, whereas a σ value of 40 is too large, since it results in a single smooth Gaussian shape. The optimum value of σ is between these two values. Although we can only visualize the Parzen window up to the two-dimensional case, the same concept applies for $M > 2$.

6.3 An introduction to kernel-based neural networks

Let us now discuss neural networks that are based on the Parzen estimator. As already discussed, our training dataset consists of a set of N known training samples t_i, which will be some well-log-derived reservoir parameter such as V_P, SP, S_W, etc. Each training sample is dependent on a vector of M seismic attribute values, correlated in time with the training samples. The issues of which seismic attributes to use and how to optimize the correlation between the training samples and the seismic data were discussed in Chapter 3, and I will be using the same approach in this section. The seismic attribute vectors can be written $\boldsymbol{s}_i = (s_{i1}, s_{i2}, \ldots, s_{iM})^T$, $i = 1, 2, \ldots, N$. The objective of our neural network is to find some function y such that:

$$y(\boldsymbol{s}_i) = t_i, \quad i = 1, 2, \ldots, N \tag{6.8}$$

Once this function has been found, it can be applied to an arbitrary set of seismic attribute vectors $\boldsymbol{x}_k$, where the attributes in the $\boldsymbol{x}_k$ vectors are identical to those in the $\boldsymbol{s}_i$ vectors. This is illustrated in Figure 6.5 for two arbitrary training samples and a single application sample. I have chosen to use a different letter to represent the training and application data vectors to emphasize the fundamental difference between them.

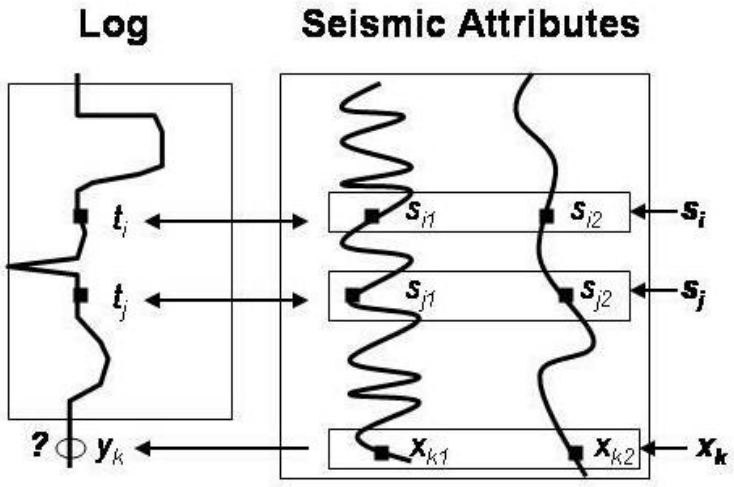


Figure 6.5: An illustration of the differences between the training vectors, $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$, in which the output samples t_i and t_j are known, and the application vector $\boldsymbol{x}_k$, in which the output sample y_k is not known.

Before discussing the RBFN and GRNN algorithms, I will first discuss the simpler PNN algorithm. The PNN algorithm is based on the concept of "distance" in attribute space. To better understand this concept, consider Figure 6.6, in which I have drawn, in graphical form, the three arbitrary two-dimensional seismic attribute vectors shown in Figure 6.5. Note that "distance" on these graphs is represented by the distance between attribute amplitudes rather than the Cartesian distance that we normally use. Recall that two of these vectors are from the training dataset ($\boldsymbol{s}_i$ and $\boldsymbol{s}_j$) and one is from the application dataset ($\boldsymbol{x}_k$). We can define the three possible distances between these vectors, as displayed in Figure 6.6. These are given by

$$d_{ij} = \left|\boldsymbol{s}_i - \boldsymbol{s}_j\right| = \sqrt{(s_{i1} - s_{j1})^2 + (s_{i2} - s_{j2})^2} ,$$

$$d_{ik} = \left|\boldsymbol{s}_i - \boldsymbol{x}_k\right| = \sqrt{s_{i1} - x_{k1})^2 + (s_{i2} - x_{k2})^2} , \text{ and}$$

$$d_{jk} = \left|\boldsymbol{s}_j - \boldsymbol{x}_k\right| = \sqrt{(s_{j1} - x_{k1})^2 + (s_{j2} - x_{k2})^2} .$$

It is important to distinguish two fundamentally different types of attribute distance in the above equations. The d_{ij} distances are the *inter-training* distances and the d_{ik} and d_{jk} distances are the *application* distances.

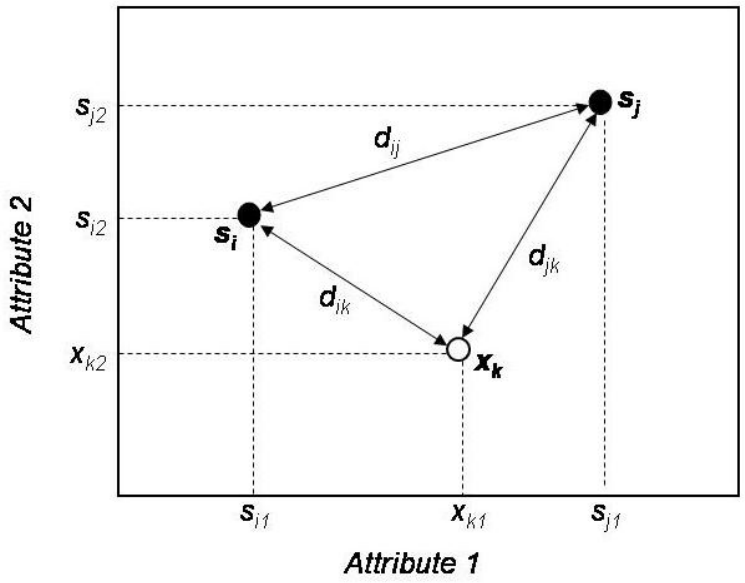


Figure 6.6: A schematic graph of the vectors, $\boldsymbol{s}_i$, $\boldsymbol{s}_j$, and $\boldsymbol{x}_k$, from Figure 6.5, where the coordinate axes represent attribute amplitude rather than Cartesian distance.

A second important concept is that it is not the distances themselves that will be used in our neural network applications, but the basis function, $\phi(d)$, of the distance. As mentioned in the last section, I will be using the Gaussian function as our basis function, which can be written as

$$\phi(d) = \exp\left[-\frac{d^2}{\sigma^2}\right], \tag{6.9}$$

where σ is a smoothness parameter. Notice that σ can also be interpreted as the variance of a Gaussian distribution centered on d. Thus, as we decrease σ, the width of the distribution becomes narrower. Equation (6.9) is obviously equivalent to the Parzen window of equation (6.5), but with the physical interpretation that the distance values used in the exponential are equivalent to distances between reservoir parameter. Which distances are used, and how they are used, will define our three neural networks: PNN, GRNN, and RBFN. I will start with the PNN or probabilistic neural network.

6.4 The probabilistic neural network

6.4.1 Theory of the probabilistic neural network

The probabilistic neural network, or PNN, is a neural network implementation of the Parzen window, and was initially proposed by Specht (1990). A description of this method, and the C++ code used to implement the algorithm, is found in books by Masters (1993, 1995). The PNN can be used for discrimination or classification, and is thus a nonlinear extension of Fisher's linear discriminant function. Based on the definitions given in the last section and shown in Figure 6.6 and 6.7, the PNN is defined for each of the $\boldsymbol{x_k}$ points as the sum over all the possible $\phi(d_{kj})$ functions, or

$$p(\boldsymbol{x}_k) = \sum_{j=1}^{N} \exp\left[-\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right] = \sum_{j=1}^{N} \phi_{kj}\ , \tag{6.10}$$

where I have used the abbreviation ϕ_{kj} for $\phi(d_{kj})$. If we use all the points in the training dataset PNN will result in a single value, which does not give us a very useful discrimination technique. But if we subdivide the training points into a number of classes, PNN becomes an excellent classification method. This can be thought of as an implementation of Bayes' Theorem. I will first consider the simplest case, that of two classes. If we have class C_1 with N_1 points, and class C_2 with N_2 points, where $N_1 + N_2 = N$, then we can define

$$p_1(\boldsymbol{x}_k) = \frac{\sum_{j \in N_1} \phi_{kj}}{p(\boldsymbol{x}_k)}, \tag{6.11}$$

and

$$p_2(\boldsymbol{x}_k) = \frac{\sum_{j \in N_2} \phi_{kj}}{p(\boldsymbol{x}_k)}, \tag{6.12}$$

where normalization by the $p(\boldsymbol{x}_k)$ term defined in equation (6.10) assures us that $p_1(\boldsymbol{x}_k) + p_2(\boldsymbol{x}_k) = 1.0$. The p_j values can be interpreted as the probability of membership in a class. That is, if $p_1(\boldsymbol{x}_k) > p_2(\boldsymbol{x}_k)$, *then* $\boldsymbol{x}_k$ *is a member of class* C_1, or, if $p_1(\boldsymbol{x}_k) < p_2(\boldsymbol{x}_k)$, *then* $\boldsymbol{x}_k$ *is a member of class* C_2.

Let us start with an example in which we have two classes, each a function of two attributes and containing three points. In this case, we can compute the total PNN function for point $\boldsymbol{x}_k$ as the combination of the basis functions from all 6 points, or

$$\begin{aligned} p(\boldsymbol{x}_k) &= \sum_{j=1}^{6} \exp\left[-\frac{\left| \boldsymbol{x}_k - \boldsymbol{s}_j \right|^2}{\sigma^2} \right] \\ &= \exp\left[\frac{(x_{1k} - s_{11})^2 + (x_{2k} - s_{21})^2}{\sigma^2} \right] + \cdots + \exp\left[\frac{(x_{1k} - s_{16})^2 + (x_{2k} - s_{26})^2}{\sigma^2} \right] \end{aligned} \tag{6.13}$$

Notice from equation (6.13) that the final result is the sum of six individual Gaussian functions.

We can compute the individual class probability functions as

$$p_1(\boldsymbol{x}_k) = \frac{\sum_{j=1}^{3} \exp\left[-\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]}{p(\boldsymbol{x}_k)}, \tag{6.14}$$

and

$$p_2(\boldsymbol{x}_k) = \frac{\sum_{j=4}^{6} \exp\left[-\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]}{p(\boldsymbol{x}_k)}. \tag{6.15}$$

This two class problem is shown below in Figure 6.7.

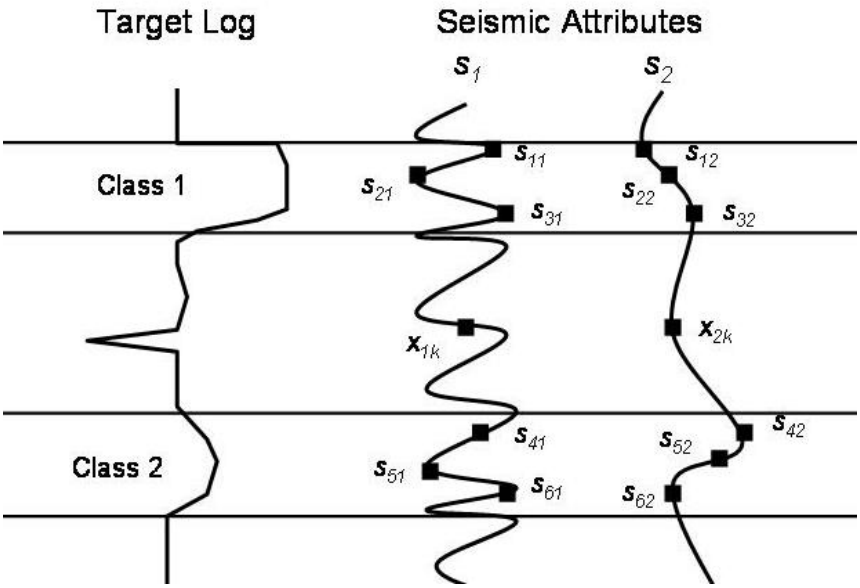


Figure 6.7: A simple example of the PNN neural network, using two classes with three points in each class, and two attributes.

The result of the calculations is shown in Figure 6.8 using an optimal sigma value. The top two figures show the un-normalized basis functions, and the bottom two figures show the normalized basis functions that correspond to the probabilities given in equation (6.14) and (6.15).

In Figure 6.8, classes 1 and 2 have been renamed classes *A* and *B*, respectively. This simple example can be generalized to *K* classes and *M* attributes, as will be done in the next section in which I will show a case study that uses 3 classes and four attributes.

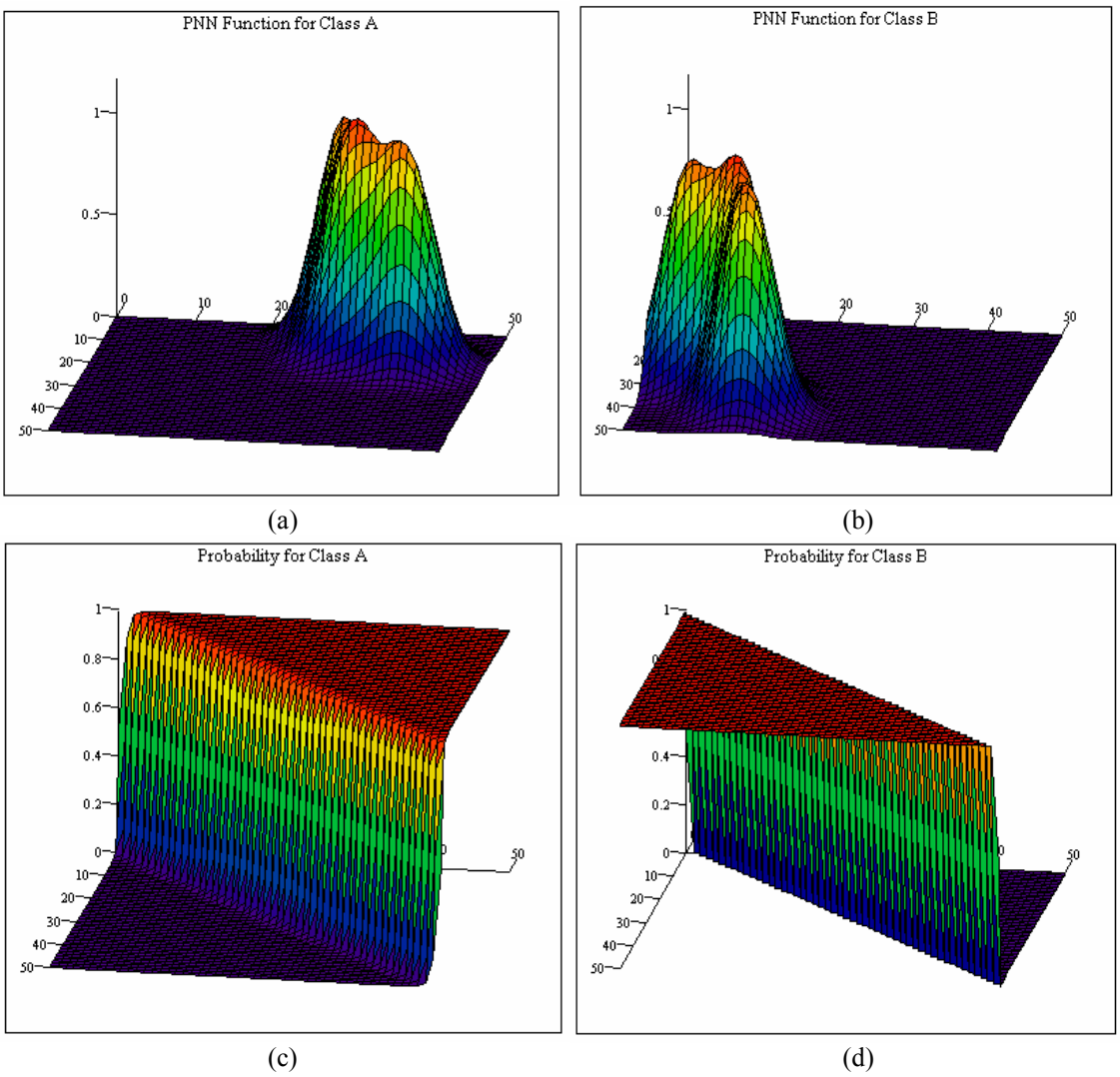


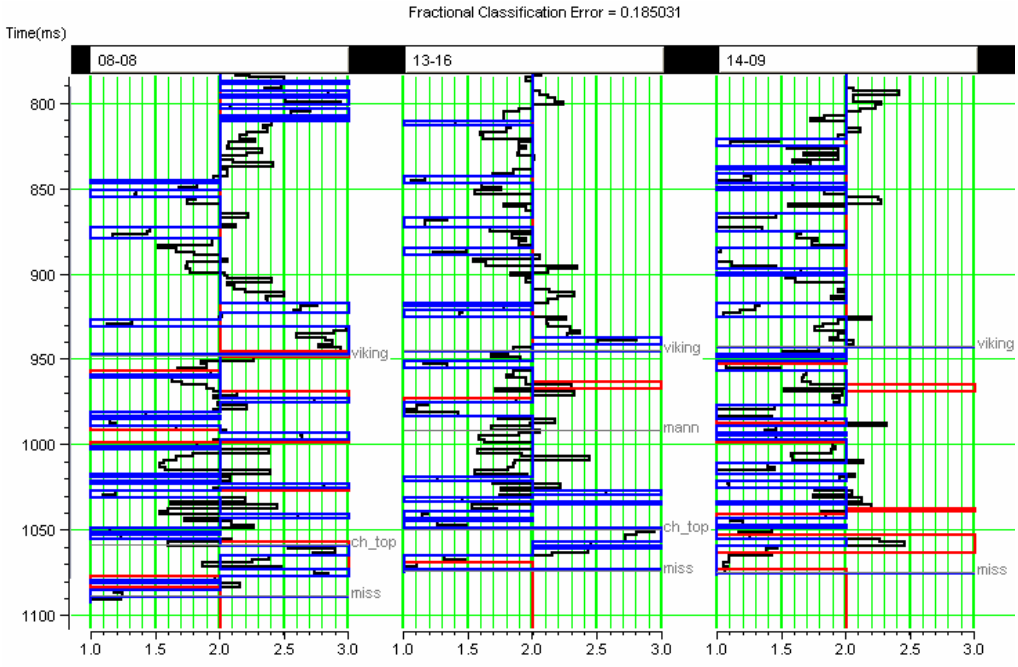
Figure 6.8: The computed basis functions for the example shown in Figure 6.7, where (a) and (b) represent the un-normalized functions for the two classes, and (c) and (d) represent the normalized probability functions.

6.4.2 Application of the probabilistic neural network to porosity classification

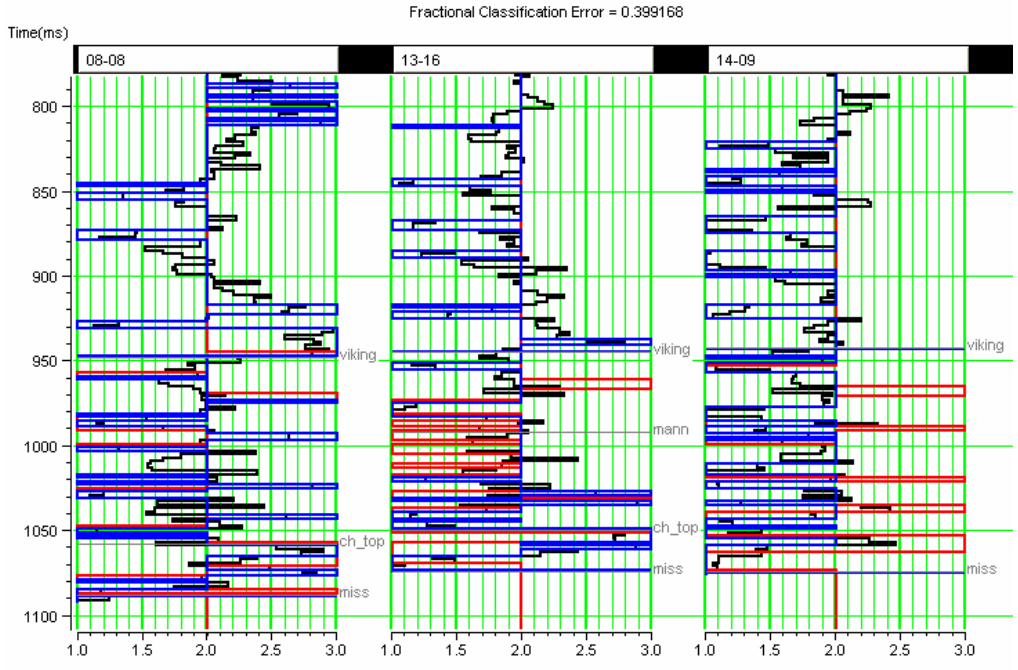
In section 4.4.2, I applied the Fisher linear discriminant to porosity classification. I will now use the same input data and apply the probabilistic neural network to classification. The base map for the study was shown in Figure 4.3, the logs for well 08-08 were shown in Figure 4.4, seismic line 95 was shown in Figure 4.5, and three of the seven wells used in the study were shown Figure 4.6. We then decided on which attributes to use for the classification of porosity using the seismic data, finding the best attributes using step-wise regression and deciding which attributes to keep using cross-validation. Table 4.1 and Figure 4.7 showed the results of this analysis. Only the first four attributes will be used, since the validation error increases after that point. These figures will not be repeated here.

I will now apply the PNN algorithm to the full seismic volume using the attributes shown in Table 4.1. Before looking at the seismic volume, we need to see how close the fit is at the wells. This is shown in Figure 6.9 for the three wells shown in Figure 4.6, where Figure 6.9(a) shows the training result and Figure 6.9(b) shows the validation result. You will recall from sections 3.7 and 4.4.2 that the training result shows the effect of computing the weighting coefficients from all the wells and applying them to all the wells, whereas the validation result shows the effect of leaving the well shown from the computation and is thus a "blind" prediction of this result.

In Figure 6.9 the RMS error is shown at the top of each result. The error for the training result is 0.185 and for the classification result is 0.399. This is an improvement over the Fisher linear discriminant of Figure 4.8, in which the training error was 0.3859 and the classification error was 0.4657.



(a)



(b)

Figure 6.9: The application of the PNN algorithm to the three wells shown in Figure 4.6, where (a) shows the training error, and (b) shows the validation error.

Finally, I will apply the PNN algorithm to the seismic data itself. The result over the seismic line from Figure 4.5 is shown in Figure 6.12, where gray shows the low porosity values, yellow shows the medium porosity values and blue shows the high porosity values. For comparison purposes I have redrawn Figure 4.8, the equivalent application of the Fisher linear discriminant, as Figure 6.13. When comparing Figure 6.12 and Figure 6.13, note that the PNN algorithm has enhanced both the lateral definition and continuity of the high porosity channel just below 1050 ms. Note that in both Figures 6.12 and 6.13, the inserted curve from well 08-08 is the density-porosity log, not the classified log.

Figure 6.13 then shows a slice of the classified porosity taken from a single sample that was extracted 20 ms below the picked event labelled Horizon 1 in Figure 6.12. The wells have also been indicated in Figure 6.13. As with the results on the single seismic line, the equivalent application of the Fisher linear discriminant over the volume, shown in Figure 4.9, has been redrawn as Figure 6.14. Notice when comparing these two figures that the lateral continuity of the porosity away from well 08-08 is better in the PNN result and that the false indications of high porosity across the upper and lower left parts of the map have been reduced, but not completely eliminated.

In general, it would appear that the PNN result for the porosity prediction case study is an improvement over the Fisher linear discriminant. However, it is important to stress that, except at the wells, we have no way of validating the result throughout the data volume. That is, we do not know was the correct answer is for the seismic data volume except when we drill. The fact that the two results give different answers (although they agree in some specific areas) is a little concerning. Remember that the Fisher linear discriminant is based on parametric statistics, whereas the PNN is based on non-parametric statistics. Thus, we would expect some differences. It is certainly worthwhile to apply both methods and make a decision as to which is preferred based on geological criteria.

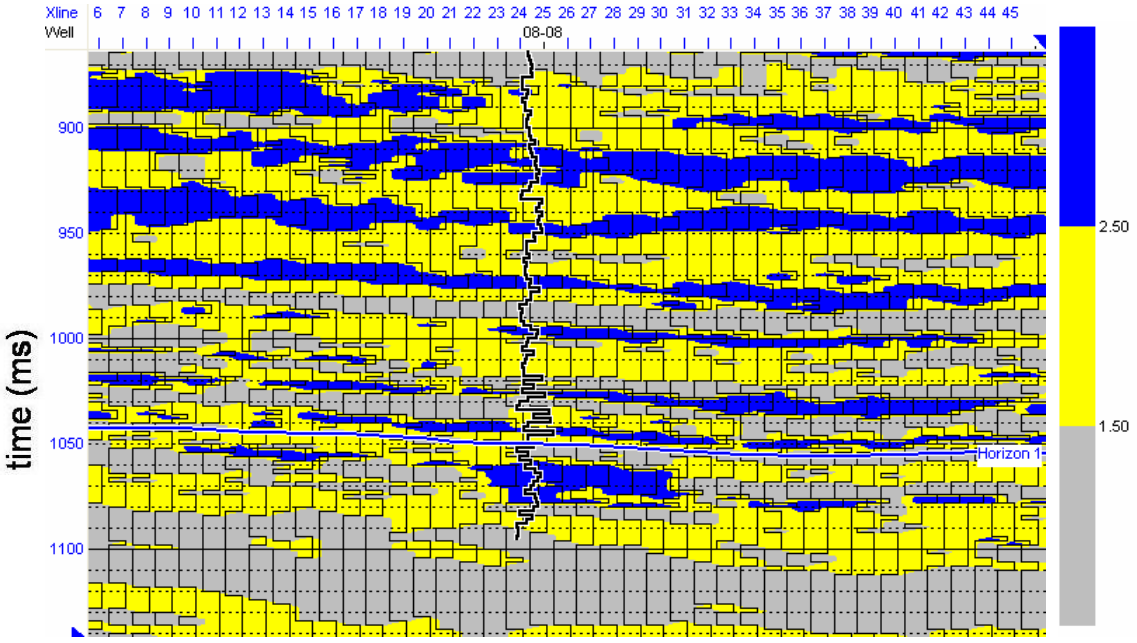


Figure 6.10: The application of the PNN to porosity classification on the seismic line of Figure 4.5.

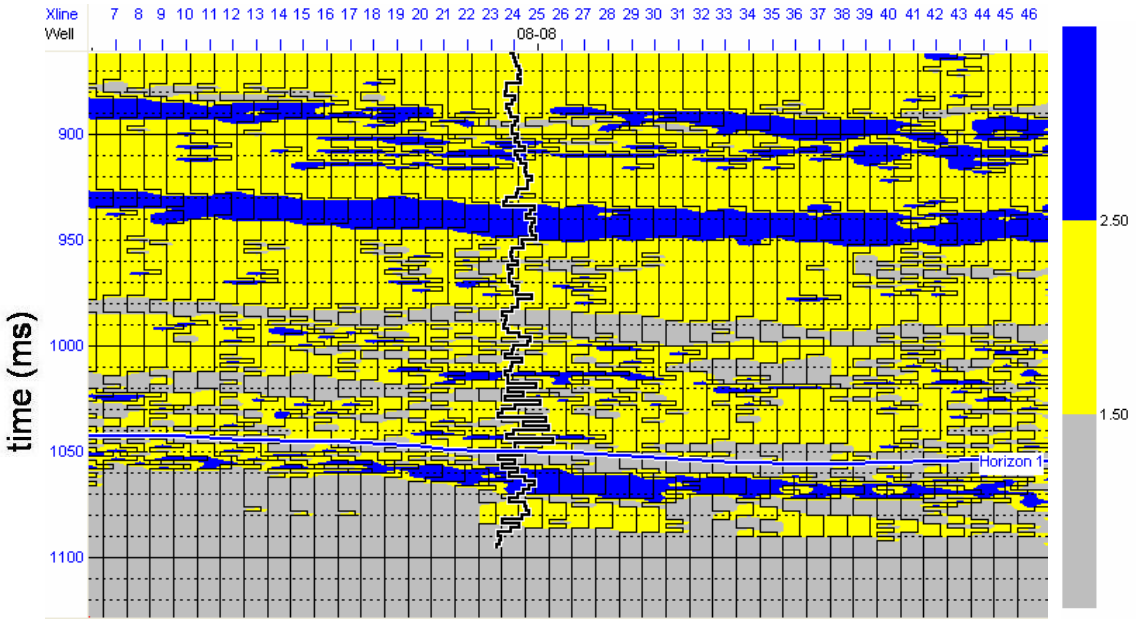


Figure 6.11: The Fisher linear discriminant results of porosity classification originally shown in Figure 4.8.

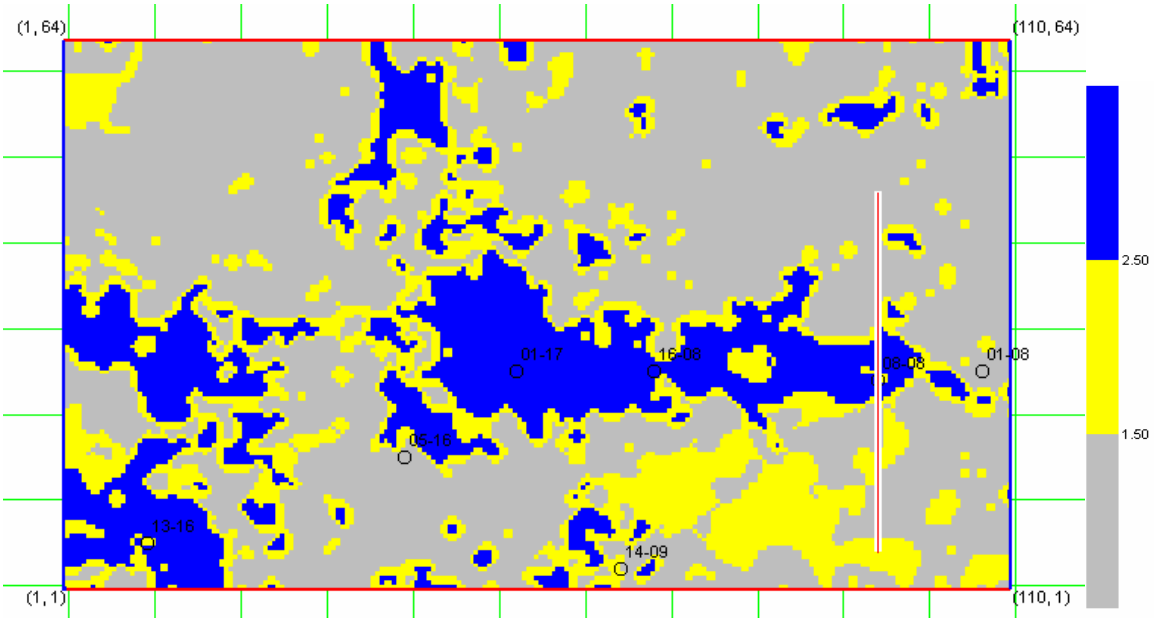


Figure 6.12: The application of the PNN to porosity classification in a window across the complete seismic survey area.

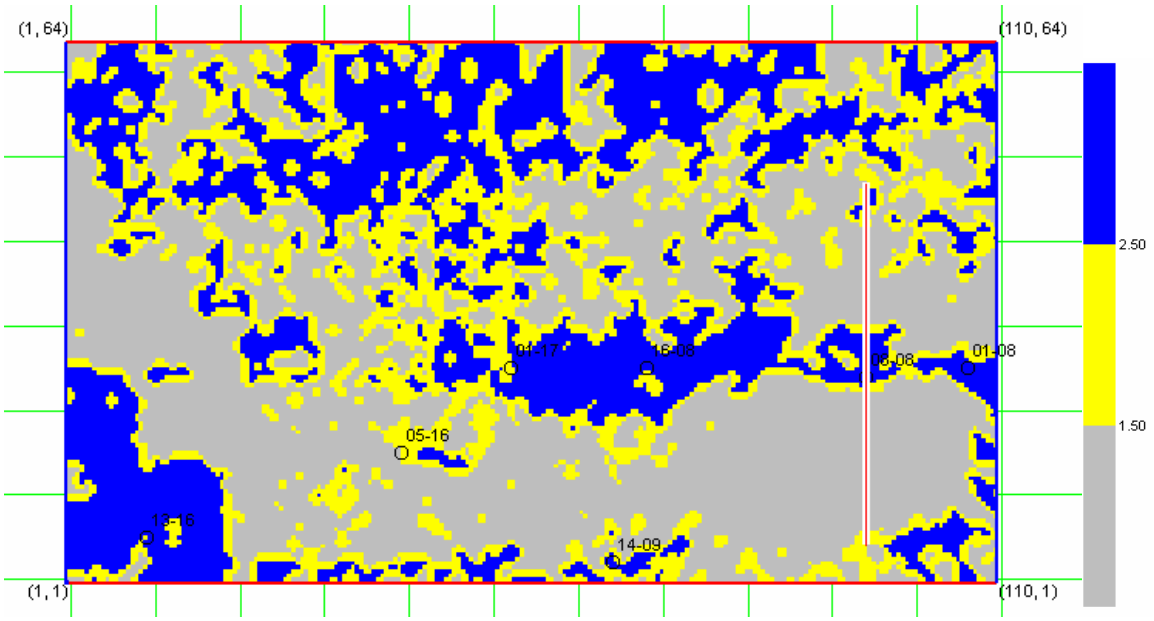


Figure 6.13: The Fisher linear discriminant results of porosity classification originally shown in Figure 4.9.

6.5 The generalized regression neural network (GRNN)

6.5.1 Theory of the GRNN

The generalized regression neural network, or GRNN, is based on the PNN and was initially proposed by Specht (1991), as was the PNN. A good summary of this method is given by Masters (1995). This method is actually a neural network implementation of a statistical technique that was proposed independently by Nadaraya (1964) and Watson (1964), and often referred to as the Nadaraya-Watson estimator. The derivation of the Nadaraya-Watson estimator can be done by using a two-dimensional Parzen kernel estimator that uses N pairs of inputs and targets, $\boldsymbol{s}_j$ and t_j, and can be written (Bishop, 1996) as

$$p(\boldsymbol{x},t)=\frac{1}{N}\sum_{j=1}^{N}\frac{1}{(2\pi\sigma^2)^{(M+1)/2}}\exp\left\{-\frac{\left|\boldsymbol{x}-\boldsymbol{s}_j\right|^2}{\sigma^2}-\frac{\left|t-t_j\right|^2}{\sigma^2}\right\}. \tag{6.16}$$

Equation (6.16) is the joint probability function and is illustrated in Figure 6.14 for a single attribute x. In Figure 6.14, the points represent the target values and the circles represent the Gaussian functions from the Parzen estimator.

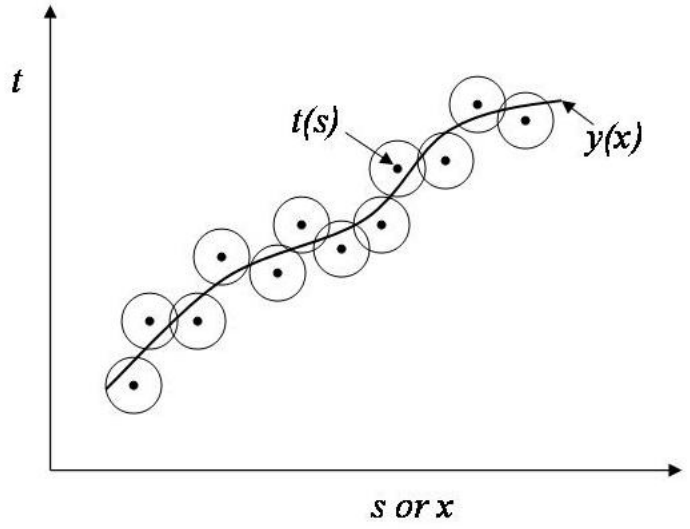


Figure 6.14: An illustration of the two-dimensional Parzen kernel estimator (adapted from Bishop, 1996).

The function *y(**x**)* shown in Figure 6.14 is the Nadaraya-Watson estimator. It can be derived (Bishop, 1996) by computing the conditional average of the target data on the input variables, given by

$$y(\boldsymbol{x}_k) = \langle t|\boldsymbol{x}\rangle = \int t\, p(t|\boldsymbol{x})dt = \frac{\int t\, p(t,\boldsymbol{x})dt}{\int p(t,\boldsymbol{x})dt}. \tag{6.17}$$

Substituting equation (6.16) into equation (6.17) gives

$$y(\boldsymbol{x}_k) = \frac{\sum_{j=1}^{N} t_j \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]}{\sum_{j=1}^{N} \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]} = \frac{\sum_{j=1}^{N} t_j \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]}{p(\boldsymbol{x}_k)}. \tag{6.18}$$

As mentioned earlier, the Nadaraya-Watson estimator of equation (6.18) was re-discovered in the context of neural networks and named the generalized regression neural network, or GRNN. This name is due to the fact that the training values themselves are weight values and thus this is indeed a type of generalized regression on the Gaussian basis functions. Note that the normalization factor in the denominator of equation (6.18) is the PNN estimate of the complete training dataset. We can observe how this equation works on the training data by re-writing equation (6.18) using the training data, or

$$y(\boldsymbol{s}_i) = \frac{\sum_{j=1}^{N} t_j \exp\left[-\frac{\left|\boldsymbol{s}_i - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]}{p(\boldsymbol{s}_i)}, \quad i = 1,2,\ldots,N. \tag{6.19}$$

Notice that when $i = j$, it follows that $\exp\left[-\frac{\left|\boldsymbol{s}_i - \boldsymbol{s}_i\right|^2}{\sigma^2}\right] = \exp[-0] = 1$.

A simple illustration of the GRNN method is given in Figure 6.15 for the same two attributes that were used to illustrate PNN in Figure 6.7. In Figure 6.15, the last sample on the target log, which is not known, is estimated using the *N* pairs of attribute values and their corresponding target values shown above.

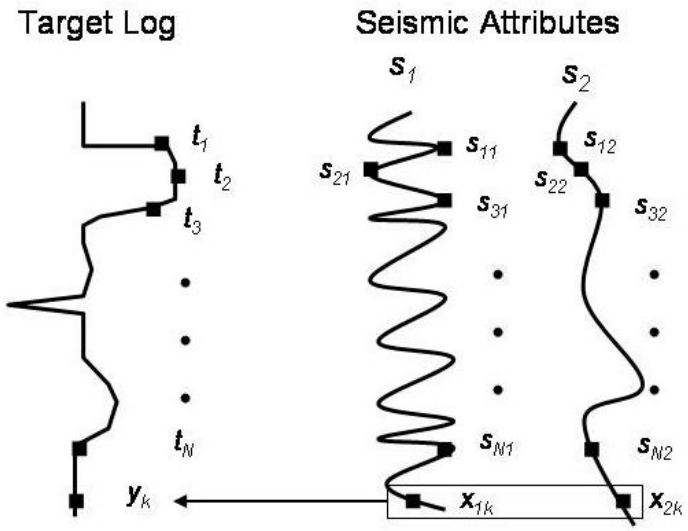


Figure 6.15: The attribute vectors and target values used in the GRNN and RBFN methods

6.5.2 Optimization of sigma in the GRNN method

As in the PNN method, the key parameter in the GRNN method is the sigma value, that will determine the width of the Parzen estimator in *N*-dimensional space. So far in our discussion, I have assumed that the sigma term is a constant. In fact, sigma can be made to vary as a function of each attribute. That is, equation (6.19) can be rewritten as

$$y(\boldsymbol{s}_i) = \frac{\sum_{j=1}^{N} t_j \exp\left[-\frac{(s_{1i} - s_{1j})^2}{\sigma_1^2} - \frac{(s_{2i} - s_{2j})^2}{\sigma_2^2} - \cdots \frac{(s_{Mi} - s_{Mj})^2}{\sigma_M^2} \right]}{p(\boldsymbol{s}_i)}, \; i = 1, 2, \ldots, N . \qquad (6.20)$$

where $p(\boldsymbol{s}_i) = \sum_{j=1}^{N} \exp\left[-\frac{(s_{1i} - s_{1j})^2}{\sigma_1^2} - \frac{(s_{2i} - s_{2j})^2}{\sigma_2^2} - \cdots \frac{(s_{Mi} - s_{Mj})^2}{\sigma_M^2} \right]$.

This approach was suggested by Masters (1996) who also supplies C++ code for finding the optimum values of the sigma values using the conjugate gradient approach.

6.5.3 Application of the GRNN to P-wave velocity prediction

In this section, I will apply the generalized regression neural network to the prediction of P-wave velocity in a channel sand from the Blackfoot oilfield of southern Alberta. This is the same case study which was discussed in section 5.5. The base map, inverted seismic line, and impedance dataslice for this study were shown in Figures 5.23 through 5.25, and will not be re-shown here. The P-wave log, extracted seismic trace and inverted trace used in the analysis are shown in Figure 6.16 for two of the nine wells in the study, which is a re-display of Figure 5.26. In Figure 6.16, the analysis window is shown by the horizontal red lines.

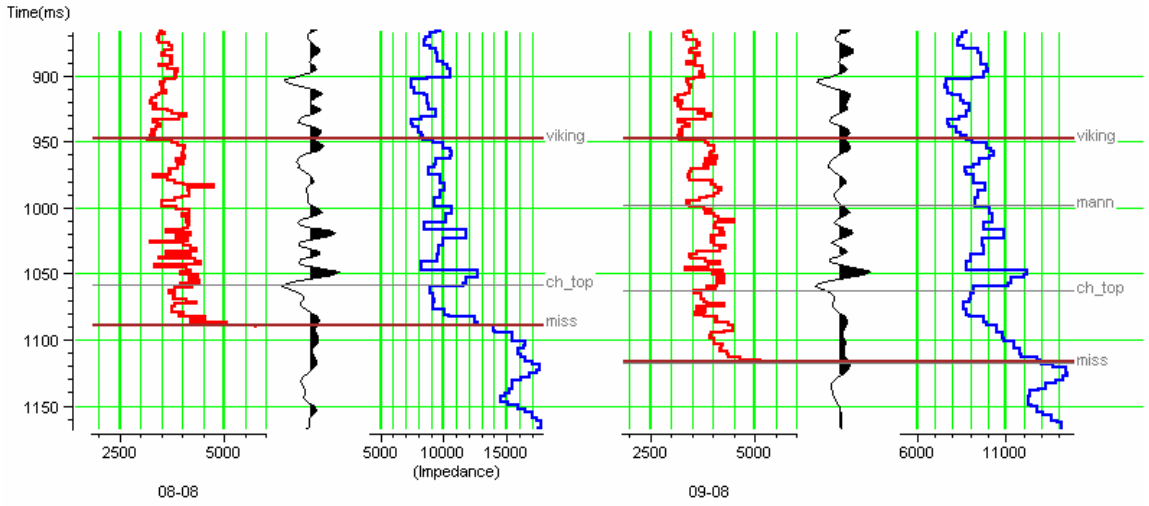


Figure 6.16. The P-wave sonic, extracted seismic trace and inverted impedance for wells 08-08 and 09-08. The zones of interest are marked by the horizontal red lines.

The best attributes were then computed using step-wise regression and the optimum attributes were found using cross-validation. Table 6.1 shows the results of this analysis. Only the first six attributes will be used, since the validation error starts to increase after six. In this analysis, I used an operator length of seven points, which was found to be the optimal length.

.	Target	Final Attribute	Training Error	Validation Error
1	Sqrt(P-wave)	1 / (Impedance)	262.794115	265.392307
2	Sqrt(P-wave)	Filter 15/20-25/30	252.846073	256.424282
3	Sqrt(P-wave)	Filter 35/40-45/50	246.300437	251.932202
4	Sqrt(P-wave)	Amplitude Weighted Phase	243.163417	250.644314
5	Sqrt(P-wave)	Filter 55/60-65/70	240.339386	249.522596
6	Sqrt(P-wave)	Amplitude Weighted Frequency	237.576412	248.880681
7	Sqrt(P-wave)	Instantaneous Frequency	235.328710	249.518669

Table 6.1. The optimum attributes for predicting P-wave velocity, computed using the multilinear regression approach with a seven-point convolutional operator.

In the case study of section 5.5, I applied both the multilinear regression approach and the multi-layer perceptron approach. For the multilinear regression approach, the comparison between the training result and validation result for the predicted and actual sonic log values at wells 08-08, 09-08 and 09-17 was shown in Figure 5.29. For the multi-layer perceptron approach, the comparison between the training result and validation result for the predicted and actual sonic log values was shown in Figure 5.33. The key point to note from those figures was that although the correlation coefficient for the multi-layer perceptron *training* result was higher than the correlation coefficient for the multi-linear regression *training* result (0.739 versus 0.709), the correlation coefficient for the multi-linear regression *validation* result was higher than the correlation coefficient for the multi-layer perceptron *validation* result (0.674 versus 0.522). This suggests that the multi-layer perceptron has been overtrained.

I will now apply the GRNN approach to the prediction of P-wave velocity, using the attributes shown in Table 6.1. This will be done by optimizing the σ value in equation (6.19), but will be extended so that we compute a different sigma value for each attribute, as discussed in the last section. Table 6.2 shows the computed σ values for each of the attributes. Since we used a 7-point symmetrical convolutional filter to

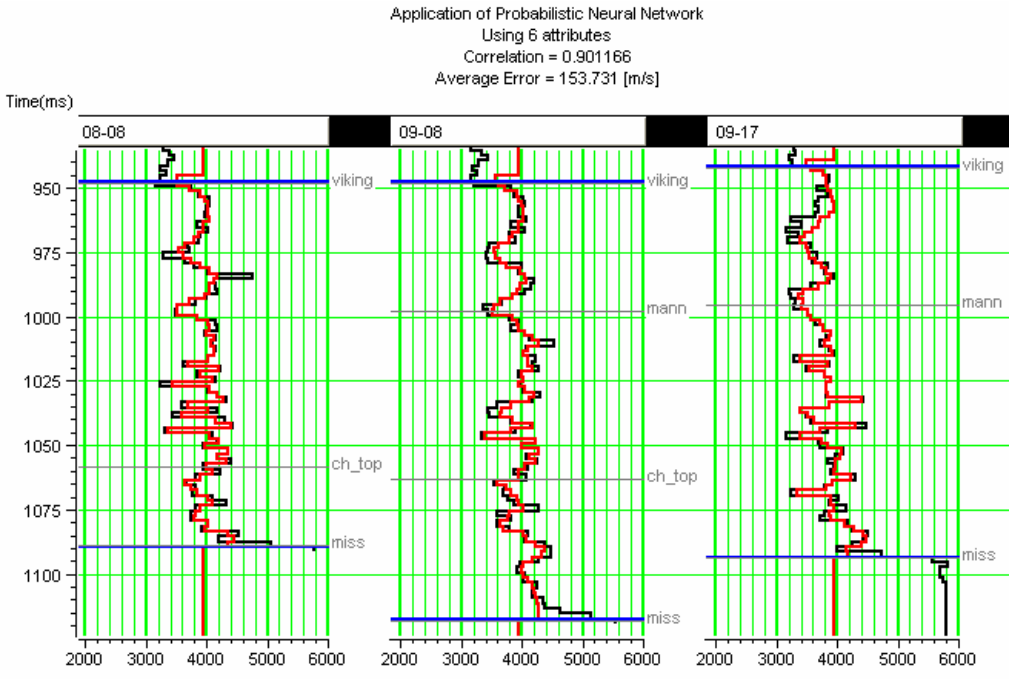
compute the attributes, this is equivalent to introducing a set of new attributes at each filter lag. A new σ value is computed at each of these lags and is shown in the table.

Lag	Impedance	Filter 1	Filter 2	Phase	Filter 3	Frequency
-3	0.819	0.206	0.740	0.442	1.194	1.137
-2	0.539	0.693	0.283	0.433	0.973	1.302
-1	0.506	0.922	0.891	1.127	1.334	1.240
0	0.207	1.077	1.091	0.509	1.364	1.173
1	1.519	1.168	1.154	0.931	1.058	1.390
2	1.135	1.198	1.011	0.991	0.804	1.295
3	0.469	1.184	0.666	1.007	1.235	1.298

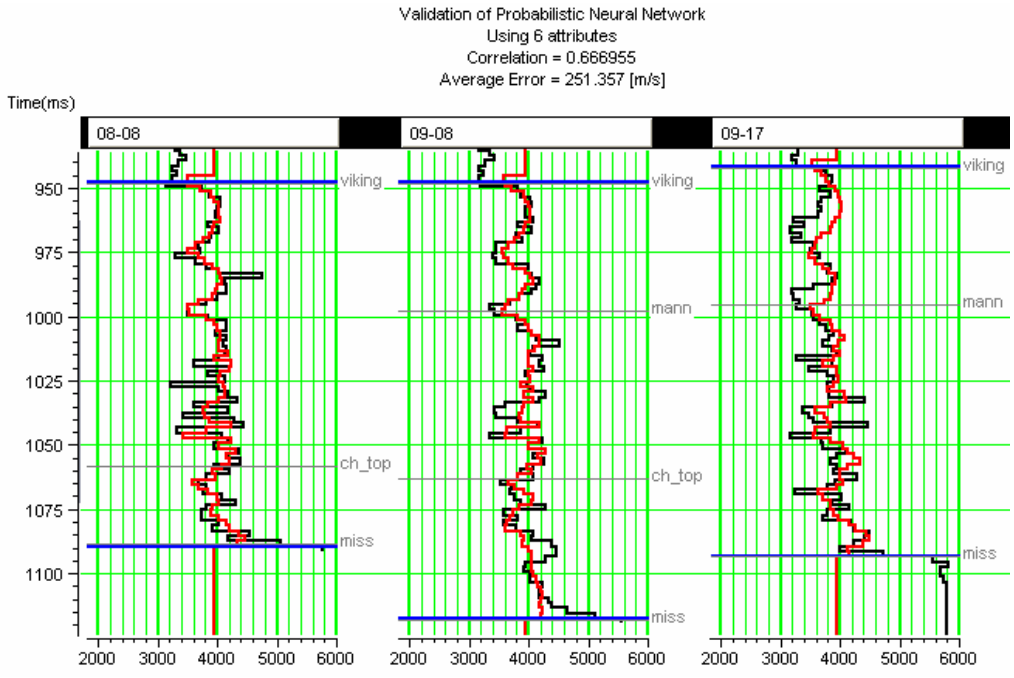
Table 6.2. The σ values computed for each of the first six attributes shown in Table 6.1.

The comparison between the predicted and actual sonic log values after the GRNN process, at wells 08-08, 09-08 and 09-17, is shown in Figure 6.17. Figure 6.17(a) shows the training result, in which all the wells are used in the calculation, and Figure 6.17(b) shows the validation result, in which the well being predicted is left out of the training calculation.

In Figure 6.17, notice that the correlation coefficient for the training result is 0.901, which is better than both multilinear regression and the multi-layer perceptron, and that although the correlation coefficient for the validation result drops to 0.667, this is still much better than for both the multi-linear regression and the multi-layer perceptron. A crossplot of all of the logs is then shown in Figure 6.18. Notice that the fit is much better than for the crossplots shown in section 5.5 for the multi-linear regression and multi-layer perceptron methods.



(a)



(b)

Figure 6.17. A comparison between the computed (red) and original (black) well logs for wells 08-08, 09-08, and 09-17, where (a) shows the training result and (b) shows the validation result.

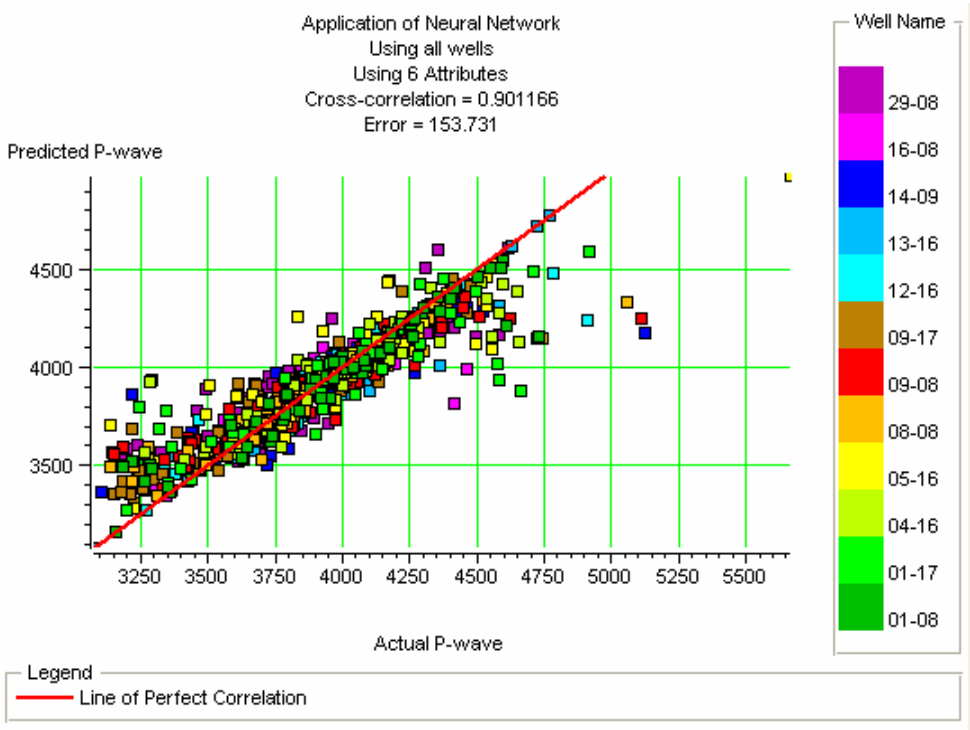


Figure 6.18. A crossplot of the predicted P-wave logs from the GRNN approach against the original logs.

I will now apply the multilinear regression coefficients derived in the GRNN training to produce a full volume of pseudo-P-wave sonic logs. Figure 6.19 shows the predicted P-wave values for seismic line 95. Notice the extra detail that is seen in the channel that intersects the well log below Horizon 1, when compared to the multilinear regression result of Figure 5.31. There is also better lateral continuity throughout the section than in the multi-layer perceptron result.

Finally, Figure 6.20 then shows a dataslice from the complete 3D survey, again showing the mean average over a 10 ms zone that was centered on a "phantom" event created by shifting Horizon 1 down by 20 ms.

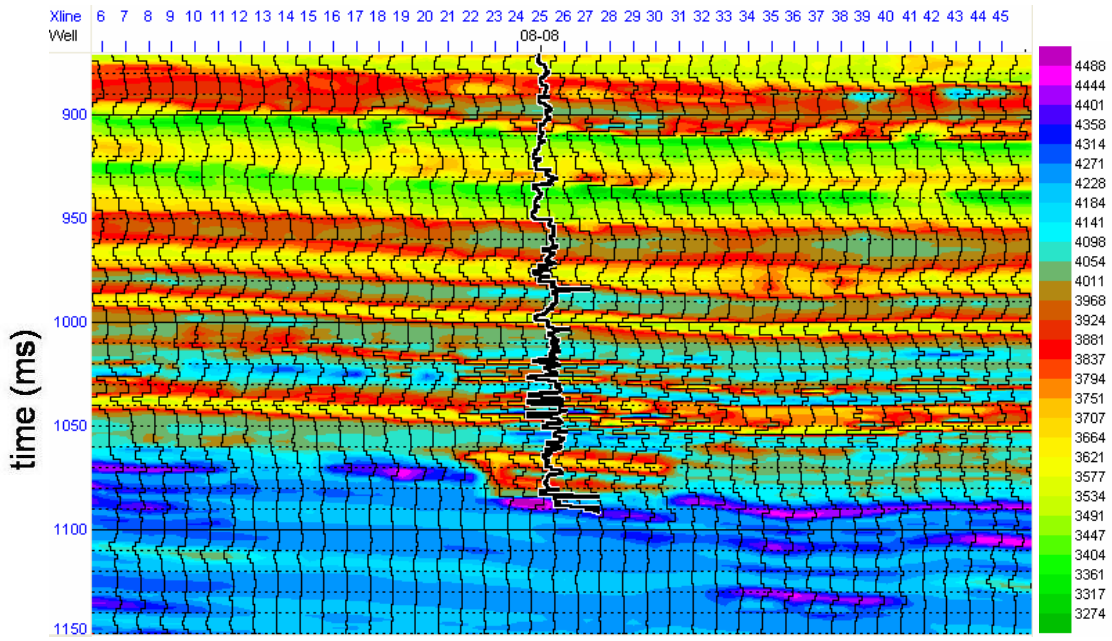


Figure 6.19. The predicted P-wave sonic logs over line 95 using the generalized regression neural network.

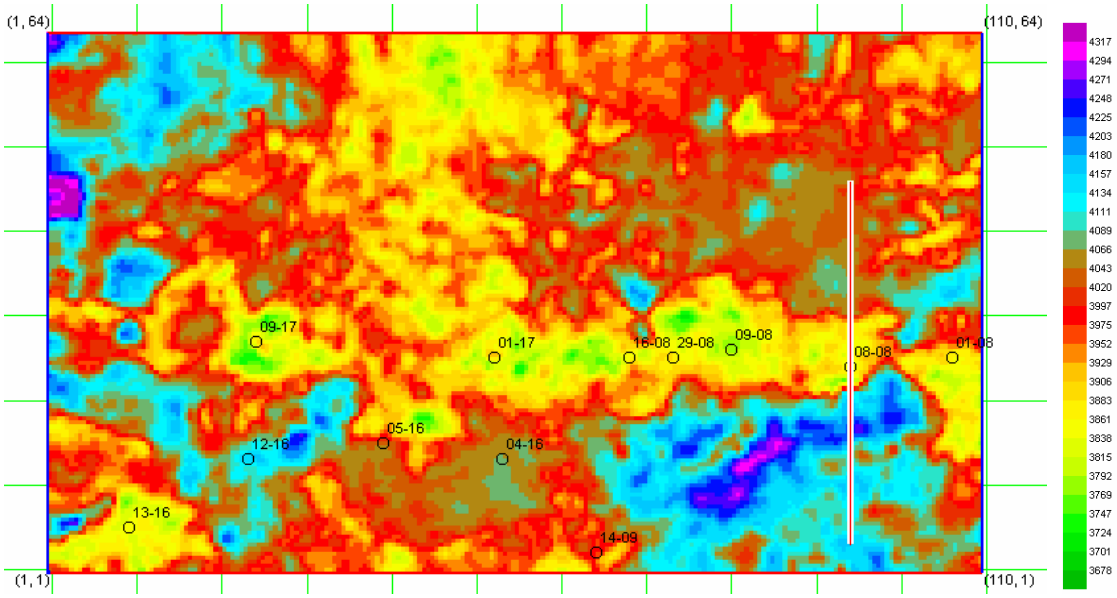


Figure 6.20. The extracted P-wave slice over the complete seismic survey, averaged over a 10 ms window that was shifted 20 ms below Horizon 1 in Figure 6.19.

6.6 The radial basis function neural network (RBFN)

6.6.1 Theory of the RBFN

The radial basis function neural network, or RBFN, was originally developed as a method for performing exact interpolation of a set of data points in multi-dimensional space (Powell, 1987). Referring again to Figures 6.5 and 6.6 in section 6.3 for our notation, we want to find a function *y(**x**)* that satisfies

$$y(\boldsymbol{s}_i) = t_i, \;\; i = 1, 2, \ldots, N, \tag{6.21}$$

where the t_i values are our training samples, and the $\boldsymbol{s}_i$ values are our attribute vectors.

In its most general form, the problem can be formulated as

$$t(\boldsymbol{s}_i) = \sum_{j=1}^{N} w_j \phi\left(\left|\boldsymbol{s}_i - \boldsymbol{s}_j\right|\right) = \sum_{j=1}^{N} w_j \phi_{ij}, \; i = 1, 2, \ldots, N, \tag{6.22}$$

where the functions $\phi\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right)$ are a set on N radial basis functions that depend on the attribute distances, which can be abbreviated as ϕ_{ij}. (Note that equation (6.22) is the generalized linear discriminant function that was given in equation (4.49).) A radial basis function is a function whose response decreases (or increases) monotonically with distance away from a central point (Orr, 1996). There are many functions that satisfy this requirement (Bishop, 1996), such as the thin-plate spline function given by

$$\phi(x) = x^2 \ln(x), \tag{6.23}$$

and the multi-quartic function given by

$$\phi(x) = (x^2 + \sigma^2)^{\beta}, \quad 0 < \beta < 1. \tag{6.25}$$

However, it has been found that the most effective function is the Gaussian basis function. Equation (6.22) can thus be written as

$$t(\boldsymbol{s}_i) = \sum_{j=1}^{N} w_j \phi_{ij} = \sum_{j=1}^{N} w_j \exp\left[-\frac{\left|\boldsymbol{s}_i - \boldsymbol{s}_j\right|^2}{\sigma^2}\right], \quad i = 1, 2, \ldots, N, \tag{6.26}$$

where the w_j, $j = 1, \ldots, N$, are the desired weights. As shown in Appendix 6, equation (6.26) can be derived from basic principles using the theory of regularization, and involves the introduction of pseudo-differential operators (Poggio and Girosi, 1990).

To solve for the weights in equation (6.26), notice that it can be written as a set of N equations in N unknowns, or

$$\begin{aligned} t_1 &= w_1 \phi_{11} + w_2 \phi_{12} + \ldots + w_{1N} \\ t_2 &= w_1 \phi_{21} + w_2 \phi_{22} + \ldots + w_{2N} \\ \vdots \quad &\quad \vdots \quad\quad \vdots \quad\quad\quad \vdots \\ t_N &= w_1 \phi_{N1} + w_2 \phi_{N2} + \ldots + w_{NN} \end{aligned} \tag{6.27}$$

Equation (6.27) can be written more compactly as the matrix equation:

$$\boldsymbol{t} = \boldsymbol{\Phi}\, \boldsymbol{w}, \tag{6.28}$$

where $\boldsymbol{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix}$, *and* $\boldsymbol{\Phi} = \begin{bmatrix} \phi_{11} \cdots \phi_{1N} \\ \vdots \ \ddots \ \vdots \\ \phi_{N1} \cdots \phi_{NN} \end{bmatrix}$.

The solution to equation (6.28) was discussed extensively in Chapter 3 and is given by

$$\boldsymbol{w} = \left[\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right]^{-1} \boldsymbol{t}, \tag{6.29}$$

where λ is a pre-whitening factor and I is the identity matrix. This equation can be solved efficiently by noting that the matrix is symmetric. Once the weights have been computed, they are applied to the application dataset (see Figure 6.5) using

$$y(\boldsymbol{x}_k) = \sum_{j=1}^{N} w_j \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]. \tag{6.30}$$

6.6.2 Optimization of sigma for the RBFN method

As in the PNN method and the GRNN methods, the key parameter in the RBFN method is the sigma value σ. I showed in section 6.4.2 that σ could be written as a function of each attribute for the GRNN method. This is also the case for the RBFN method in theory. That is, equation (6.20) could be rewritten in the RBFN case as

$$y(\boldsymbol{s}_i) = \sum_{j=1}^{N} w_j \exp\left[-\frac{(s_{1i} - s_{1j})^2}{\sigma_1^2} - \frac{(s_{2i} - s_{2j})^2}{\sigma_2^2} - \cdots \frac{(s_{Mi} - s_{Mj})^2}{\sigma_M^2} \right], \; i = 1, 2, \ldots, N \,. \qquad (6.31)$$

Unfortunately, no efficient way of optimizing σ as a function of each attribute has been obtained for the RBFN method, as was the case for the GRNN method. In fact, even optimizing a constant σ value turns out to be very time consuming for the RBFN method, since a matrix inversion is required for each change in this parameter. Initially, a trial-and-error approach was adopted, in which the value of σ was varied over a range of values and the σ value that gave the minimum least-squared error was chosen. However, the problem with this approach is that as the increment of σ becomes small, and the search range becomes large, the computer time requirements for solving the problem tend to approach infinity! Thus, I chose to use the parabolic search method to find the optimum value for σ.

This parabolic search method, described by Press et al. (1992), is illustrated in Figure 6.21. We initially choose three values of σ: low, intermediate and high, and compute the RMS error between RBFN response of our training data and the actual values of the training data. These error values are illustrated by points 1, 2 and 3 in Figure 6.21. We then fit a parabola to these three points, illustrated by Parabola A in the figure. We next find the minimum of Parabola A, which is labelled point 4 in Figure 6.21. We then replace point 3 with point 4 and compute a new parabola, labelled Parabola B in the figure. The minimum of Parabola B is point 5, and this in turn replaces point 4

for a new parabolic search. If we consider the three points on the parabola to be given as *a, b* and *c*, and the error functions of these points as *f(a), f(b)* and *f(c)*, the equation for finding the minimum is given by

$$\sigma = b - \frac{1}{2}\frac{(b-a)^2[f(b)-f(c)]-(b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(c)]-(b-c)[f(b)-f(a)]}, \qquad (6.32)$$

where σ is the new value of sigma. This procedure is repeated until we converge to a minimum error, with the resulting σ value being the optimum value. In Figure 6.21, point 5 appears to be a reasonable estimate of the minimum value.

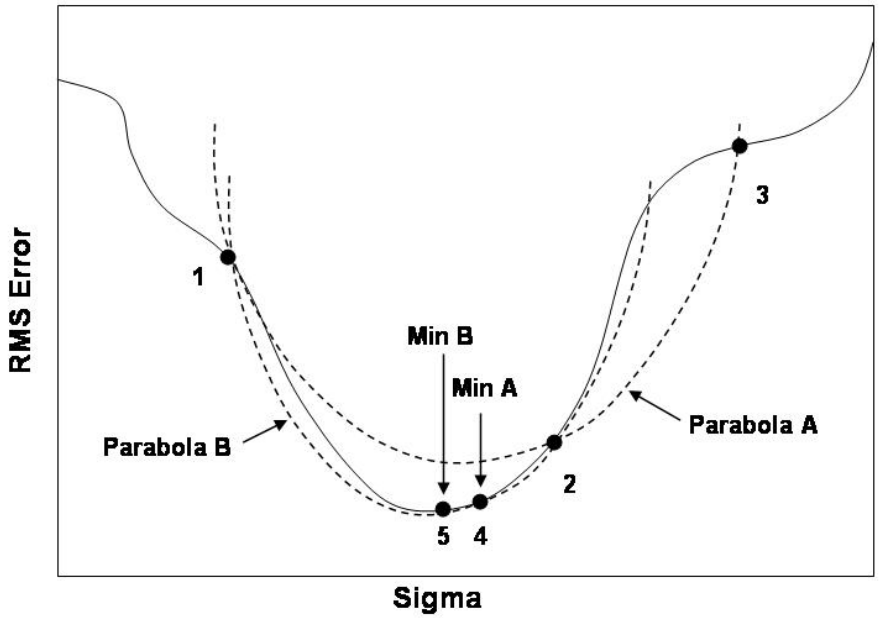


Figure 6.21. An illustration of the parabolic search method (adapted from Press et al., 2002)

Before applying the RBFN to a real dataset, I will discuss the relationship between the RBFN and the GRNN, both in theory and then using two straightforward functions.

6.7 The relationship between the RBFN and the GRNN methods

To understand the relationship between the RBFN and the GRNN methods, recall that the GRNN relationship given in equation (6.18) was written

$$y(\boldsymbol{x}_k) = \frac{\sum_{j=1}^{N} t_j \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]}{p(\boldsymbol{x}_k)}, \tag{6.33}$$

where $p(\boldsymbol{x}_k)$ is a normalization constant which is identical to the PNN function for all N training points that was defined in equation (6.10).

By equating equations (6.33) and (6.30), note that equation (6.33) can be thought of as the general form of both the RBFN and the GRNN if we rewrite the weights as

$$w_j = \frac{t_j}{\sum_{j=1}^{N} \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]} = \frac{t_j}{p(\boldsymbol{x}_k)}. \tag{6.34}$$

If we assume that the scaling factors σ are identical in equations (6.30) and (6.33), and that $p(\boldsymbol{x}_k) = 1$, then we simply need to look at the relationship between t_j and w_j to equate the two methods. By expanding equation (6.29), we find that

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} \phi_{11}+\lambda & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22}+\lambda & \cdots & \phi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN}+\lambda \end{bmatrix}^{-1} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} \psi_{11} & \psi_{12} & \cdots & \psi_{1N} \\ \psi_{21} & \psi_{22} & \cdots & \psi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{N1} & \psi_{N2} & \cdots & \psi_{NN} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}, \tag{6.35}$$

where $\phi_{ij} = \exp\left[\frac{-\left|\boldsymbol{s}_i - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]$, and ψ_{ij} is an element of the inverted matrix. Therefore, we can write the equation for a single weight as

$$w_j = \psi_{j1} t_1 + \psi_{j2} t_2 + \ldots + \psi_{jN} t_N . \tag{6.36}$$

From equation (6.34) we see that w_j is a weighted sum of all the training values. If the inverted matrix consisted only of values along the main diagonal, we could rewrite equation (6.35) as

$$w_j = \psi_{jj} t_j . \tag{6.37}$$

If this was the case, Φ would consist only of the values ϕ_{jj} along the main diagonal. But note that

$$\phi_{jj} = \exp\left[-\frac{\left| \boldsymbol{s}_j - \boldsymbol{s}_j \right|^2}{\sigma^2} \right] = 1 . \tag{6.38}$$

Thus, for the case given in equation (6.38), $\Phi = I$, the identity matrix, and $\Phi^{-1} = I$. Therefore, for this trivial case, the RBFN and GRNN methods are identical to a scale factor. This would occur in two situations:

(1) $\left| s_i - s_j \right| \to 0,$ for all i and j, and

(2) $\sigma \to 0$

In the more general case, the off-diagonal elements in the covariance matrix are all non-zero. Indeed, they will only be equal to zero for the case in which the seismic attributes are statistically independent. Thus, we can think of the GRNN method as a subset of the RBFN method for the case of statistical independence of the attributes. We would therefore expect the RBFN method to give a more high resolution result than the GRNN method, since the off-diagonal covariance elements are being used. Before trying to observe this for a real data case, I will discuss two simpler cases involving analytical functions.

6.8 RBFN/GRNN Comparison for two simple functions

In this section, I will compare the RBFN and GRNN approaches by applying them to the prediction of known analytical functions. The two functions I will use are the sine wave and the step function. These functions were chosen because of their contrasting nature. The sine wave is a smooth function with no discontinuities and can be thought of as a prototypical seismic trace, whereas the square wave has a discontinuity at its step and can be thought of as a prototypical well log curve. Since we are dealing with a one-dimensional problem, we can replace the vector notation given earlier with scalar notation. That is, we can write the equations for the GRNN as

$$y(x) = \frac{\sum_{j=1}^{N} t_j \exp\left[-\frac{(x-x_j)^2}{\sigma^2}\right]}{\sum_{j=1}^{N} \exp\left[-\frac{(x-x_j)^2}{\sigma^2}\right]}, \tag{6.39}$$

and for the RBFN as:

$$y(x) = \sum_{j=1}^{N} w_j \exp\left[-\frac{(x-x_j)^2}{\sigma^2}\right], \tag{6.40}$$

where the weights in equation (6.40) are computed from the solution to

$$t(\boldsymbol{x}_n) = \sum_{j=1}^{N} w_j \exp\left[-\frac{\left|\boldsymbol{x}_n - \boldsymbol{x}_j\right|^2}{\sigma^2}\right] = \sum_{j=1}^{N} w_j \phi_{ij}, \quad n = 1, 2, \ldots, N. \tag{6.41}$$

I will start with the sine wave example. To train the two algorithms, I will use the function

$$t(x_i) = \sin(2\pi x_i), \quad x_i = 0,\ 0.2,\ \ldots,\ 1.0, \tag{6.42}$$

as suggested by Bishop (1995). (However, Bishop only tests the RBFN method on this function, and does not compare RBFN to GRNN.) Notice that we have sampled this function six times in one period. To validate the algorithms, we will use the function

$$y(x_i) = \sin(2\pi x_i), \quad x_i = 0,\ 0.05,\ \ldots,\ 1.0, \tag{6.43}$$

where the sine wave has been sampled twenty-one times in one period. Note that we are able to check the results of the validation function since we know the right answer.

A graph of the functions given in equations (6.41) and (6.42) is shown in Figure 6.22, where the validation curve is shown as the smoothly interpolated line, and the sampled training values are shown by the solid dots.

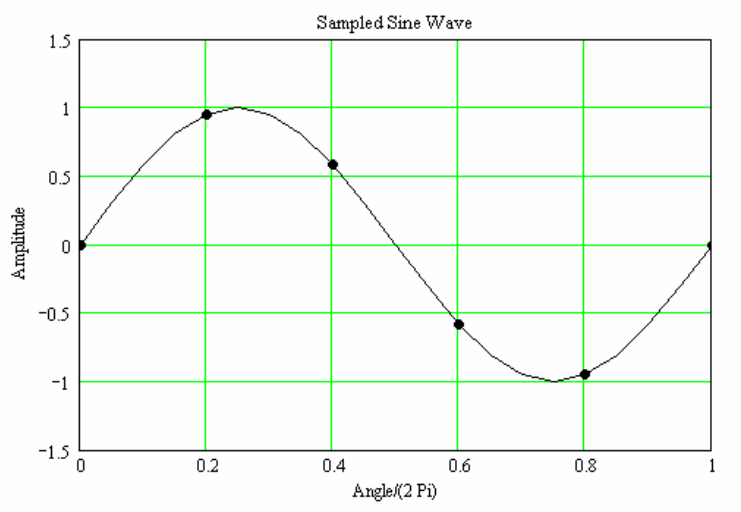


Figure 6.22: A graph of the sine function in equations (6.40) and (6.41), where the training samples are shown by dots, and the true values of the validation function by the solid curve.

The results of applying both the RBFN and the GRNN algorithms to this sampled sine wave are shown in Figure 6.23. It is important to note that the value of σ has a major impact on the results, and the optimum value is usually different for the two methods. I have therefore used four different values of sigma in Figure 6.23: (a) *1.0,* (b) *0.1,* (c) *0.01,* and (d) *0.001*. In each case in Figure 6.23, the solid curve shows the true sine wave values, the dotted curve shows the RBFN results, and the dashed curve shows the GRNN results.

A number of observations can be made by referring to Figure 6.23. First, for $\sigma =$ *1.0*, shown in Figure 6.23(a), RBFN has done an almost perfect job in predicting the sine wave curve, whereas the GRNN has simply fit a straight line to the curve. For $\sigma =$ *0.1*, as in Figure 6.23(b), neither the RBFN nor the GRNN has done a perfect job of fitting,

but GRNN appears to fit the sine curve better than RBFN. When we decrease the $\sigma =$ 0.01, as in Figure 6.23(c), the GRNN is still doing a reasonable job of fitting the sine curve, but the RBFN has fit the function with spikes at the six training points, going to zero otherwise. Finally, when we decrease the $\sigma = 0.001$, as in Figure 6.23(d), both the RBFN and the GRNN methods have converged to the same result, fitting the function with spikes at the six training points, but going to zero otherwise. This is the same result obtained with the RBFN using a σ value of 0.01.

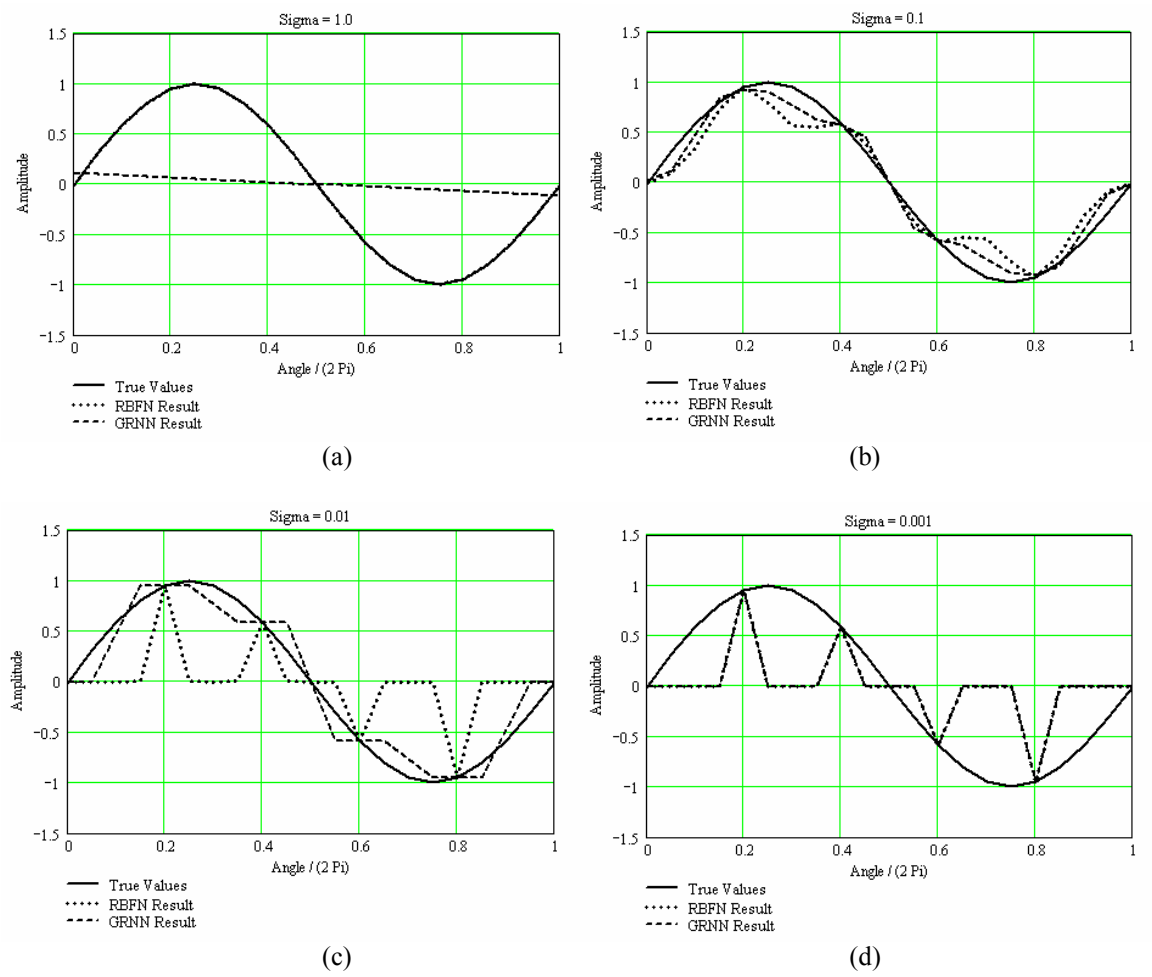


Figure 6.23: The GRNN and RBFN results of the sampled sine wave shown in Figure 6.21, for σ values of (a) 1.0, (b) 0.1, (c) *0.01*, and (d) 0.001.

To quantify these observations, Figure 6.24 shows the errors between the true sine wave and the GRNN and RBFN results shown in Figure 6.23.

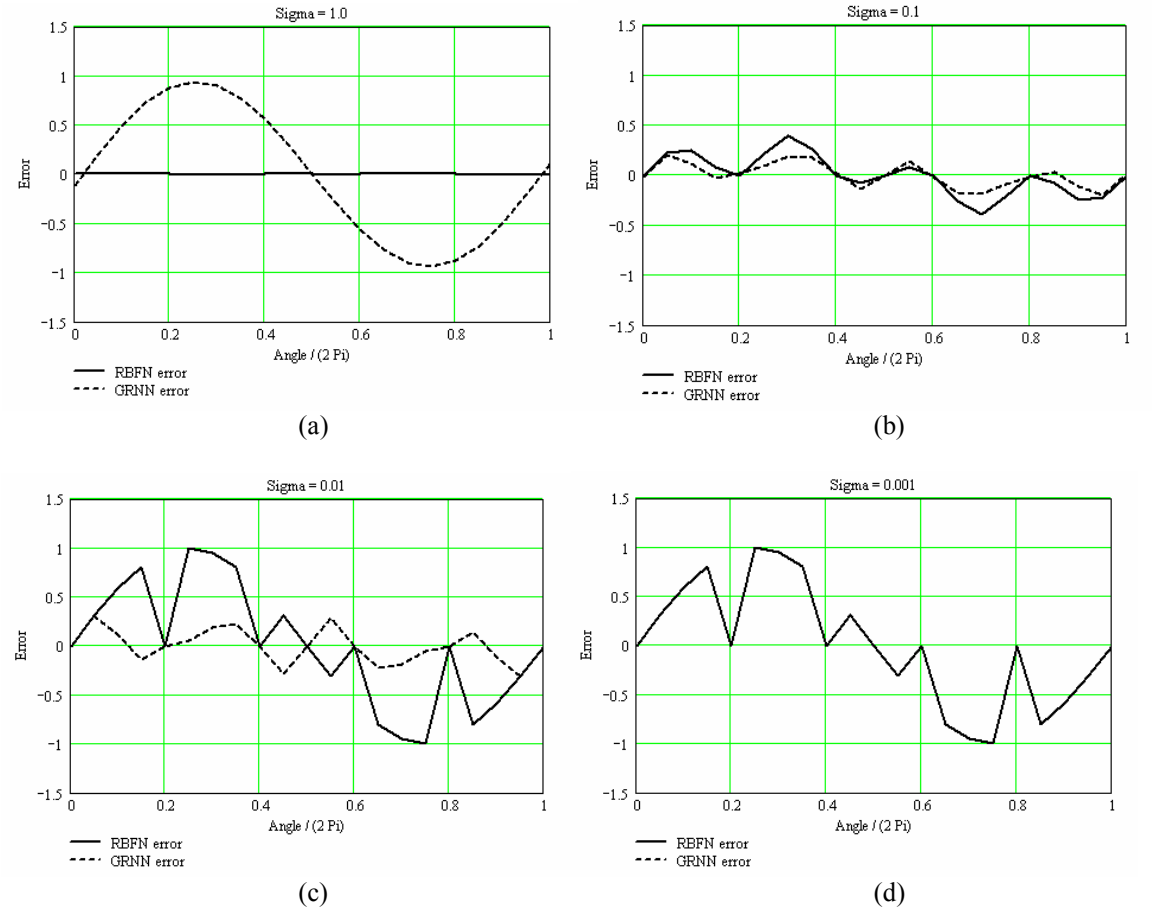


Figure 6.24: The errors for the GRNN and RBFN results shown in Figure 6.23, for (a) $\sigma = 1.0$, (b) $\sigma = 0.1$, (c) $\sigma = 0.01$, and (d) $\sigma = 0.001$.

As seen in Figure 6.24, the RBFN error is almost zero for a $\sigma = 1.0$, the GRNN error is consistent over a range of σ values between 0.1 and 0.01, the RBFN method gives identical results for σ values below 0.01, and the two methods converge to the same answer when $\sigma = 0.001$. This last point confirms point (2) from the theory in section 6.6, that the two methods should converge as σ approaches 0.

Next, I will consider the step function defined by

$$y(x) = \begin{cases} +1.0, & x \geq 0.5, \\ -1.0, & x < 0.5. \end{cases} \tag{6.44}$$

Figure 6.25 is a graph of this wave, where we have sampled the function six times for the training and 101 times for the validation. (We used more points in the validation curve than for the sine wave in order to image the step more accurately).

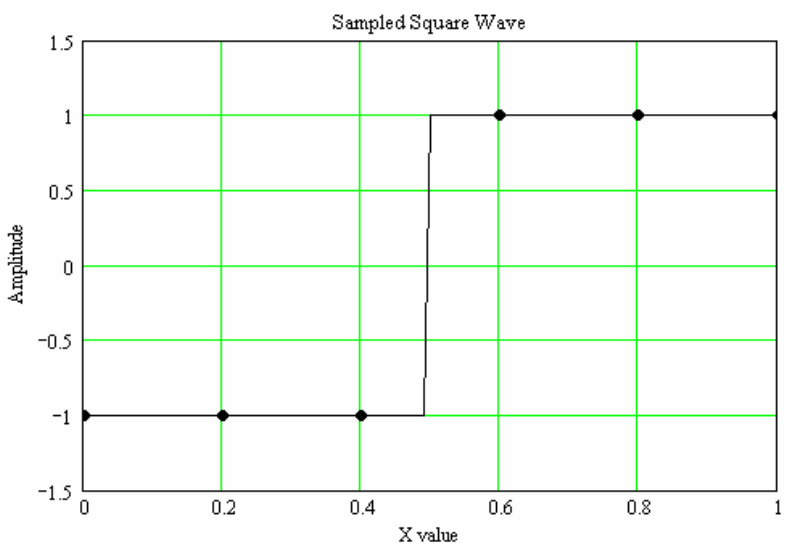


Figure 6.25: A graph of the step function in equation (6.42), where the training samples are shown by dots, and the true values of the validation function by the solid curve.

The results of applying both the RBFN and the GRNN algorithms to this sampled step function are shown in Figure 6.26. I have again used four different values of sigma in Figure 6.26: (a) *1.0,* (b) *0.1,* (c) *0.01,* and (d) *0.0001*. In each case in Figure 6.26, the solid curve shows the true step function values, the dotted curve shows the RBFN results, and the dashed curve shows the GRNN results.

The results shown in Figure 6.26 are quite different than the results of Figure 6.23. For σ=1.0, shown in Figure 6.26(a), the GRNN has again fit a smooth line to the function, but the RBFN has "overshot" the square wave on both the negative and positive sides. This is a classic "Gibbs phenomena" effect, observed in the theory of Fourier series. When σ= 0.1, as in Figure 6.26(b), the RBFN has again "overshot" the two sides of the square wave, but the GRNN has actually fit the square wave perfectly except at the step. At the step, both the RBFN and the GRNN have behaved almost identically. When

$\sigma = 0.01$, as in Figure 6.26(c), the GRNN now fits almost perfectly, but the RBFN has fit the function with spikes at the six training points, and goes to zero otherwise. Finally, when we decrease the $\sigma = 0.0001$, as in Figure 6.26(d), both the RBFN and the GRNN methods have converged to the same result, fitting the function with spikes at the six training points, but going to zero otherwise. This is the same result obtained with the RBFN using a σ value of 0.01.

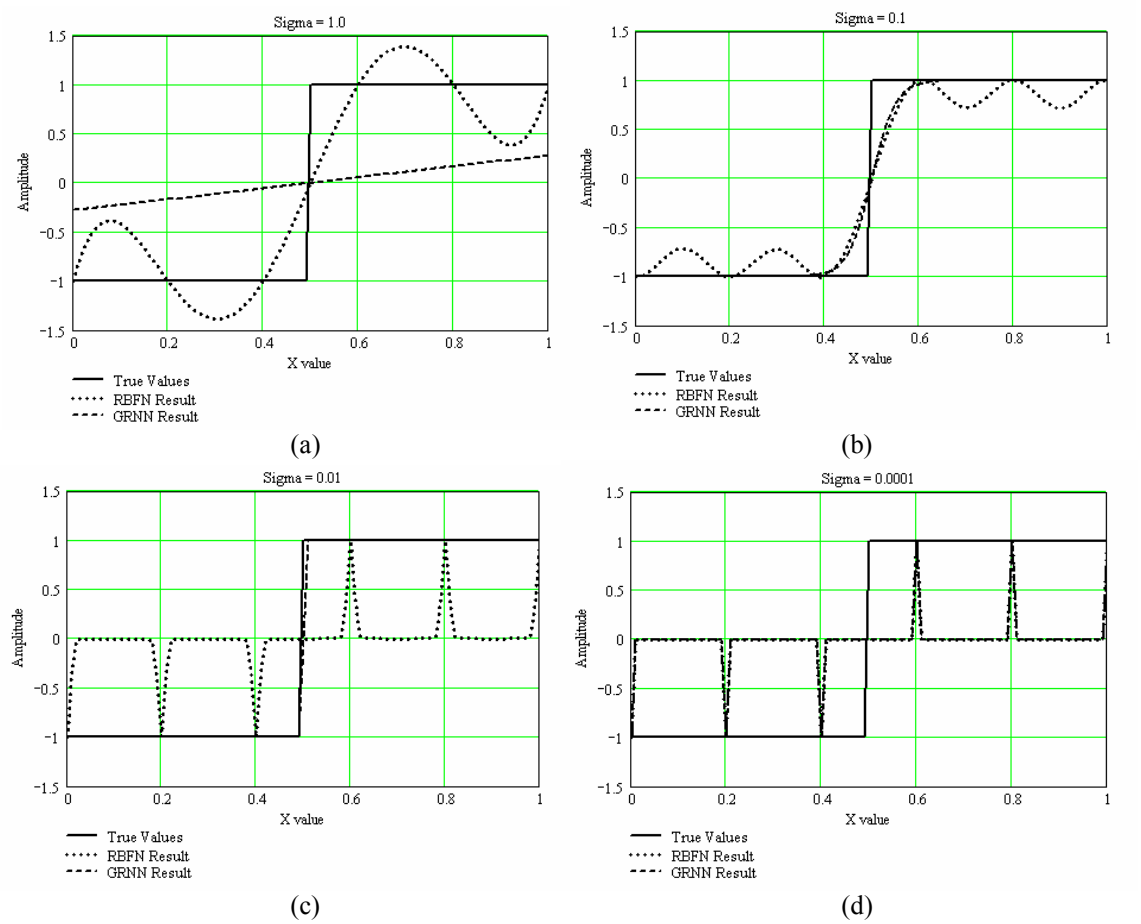


Figure 6.26: The GRNN and RBFN results of the sampled square wave shown in Figure 6.25, where the scale values are (a) $\sigma = 1.0$, (b) $\sigma = 0.1$, (c) $\sigma = 0.01$, and (d) $\sigma = 0.0001$.

To quantify these observations, Figure 6.27 shows the errors between the true square wave and the GRNN and RBFN results shown in Figure 6.26. As seen in Figure 6.27, the RBFN error shows an "overshoot" for a σ value of 1.0, the GRNN error appears to be very consistent over a range of σ values between 0.1 and 0.01, and fits almost

perfectly for $\sigma = 0.01$, the RBFN method gives identical results for σ values below 0.01, and the two methods converge to the same answer when $\sigma = 0.0001$.

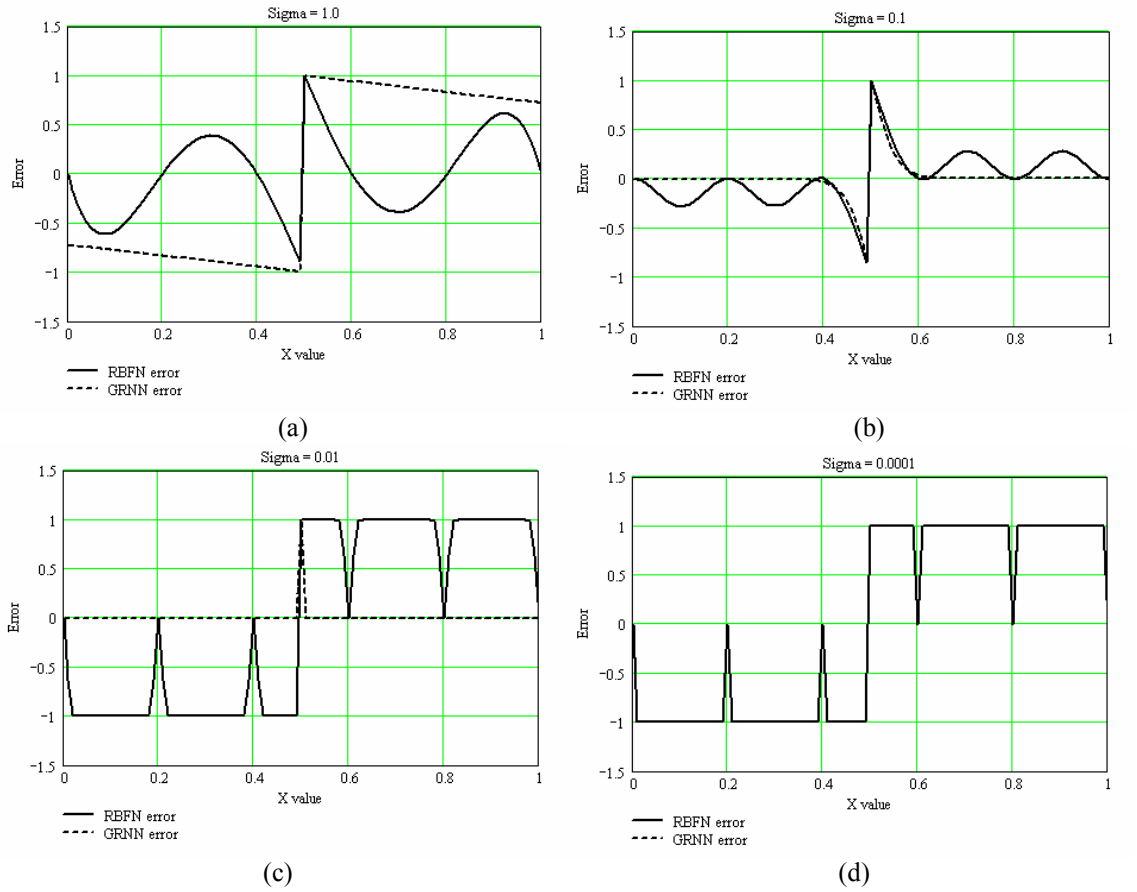


Figure 6.27: The errors for the GRNN and RBFN results shown in Figure 6.25, for (a) $\sigma = 1.0$, (b) $\sigma = 0.1$, (c) $\sigma = 0.01$, and (d) $\sigma = 0.0001$.

The results of these two tests lead us to three conclusions. First, the RBFN method is better able to predict smooth functions such as a sine wave curve than the GRNN method. Second, the GRNN method is better able to predict discontinuous function such as the step function than the RBFN method. Third, an optimized value of sigma is crucial to the success of both methods. One last point that should be made is that no pre-whitening (the λ term in equation (6.29)) was used in any of these examples, since we were dealing with noise-free analytical functions. In the real data case, I have found that up to 10% prewhitening is needed in order to stabilize the solution.

6.9 Comparison of the RBFN and the GRNN methods on a real data example

I will now illustrate our methodology using the channel sand case study that was discussed in sections 5.5 and 6.4.2. As I have already fully discussed this case study, the main objective in this section will be to compare the RBFN and GRNN results. For reference, the base map showing the twelve wells used in this study, the inverted seismic data for line 95, and the impedance dataslice were shown in Figures 5.23 through 5.25, respectively. The multi-linear attribute training result was shown in Table 6.1. In this analysis, we used all twelve wells in the training, a convolutional operator length of seven points, and seven attributes.

Figure 6.28 shows the result of applying the GRNN algorithm the prediction of P-wave velocity using the first six attributes from Table 6.1. In Figure 6.28, which shows only four of the twelve wells, notice that we see an excellent fit between the original and modeled logs. As seen at the top of the figure, we get a correlation coefficient of 0.869 and an average error of 179 m/s, which is an excellent fit.

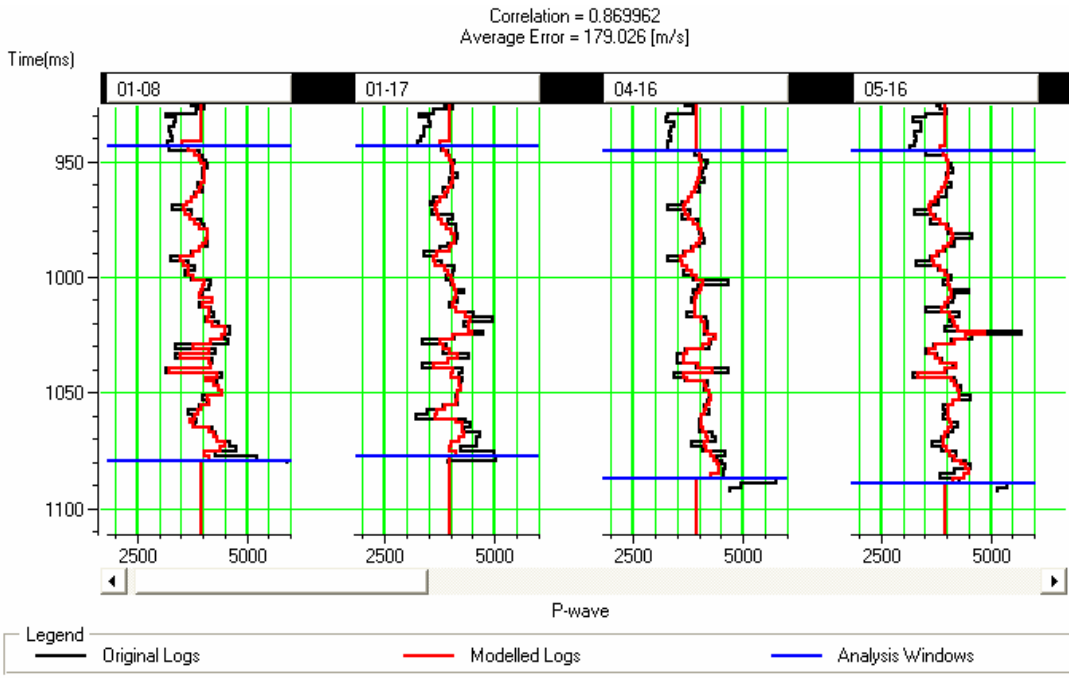


Figure 6.28. Application of the GRNN algorithm to four of the P-wave sonic logs in the twelve log suite, using all the training samples in the prediction.

Figure 6.29 then shows the result of applying the RBFN algorithm to the same set of four attributes used in the GRNN approach, again using all twelve wells in the training. The result is close to that seen in Figure 6.28, although the fit is not quite as good, either visually or in terms of the correlation coefficient (0.8236) or the average error (196 m/s).

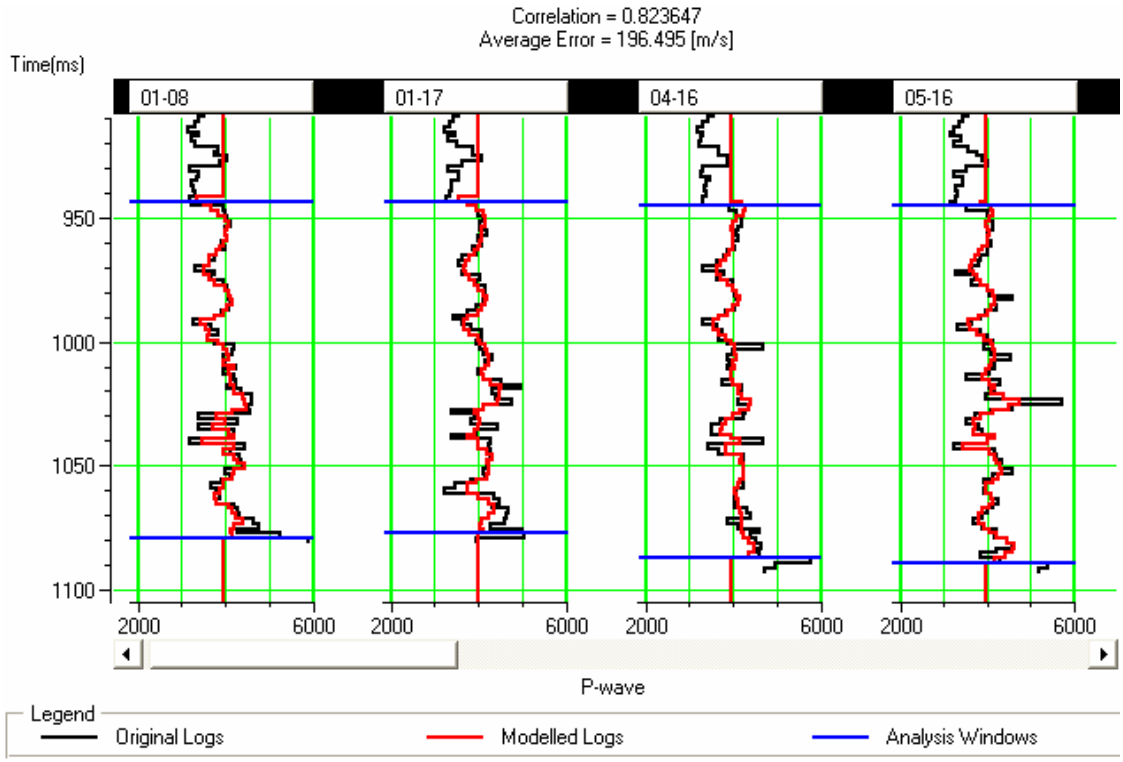


Figure 6.29. Application of the RBFN algorithm to four of the P-wave sonic logs in the twelve log suite using all the training samples in the prediction.

Next, we will look at the cross-validation plots, in which we will leave the predicted well out of the training. This is shown in Figure 6.30 for the GRNN algorithm and in Figure 6.31 for the RBFN algorithm. In both plots, the fit is not as good as it was when we used all the wells in the training. Again, we also see a slightly better result for the GRNN algorithm, both visually and analytically. Note that the correlation coefficient for GRNN is 0.5935, compared with a value of 0.535 for RBFN, and the average error is 273 m/s compared with 290 m/s.

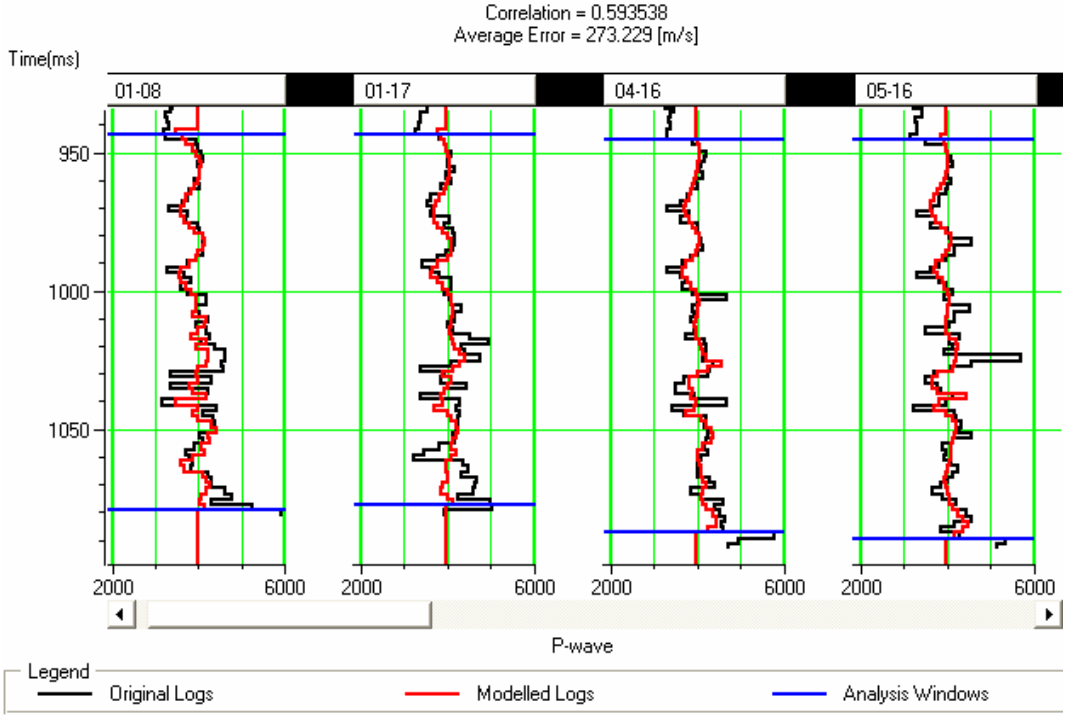


Figure 6.30: Validation of the four P-wave sonic logs from Figure 6.27 using the GRNN algorithm, where the predicted well has been left out of the training.

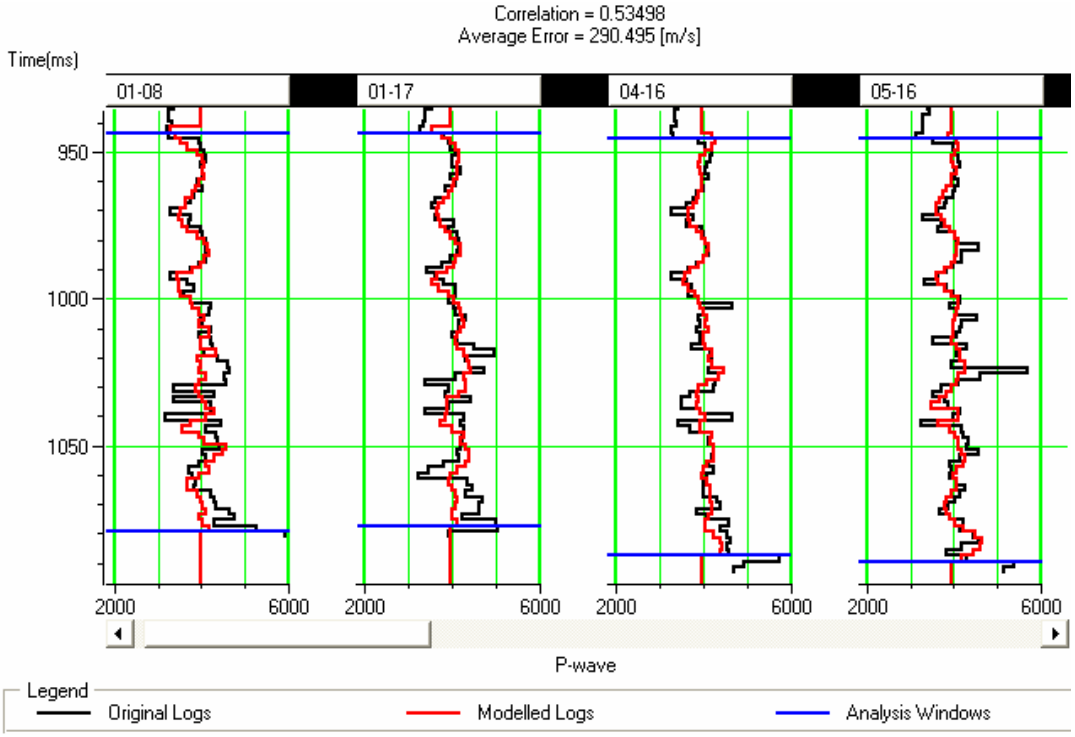


Figure 6.31: Validation of the four P-wave sonic logs from Figure 6.28 using the RBFN algorithm, where the predicted well has been left out of the training.

We will next see how the algorithms compare when we look at the application to real seismic data. Figure 6.32 shows the application of the GRNN algorithm and Figure 6.33 shows the application of the RBFN algorithm.

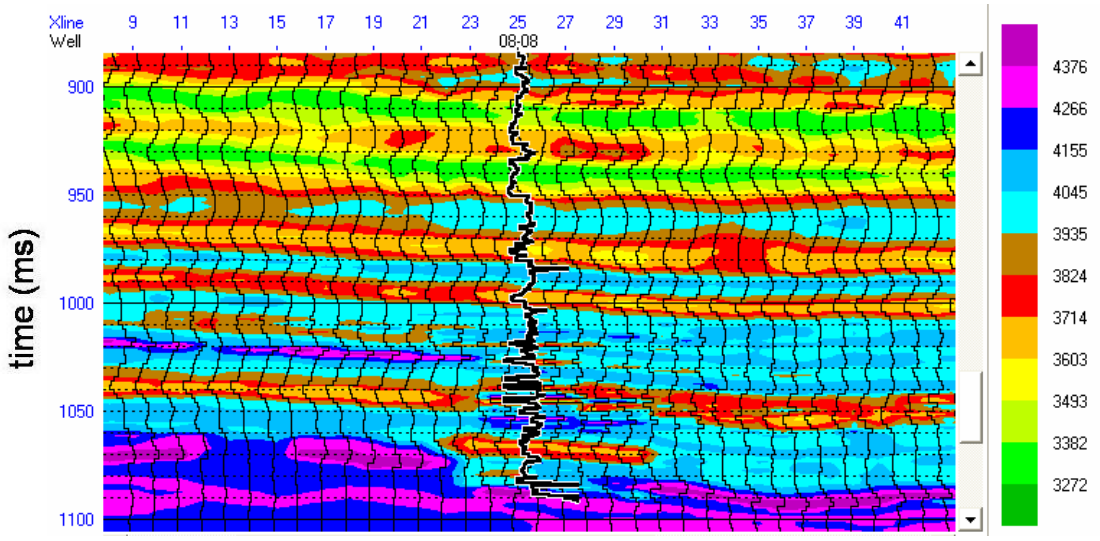


Figure 6.32: Application of the GRNN algorithm to line 95 of the 3D volume, after training using all the wells.

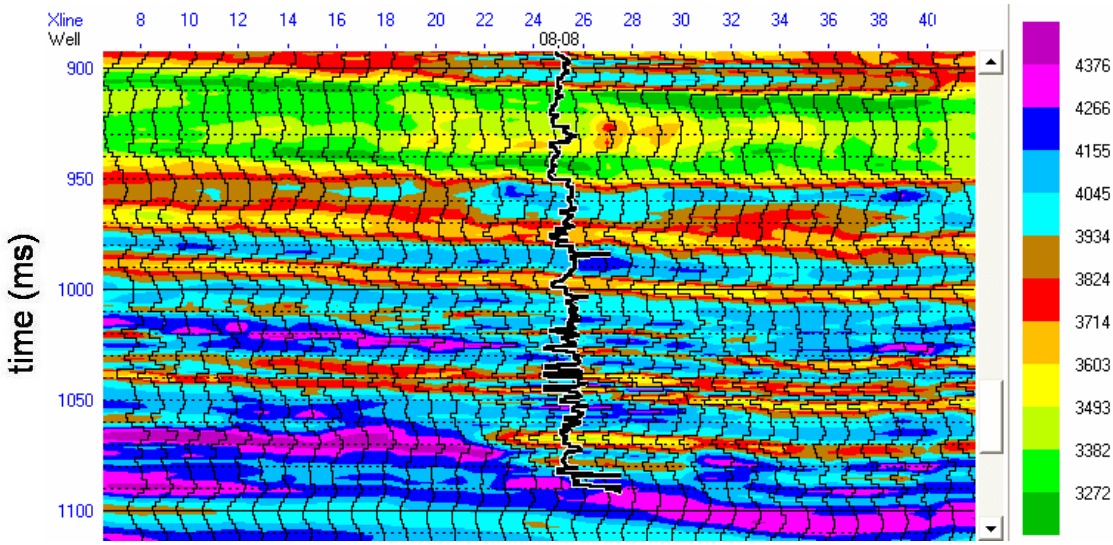


Figure 6.33: Application of the RBFN algorithm to line 95 of the 3D volume, after training with all the wells.

It is clear from Figures 6.32 and 6.33 that the RBFN approach has produced a higher frequency result than the GRNN approach. Also, this high frequency content appears to be realistic, since there is lateral continuity to the extra events that appear on the section.

Next, I will repeat the same analysis, but using only three wells in the training. Table 6.3 and Figure 6.34 shows the results of the training, where the table shows the best five attributes chosen by the step-wise regression algorithm discussed earlier, and the figure shows that only the first four attributes are statistically significant, based on cross-validation. Note that the error on the last attribute shoots up almost vertically. It is therefore important to use only the first four attributes.

.	Target	Final Attribute	Training Error	Validation Error
1	P-wave	Integrate	177.065432	223.717443
2	P-wave	Filter 15/20-25/30	140.606764	162.306769
3	P-wave	Amplitude Weighted Phase	134.536481	146.315146
4	P-wave	Amplitude Weighted Frequency	130.807254	140.102127
5	P-wave	Integrated Absolute Amplitude	127.647020	4205.358509

Table 6.3: The list of attributes determined by cross-validation analysis of multilinear regression.

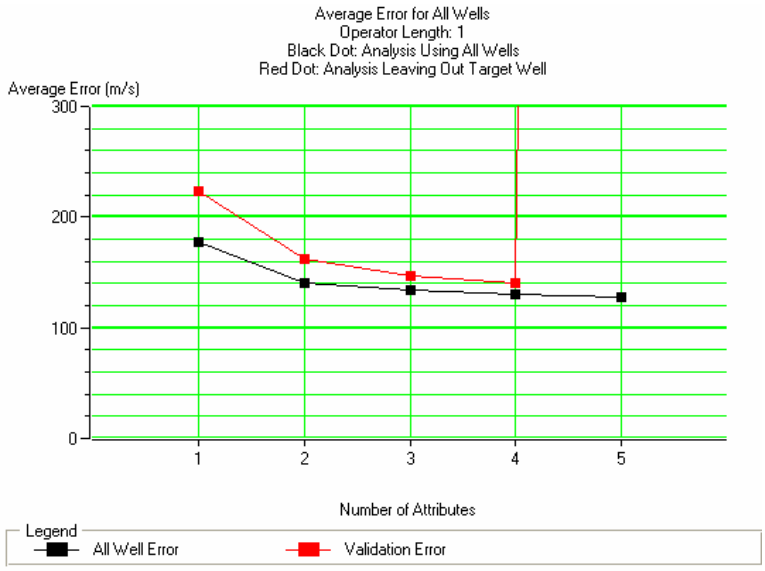


Figure 6.34: The error results of the multi-attribute training.

I will now look at the results of applying the training from all three wells to the wells themselves, shown in Figure 6.35 for GRNN and in Figure 6.36 for RBFN.

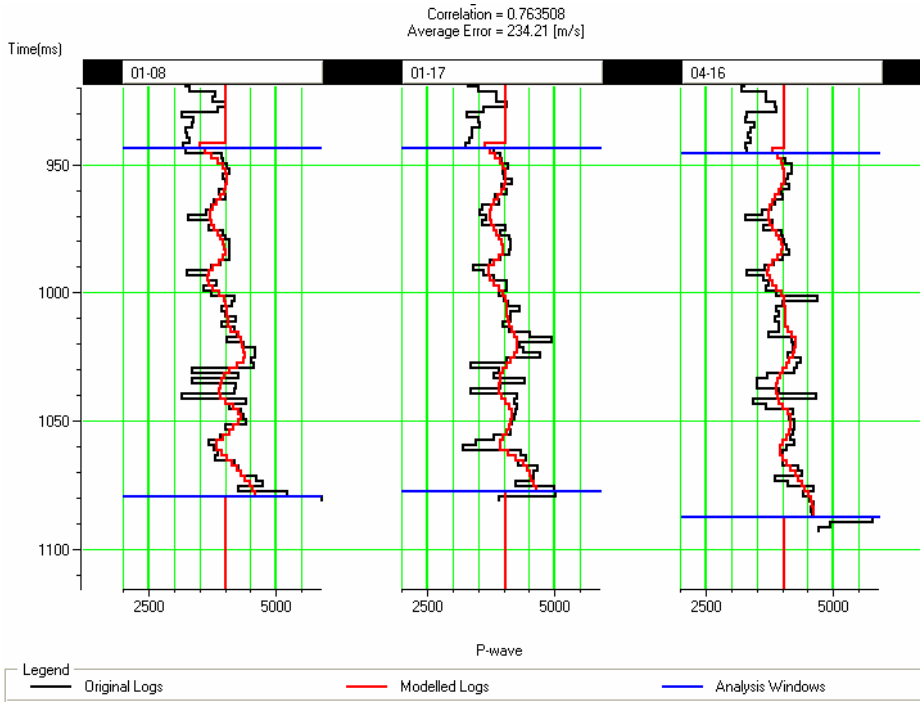


Figure 6.35: Application of the GRNN algorithm to the three P-wave sonic logs used in the training, where all the training samples are used in the prediction.

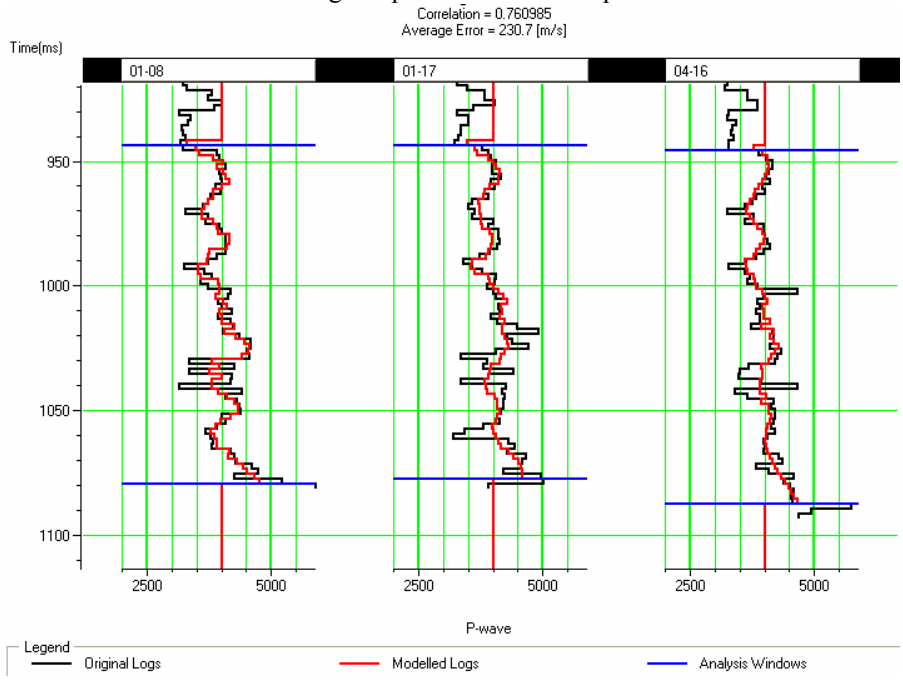


Figure 6.36: Application of the RBFN algorithm to the three P-wave sonic logs used in the training, where all the training samples are used in the prediction.

Note in the previous figures that the visual match is almost the same for both methods, as are the correlation coefficients and average error. The RBFN algorithm has done slightly better than the GRNN algorithm in the average error (230.7 m/s vs 234.21 m/s) and slightly worse for the correlation coefficient (0.760 vs 0.763).

I will now look at the validation results for the three well results, shown in Figure 6.37 for the GRNN case, and in Figure 6.28 for the RBFN case. RBFN has definitely done better than GRNN, with an average error of 259.58 m/s against 263.89, and a correlation coefficient of 0.681 against 0.667 for GRNN.

Finally, we will apply the results of the training to the 3D seismic dataset itself. Figure 6.39 shows the application of the GRNN method to the seismic data. Notice that although the continuity in the events of the result is quite good, the frequency content is very low. This suggests that not enough training points were used for this method to be effective.

When we apply the RBFN method, as shown in Figure 6.40, much more high frequency detail has come through. Also, the lateral continuity of the events is still very good. I would therefore conclude that, as the number of wells in our training dataset goes down, the RBFN algorithm becomes preferable to the GRNN algorithm.

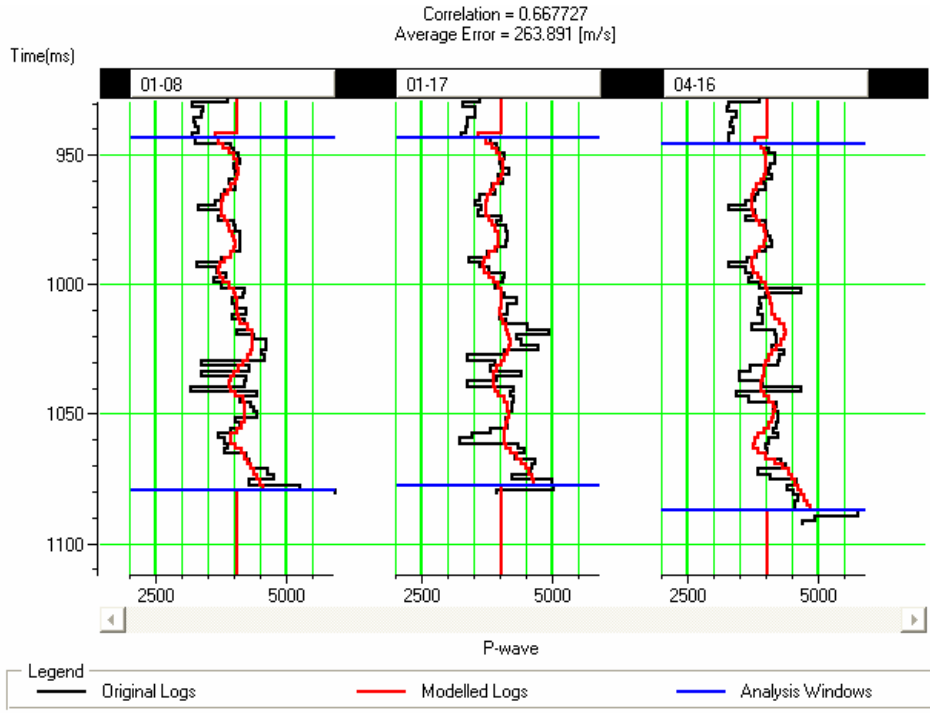


Figure 6.37: Validation of the GRNN algorithm to the three P-wave sonic logs used in the training, where the each log has been successively left out of the training.

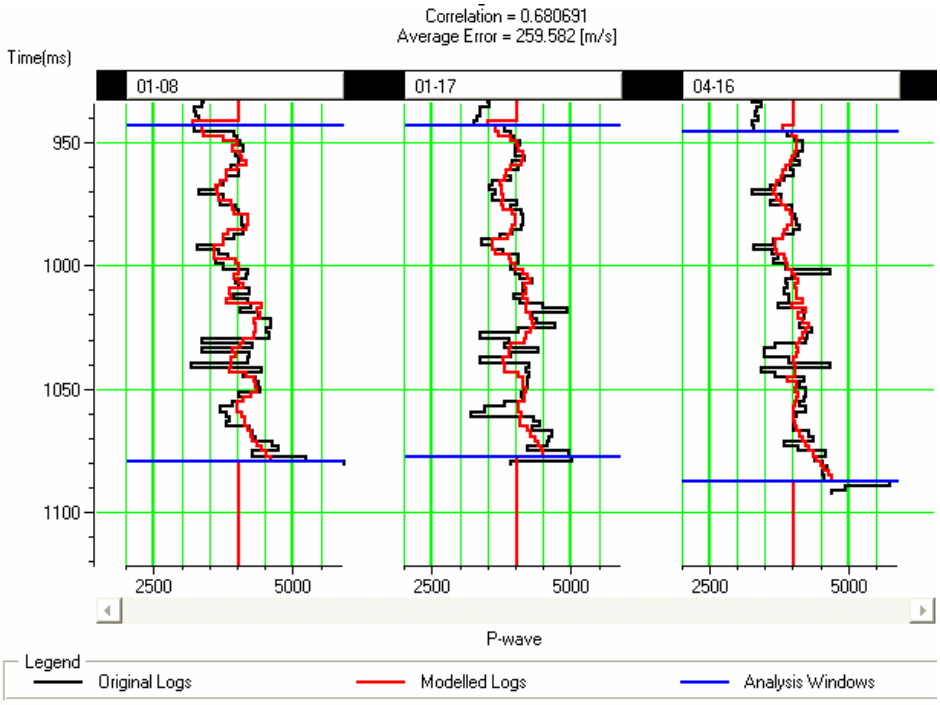


Figure 6.38: Validation of the RBFN algorithm to the three P-wave sonic logs used in the training, where the each log has been successively left out of the training.

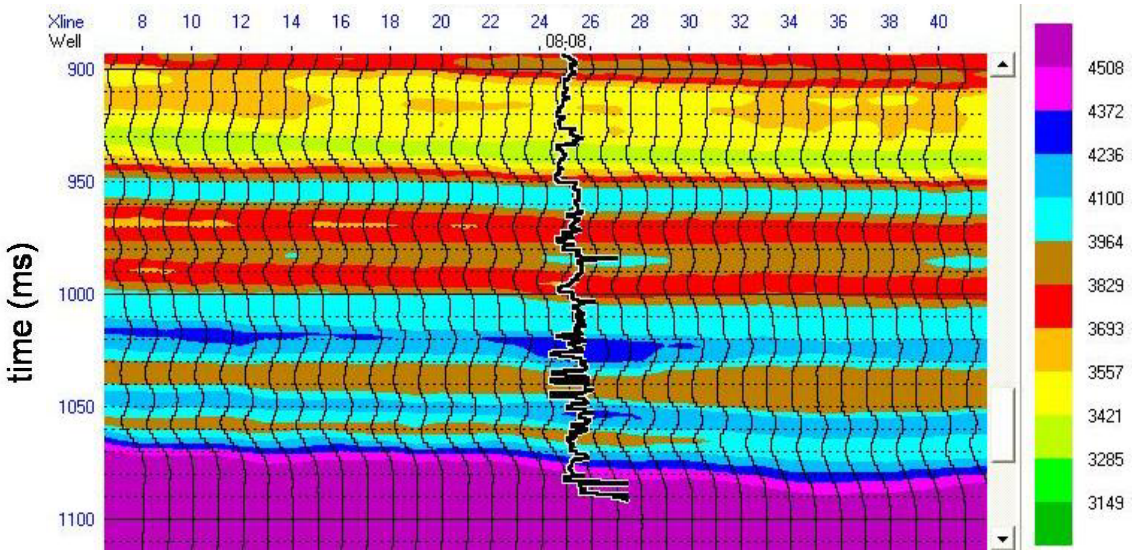


Figure 6.39: Application of the GRNN algorithm to line 95 of the 3D volume, after training using only three of the wells.

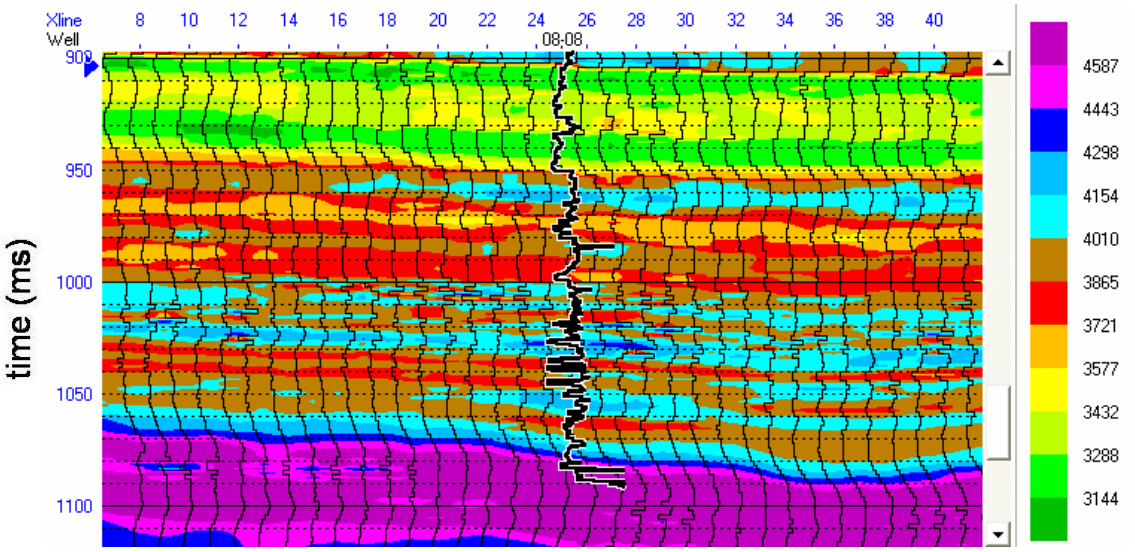


Figure 6.40: Application of the RBFN algorithm to line 95 of the 3D volume, after training using only three of the wells.

6.10 Conclusions

In this chapter, I have discussed several neural network approaches for the classification and prediction of log properties using multiple seismic attributes. These methods consisted of the probabilistic neural network, or PNN, the generalized regression neural network, or GRNN, and the radial basis function neural network, or RBFN. These methods are based on Gaussian basis functions of distance in attribute space. The PNN approach uses these Gaussian basis functions to implement Bayes' Theorem for event classification. I applied the PNN classification technique to the porosity example considered in Chapter 4, and found improved results over the Fisher linear discriminant.

The GRNN and RBFN methods are also built from Gaussian basis functions. The key difference between the two methods is that the GRNN prediction is a weighted sum of the basis functions and the training values, whereas in the RBFN the weights are pre-computed by generalized matrix inversion of the basis functions. The GRNN approach has its origins in the statistical theory of Parzen density estimation, whereas the RBFN method can be thought of as a nonlinear extension of the linear regression methods considered in Chapter 3.

I then applied the GRNN and RBFN methods to a channel sand case study from Alberta. We did the training using all twelve wells in the study area, and then using a subset of only three wells. Our conclusion is that as the number of wells in the training dataset goes down RBFN provides a better technique for predicting log properties from seismic attributes, both in the fit at the training wells and in the application to the seismic. As the number of wells increases, the two methods produce fairly consistent results.

In the next chapter, I will consider an alternate form of the RBFN method that uses a subset of points from the training dataset, called basis centres. I will also extensively discuss the problem of finding the basis centres.

CHAPTER 7 : RBF NETWORKS WITH BASIS CENTRES

7.1 Introduction

In the radial basis function neural network (RBFN) approach discussed in the last chapter, I designed a neural network operator by computing the basis functions for all the possible distances between the N points in our training dataset. This involved the inversion of an NxN dimensional matrix, which became costly as N got very large, due to space and time limitations. This approach is often called the strict interpolation RBFN, since all the training points are used to calculate the weights. An alternate approach to strict interpolations RBFN is to reduce the number of input samples to a subset of K values, often called centres, and compute the basis functions of the distances between the N input points and these K centres. Although this results in a non-square NxK matrix, this matrix can be inverted using the generalized inverse approach, and results in the inversion of a KxK, greatly reducing space requirements and speeding up the computation.

In this chapter, I will first describe the theory of RBFN with basis centres and then show an example using the AVO classification problem. I will then discuss an approach to finding the basis function centres using K-means clustering, and discuss how to apply K-means clustering methods to the RBFN with centres neural network. This will be illustrated with a simple numerical example, and then applied to the prediction of P-wave velocity over a channel sand. In this chapter, I will also introduce an extension to the K-means clustering method, called the Mahalanobis clustering method, and apply this method to an AVO crossplot example.

7.2 Theory of RBF networks with basis centres

In the full interpolation RBFN approach, recall that we wish to solve for the N weights w_j given the N pairs of attribute vectors s_i and the N training values t_i, as shown in Figure 7.1.

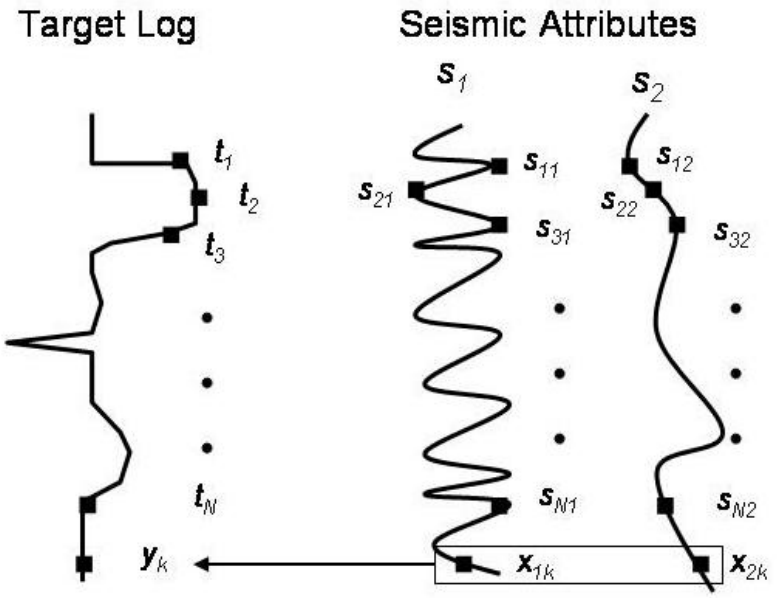


Figure 7.1: The attribute vectors and target values used in the RBFN method.

The solution is found by solving the set of equations

$$t(\boldsymbol{s}_i) = \sum_{j=1}^{N} w_j \phi_{ij}, \quad i = 1, 2, \ldots, N, \tag{7.1}$$

where the ϕ_{ij} functions are the Gaussian basis functions given by

$$\phi_{ij} = \exp\left[-\frac{\left| \boldsymbol{s}_i - \boldsymbol{s}_j \right|^2}{\sigma^2} \right]. \tag{7.2}$$

Equation 7.1 can be written more compactly as the N x N matrix equation

$$\boldsymbol{t} = \boldsymbol{\Phi}\, \boldsymbol{w}, \tag{7.3}$$

where $\boldsymbol{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$ is a vector containing the target values, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix}$ is a vector containing the weights, and $\boldsymbol{\Phi} = \begin{bmatrix} \phi_{11} \cdots \phi_{1N} \\ \vdots \ddots \vdots \\ \phi_{N1} \cdots \phi_{NN} \end{bmatrix}$ is an N x N matrix containing the basis functions. The solution to equation (7.3) is given by the matrix inverse

$$\boldsymbol{w} = \left[\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right]^{-1} \boldsymbol{t}, \tag{7.4}$$

where λ is a pre-whitening factor and I is the identity matrix. Once the weights have been computed, they are applied to the full dataset using the equation

$$y(\boldsymbol{x}_k) = \sum_{j=1}^{N} w_j \exp\left[\frac{\left|\boldsymbol{x}_k - \boldsymbol{s}_j\right|^2}{\sigma^2}\right]. \tag{7.5}$$

The solution to equation (7.4) is very time-consuming and possibly unstable as N becomes very large. To solve for the instability, a large prewhitening value λ is often needed.

The solution to the problem of large N is to reduce the number of linear equations by defining a modified version of equation (7.1) given by

$$t(\boldsymbol{s}_i) = \sum_{k=1}^{K} w_k \exp\left[-\frac{\left|\boldsymbol{s}_i - \boldsymbol{\mu}_k\right|^2}{\sigma^2}\right], \quad i = 1, 2, \ldots, N, \tag{7.4}$$

where the $\boldsymbol{\mu}_k$ terms are a set of K cluster centres, or means, and $K << N$. Notice that these are vector-valued means, with a dimension equal to M, the number of attributes. Equation (7.4) can be written in expanded form as the set of linear equations given by

$$\begin{aligned} t_1 &= w_1 \phi_{11} + w_2 \phi_{12} + \ldots + w_{1K} \phi_{1K} \\ t_2 &= w_1 \phi_{21} + w_2 \phi_{22} + \ldots + w_{2K} \phi_{2K} \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ t_N &= w_1 \phi_{N1} + w_2 \phi_{N2} + \ldots + w_{NK} \phi_{NK} \end{aligned} \tag{7.5}$$

which can again be written as the matrix equation

$$\boldsymbol{t} = \boldsymbol{\Phi}\,\boldsymbol{w}, \tag{7.6}$$

where $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix}, \boldsymbol{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}, \Phi = \begin{bmatrix} \phi_{11} & \cdots & \phi_{1K} \\ \vdots & \ddots & \vdots \\ \phi_{N1} & \cdots & \phi_{NK} \end{bmatrix}$. In many applications (such as the one shown in the next section) a zero weight w_0 is added to each linear equation. This weight can be implicitly assumed in equations (7.5) and (7.6) if we let the first row of ϕ values all equal one. Equation (7.6) represents an over-determined system, in which we have more observations than unknowns. The solution to this problem is given by the least-squares Moore-Penrose inverse

$$\boldsymbol{w} = \left[\Phi^T\Phi + \Lambda\right]^{-1}\Phi^T\boldsymbol{t}, \tag{7.7}$$

where Λ is the K x K regularization matrix given by $\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_K \end{bmatrix}$. The λ_k terms are called regularization parameters. In the simplest case, the regularization terms are all equal to the value λ, and $\Lambda = \lambda I$, where I is the K x K identity matrix. In neural network terminology, Φ is called the design matrix, and the term $[\Phi^T\Phi + \Lambda]$ is called the covariance matrix.

The crucial question is how to determine the basis centres μ_k in equation (7.4). This is illustrated in Figure 7.2 for the simple case of two attributes with 18 points. As can be seen in Figure 7.2, these points separate naturally into 4 clusters and we have replaced each cluster of points with its mean, resulting in $K = 4$. Before discussing the K-means method, and showing how the clusters in Figure 7.2 can be found, I will revisit the AVO classification problem and show how it can be solved using RBFN with basis centres.

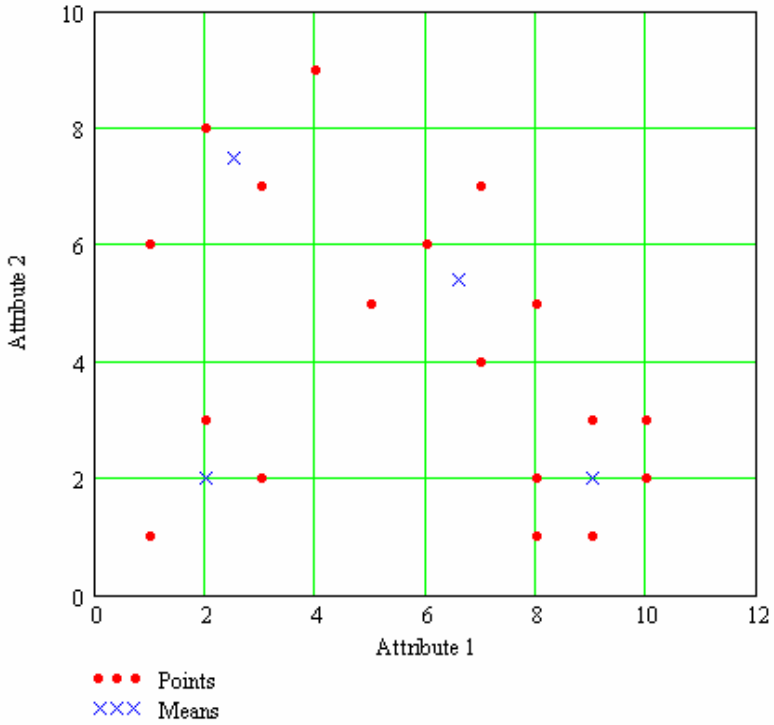


Figure 7.2: Computing the means for a 4 cluster, 18 point example.

7.3 An AVO classification example

In Chapters 4 and 5, I used the multilayer perceptron to differentiate between a class 3 gas sand and a wet sand on an AVO crossplot. Recall that the Aki-Richards equation as given in equation (2.19) of section 2.7 is given by

$$R(\theta) = A + B\sin^2\theta, \tag{7.8}$$

where A is the intercept, and B is the gradient.

Using the values for V_P, V_S, and density ρ shown in Figure 4.6 of section 4.5.2, I computed values for A and B, scaled by a factor of 10 (to give values of +1 and -1) and created the A-B crossplot shown in Figure 7.3(a). The objective is to separate the wet points from the gas points but, as shown in the figure, there is a nonlinear decision boundary between these points.

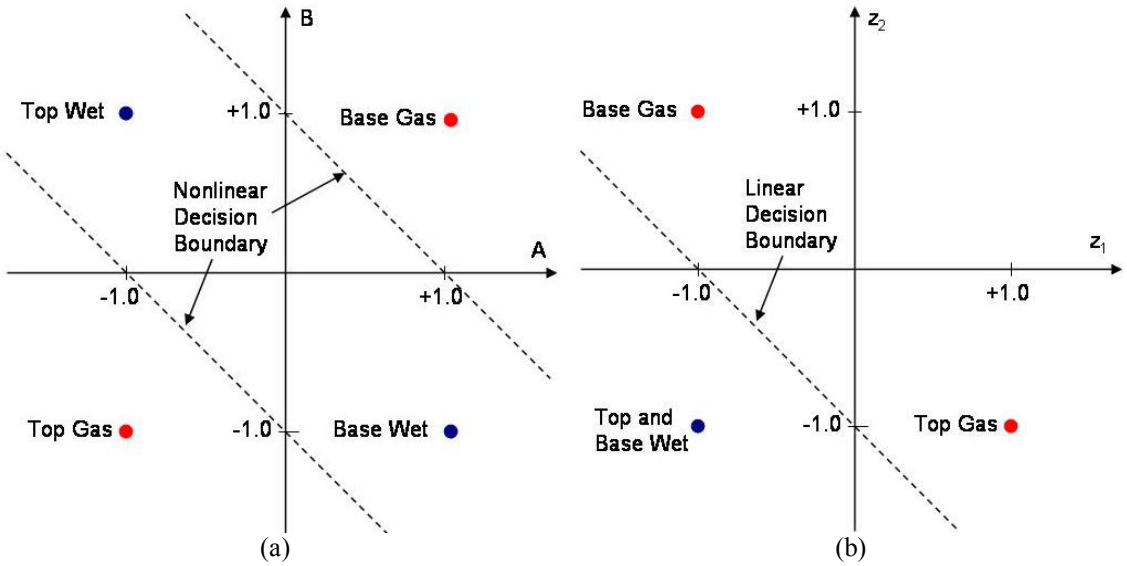


Figure 7.3. Intercept versus gradient crossplot from the wet and gas models of Figure 4.6, both (a) before and (b) after application of the multi-layer perceptron.

Since the gas sand points shown in Figure 7.3(a) were not linearly separable, I found that a single layer perceptron could only solve for either the top or base of the sand, not for both simultaneously. By applying the multi-layer perceptron we were able to transform the values shown in Figure 7.3(a) into the space shown in Figure 7.3(b), in which the top and base of the sand were linearly separable. However, recall that the solution to the problem was quite involved, as it requires the nonlinear optimization of two layers of weights.

Let us now use the RBFN with centres method to solve the same problem. Note that the four input vectors can be written

$$\boldsymbol{x}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad \boldsymbol{x}_2 = \begin{bmatrix} +1 \\ -1 \end{bmatrix}, \quad \boldsymbol{x}_3 = \begin{bmatrix} -1 \\ +1 \end{bmatrix}, \quad \boldsymbol{x}_4 = \begin{bmatrix} +1 \\ +1 \end{bmatrix}.$$

where points $\boldsymbol{x}_1$ and $\boldsymbol{x}_4$ are the top and base of the gas zone and points $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ are the top and base of the wet zone. I will define two centres using the gas sand values of

$$\boldsymbol{c}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad \boldsymbol{c}_2 = \begin{bmatrix} +1 \\ +1 \end{bmatrix}.$$

I have used the symbol $\boldsymbol{c}$ rather than $\boldsymbol{\mu}$ for the centres since these are not the mean values. Thus, equation (7.4) can be rewritten as

$$\phi_{ij} = \phi_j(\boldsymbol{x}_i) = \exp\left[-\frac{\left|\boldsymbol{x}_i - \boldsymbol{c}_j\right|^2}{\sigma^2}\right]. \tag{7.9}$$

Expanding the vector components for the two-dimensional vectors we can write

$$\phi_{ij} = \exp\left[-\frac{(x_{i1} - c_{j1})^2 + (x_{i2} - c_{j2})^2}{\sigma^2}\right]. \tag{7.10}$$

From the definitions of our input vectors and centers, we can see that there are only three possible distances $d = \left|\boldsymbol{x}_i - \boldsymbol{c}_j\right|^2$. If we set the value of σ equal to *1*, the values of ϕ_{ij} are $\phi_{11} = \phi_{42} = \exp[-0] = 1.0$, $\phi_{12} = \phi_{41} = \exp[-8] = 0.0003$, and $\phi_{21} = \phi_{22} = \phi_{31} = \phi_{32} = \exp[-4] = 0.018$. Grouping the terms, we find that

$$\boldsymbol{\phi}_1 = \begin{bmatrix} \phi_{11} \\ \phi_{21} \\ \phi_{31} \\ \phi_{41} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.018 \\ 0.018 \\ 0.0003 \end{bmatrix},$$

and

$$\boldsymbol{\phi}_2 = \begin{bmatrix} \phi_{12} \\ \phi_{22} \\ \phi_{32} \\ \phi_{42} \end{bmatrix} = \begin{bmatrix} 0.0003 \\ 0.018 \\ 0.018 \\ 1.0 \end{bmatrix}.$$

Figure 7.4(a) shows the original AVO problem in *A-B* space, and Figure 7.4(b) shows a crossplot of $\phi_1(x_i)$ versus $\phi_2(x_i)$. Notice that we have now achieved a linear separation between the gas sand values and the wet sand values. Next, I will use the radial basis function neural network to compute the weights.

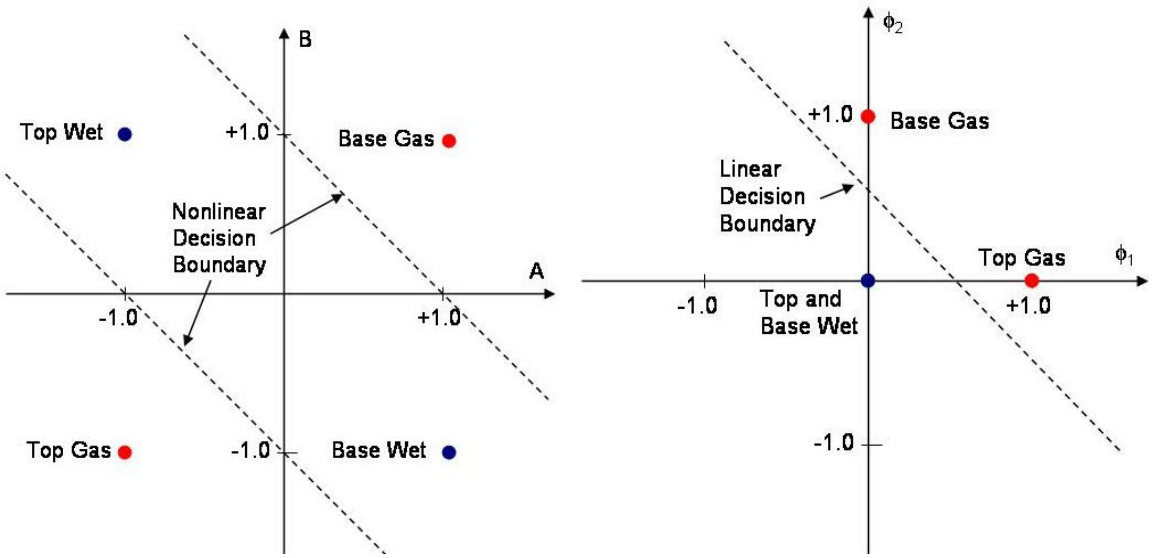


Figure 7.4: The AVO classification problem, where (a) shows the original problem in intercept-gradient space and (b) shows the same problem in ϕ_1-ϕ_2 space.

Figure 7.5 shows how the concepts of the previous section can be applied in neural network form. Note that this is a similar diagram to the MLP except for two important differences. First, there are no weights between the inputs and the application of the ϕ functions in the first layer neurons. Second, although there are weights and a bias value leading to the second layer neuron, there is no application of a nonlinear function at this neuron. In other words, the second layer is simply a linear sum.

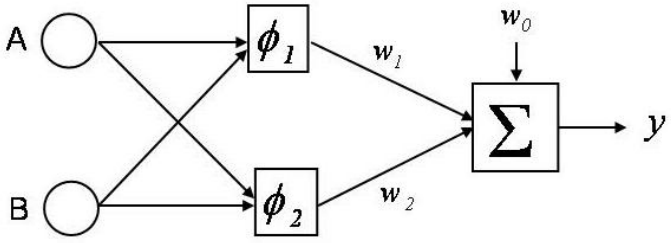


Figure 7.5: The radial basis function neural network implementation of the AVO classification problem.

In this case we can write the RBFN mathematically as

$$t(\boldsymbol{x}_i) = w_0 + \sum_{j=1}^{2} w_j \phi_j(\boldsymbol{x}_i), \quad i = 1, \ldots, 4. \tag{7.11}$$

To solve for the weights, I set each output value to the training data values given in Table 1, where *+1* indicates the presence and *-1* indicates the absence of gas.

i	A_i	B_i	ϕ_{i1}	ϕ_{i2}	t_i
1	-1	-1	1	0.0003	+1
2	+1	-1	0.018	0.018	-1
3	-1	+1	0.018	0.018	-1
4	+1	+1	0.0003	1	+1

Table 7.1. The input values and basis function values for the AVO classification problem.

Thus, we have four linear equations with three unknowns, which can be written as the matrix equation

$$\boldsymbol{t} = \boldsymbol{\Phi}\, \boldsymbol{w}, \tag{7.12}$$

where $\boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$, $\Phi = \begin{bmatrix} 1 & \phi_{11} & \phi_{12} \\ 1 & \phi_{21} & \phi_{22} \\ 1 & \phi_{31} & \phi_{32} \\ 1 & \phi_{41} & \phi_{42} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0.003 \\ 1 & 0.018 & 0.018 \\ 1 & 0.018 & 0.018 \\ 1 & 0.003 & 1 \end{bmatrix}$, and $\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$.

Rewriting equation (7.7) using a constant regularization of prewhitening term, we get

$$\boldsymbol{w} = \left[\Phi^T \Phi + \lambda I\right]^{-1} \Phi^T \boldsymbol{t}, \tag{7.13}$$

where we can expand the terms to get $\Phi^T \Phi = \begin{bmatrix} 4 & 1.039 & 1.039 \\ 1.039 & 1.001 & 0.007 \\ 1.039 & 0.007 & 1.001 \end{bmatrix}$, and

$\Phi^T \boldsymbol{t} = \begin{bmatrix} 0 \\ 0.967 \\ 0.967 \end{bmatrix}$. Using zero pre-whitening ($\lambda = 0$) the solution is

$$\boldsymbol{w} = \begin{bmatrix} -1.074 \\ 2.068 \\ 2.068 \end{bmatrix}.$$

Thus, the intercepts on both the A and B axes are given by the ratio

$$-w_0/w_1 = -w_0/w_2 = -(-1.074 / 2.068) = 0.519,$$

which are the points where the dashed line in Figure 7.4(b) intersects each axis.

I have therefore shown how to construct a linear separation boundary for our AVO classification problem using the radial basis function neural network, and found that the solution is actually more straightforward than the multi-layer perceptron approach. However, our real interest is in the solution of problems in which the number of points is very large. This will involve using a clustering algorithm to compute the RBFN centres.

7.4 Data clustering methods

7.4.1 Introduction to clustering

Clustering has been long been used in the field of multivariate statistics (Johnson and Wichern, 1998), but has only recently found application in seismic analysis. In clustering, we seek to find natural groupings, or clusters, within a particular dataset. These clusters could pertain to different lithologies, fluids, etc. Although closely related to classification, clustering is more basic in that we normally do not know how many clusters we have, or what form these clusters will take. Classification is usually supervised by the knowledge of what our classes should look like, whereas clustering is unsupervised, without any such knowledge.

A key question is where we should do the clustering. In this chapter, I will apply the clustering in multi-dimensional parameter space. The simplest example of this is two-dimensional space, and our study will involve A-B crossplots from the AVO technique (Ross, 2000). The advantage of using two-dimensional space is that it is easily visualized and the results can be checked by eye. The method can also be applied to higher dimensional spaces, which cannot be visualized. For this example, I chose multi-

attribute space, in which multiple seismic attributes are grouped and used to predict reservoir parameters (Hampson et al., 2001). In the following discussion, I will refer to the group of L attributes for a single recorded time sample in L-dimensional space as an attribute vector. In general, there will be N recorded time samples.

I will start by considering the simplest clustering approach, called K-means clustering (Bishop, 1996). This method is based on the Euclidean distance between points. I will show that this method works well for the case of well-separated, roughly spherical clusters, but not when the clusters become elliptical in shape. I then propose a new method called Mahalanobis clustering, in which statistical distance, rather than Euclidean distance, is used for the clustering. Although the use of this distance metric was discussed by Johnson and Wichern (1998), they state that it is not used in practice, and do not discuss a method for implementing this procedure.

7.4.2 K-means clustering

In the K-means clustering technique, I start with a random estimate of the cluster centres, and iterate toward a solution by minimizing the distance from each input cluster centre to the points surrounding it. As pointed out by Haykin (1999) this has the desirable property of placing the centres of the clusters in those regions of the input space where significant amounts of data are present. The steps involved in K-means clustering are as follows:

1. Decide on a number of clusters, K, and divide the input data points randomly into these K clusters. If we have N L-dimensional input vectors of attributes, $\boldsymbol{x_i}$, we initially set the number of points in each of the first K-1 clusters to $N_k = int(N/K)$, and the last cluster equal to $N - (N_k*(K-1))$. Also, the decision of what value to assign to K is important and will affect the result.

2. Compute the means $\boldsymbol{\mu_k}$, where $k = 1, 2, \ldots, K$. The means are simply the sum of the N_k attribute vectors divided by N_k, where N_k is the number of points in the k^{th} cluster.
3. Compute the matrix of distances $d_{ik} = | \boldsymbol{x_i} - \boldsymbol{\mu_k} |$, and assign each input attribute vector to the cluster for which this distance is a minimum. Note that the N_k values are now updated.
4. Re-compute the means based on the new cluster assignments
5. Iterate through steps 2-4 until convergence.

Obviously, the *K*-means clustering algorithm can be computed for any number of input points, attributes and clusters. Before considering any real data application, let us consider the following two-dimensional numerical example in which we have the eighteen input vectors given by:

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 8 \\ 1 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} 4 \\ 9 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \boldsymbol{x}_5 = \begin{bmatrix} 8 \\ 2 \end{bmatrix}, \boldsymbol{x}_6 = \begin{bmatrix} 5 \\ 5 \end{bmatrix},$$

$$\boldsymbol{x}_7 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \boldsymbol{x}_8 = \begin{bmatrix} 9 \\ 3 \end{bmatrix}, \boldsymbol{x}_9 = \begin{bmatrix} 6 \\ 6 \end{bmatrix}, \boldsymbol{x}_{10} = \begin{bmatrix} 1 \\ 6 \end{bmatrix}, \boldsymbol{x}_{11} = \begin{bmatrix} 9 \\ 1 \end{bmatrix}, \boldsymbol{x}_{12} = \begin{bmatrix} 7 \\ 7 \end{bmatrix},$$

$$\boldsymbol{x}_{13} = \begin{bmatrix} 2 \\ 8 \end{bmatrix}, \boldsymbol{x}_{14} = \begin{bmatrix} 10 \\ 2 \end{bmatrix}, \boldsymbol{x}_{15} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}, \boldsymbol{x}_{16} = \begin{bmatrix} 3 \\ 7 \end{bmatrix}, \boldsymbol{x}_{17} = \begin{bmatrix} 10 \\ 3 \end{bmatrix}, \boldsymbol{x}_{18} = \begin{bmatrix} 8 \\ 5 \end{bmatrix}.$$

These eighteen vectors are plotted in Figure 7.6, with the input vector number labelled on the graph. Note that we see four distinct clusters, although the order of the input points is random, and bears no relationship to the cluster order.

I will perform the first step of the *K*-means clustering by assuming that we have four clusters, although the optimal number of clusters will not be obvious in a real dataset. This will give us the result that the first three clusters have four points each, and the last cluster has six points.

The resulting means are

$$\mu_1 = \begin{bmatrix} 3.75 \\ 3.5 \end{bmatrix}, \mu_2 = \begin{bmatrix} 6.25 \\ 3 \end{bmatrix}, \mu_3 = \begin{bmatrix} 5.75 \\ 5 \end{bmatrix}, \text{and } \mu_4 = \begin{bmatrix} 6.667 \\ 4.833 \end{bmatrix}.$$

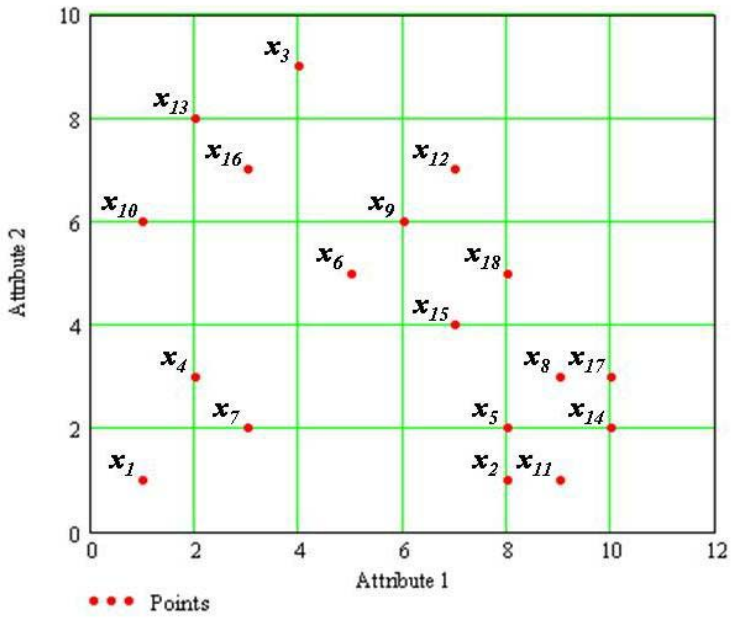


Figure 7.6: The red dots show an set of eighteen points, grouped in four clusters, with the labels indicating the input order of the two-dimensional attribute vectors.

The means of the four clusters are plotted using blue crosses on Figure 7.7(b). (Figure 7.7(a) is simply a repeat of Figure 7.6, for comparison purposes). Since the points were entered in random order, the initial means are grouped together in the centre of the plot and do not define the actual cluster means.

I next re-compute the clusters by finding the points that are closest to the initial means. After this first iteration, we find that the number of points in each cluster is $n_1 = 4$, $n_2 = 6$, $n_3 = 5$, $n_4 = 3$, and their means become

$$\mu_1 = \begin{bmatrix} 1.75 \\ 3 \end{bmatrix}, \mu_2 = \begin{bmatrix} 9 \\ 2 \end{bmatrix}, \mu_3 = \begin{bmatrix} 4 \\ 7 \end{bmatrix}, \text{and } \mu_4 = \begin{bmatrix} 7.333 \\ 5.333 \end{bmatrix}.$$

The new means are shown in Figure 7.7(b). Notice that the means have spread out and become more reasonable and that the second cluster, with six points and a mean of (9, 2), has been computed exactly.

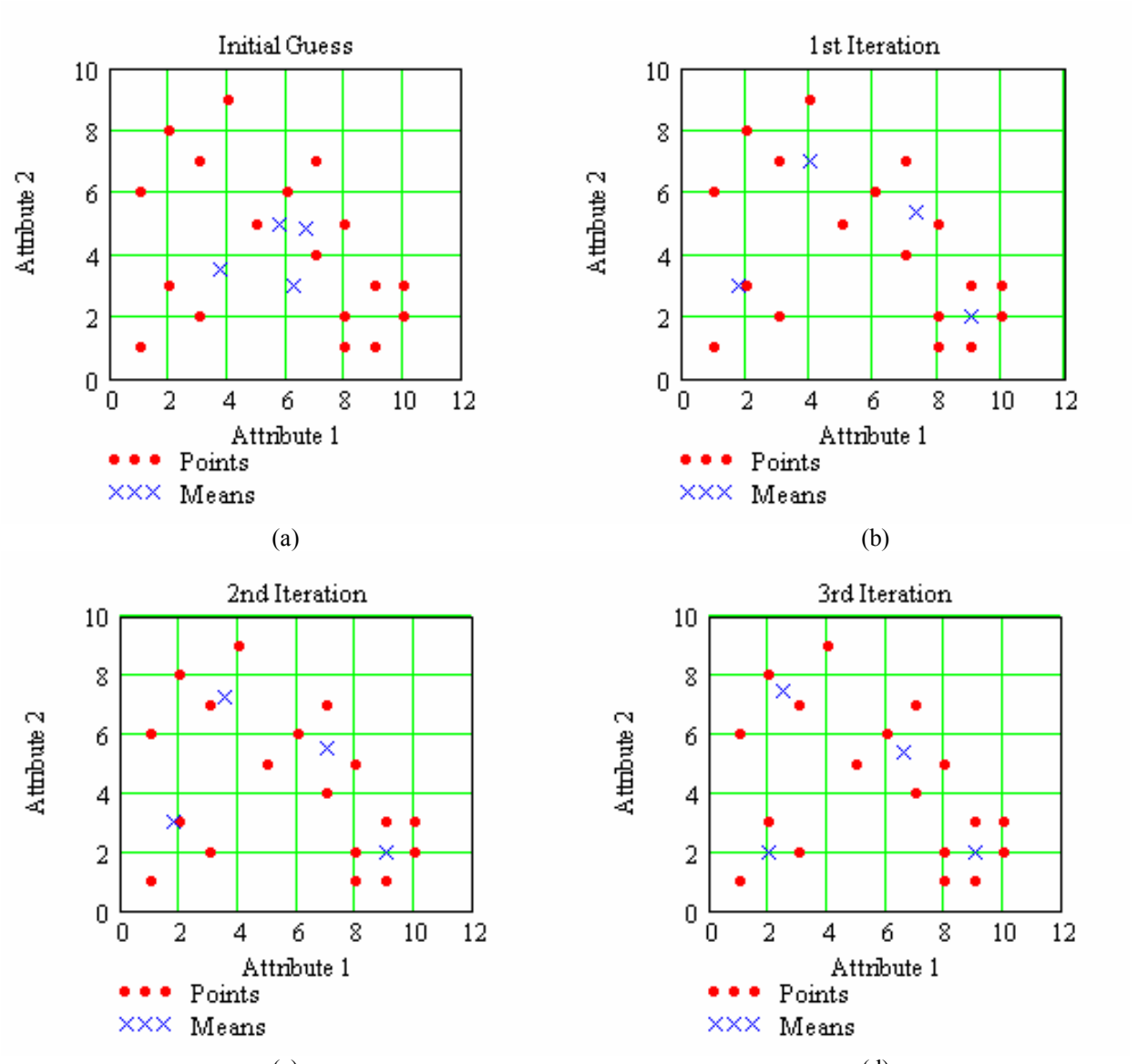


Figure 7.7: The results of applying the K-means clustering example to the example in Figure 7.6, showing (a) the initial calculation, (b) the result of the first iteration, (c) the result of the second iteration, and (d) the result of the third iteration, which is the correct answer.

I next repeat the clustering operation, finding the points that are closest to the new means. The second iteration leads to $n_1 = 4$, $n_2 = 6$, $n_3 = 4$, $n_4 = 4$, with means

$$\mu_1 = \begin{bmatrix} 1.75 \\ 3 \end{bmatrix}, \mu_2 = \begin{bmatrix} 9 \\ 2 \end{bmatrix}, \mu_3 = \begin{bmatrix} 3.5 \\ 7.55 \end{bmatrix}, \text{and } \mu_4 = \begin{bmatrix} 7 \\ 5.5 \end{bmatrix}.$$

The results of the second iteration are shown in Figure 7.7(c). Although cluster 1 hasn't improved, clusters 3 and 4 have moved closer to their correct positions.

The third iteration leads to the values $n_1 = 3$, $n_2 = 6$, $n_3 = 4$, $n_4 = 5$, with means

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 9 \\ 2 \end{bmatrix}, \mu_3 = \begin{bmatrix} 2.5 \\ 7.5 \end{bmatrix}, \text{and } \mu_4 = \begin{bmatrix} 6.6 \\ 5.4 \end{bmatrix}.$$

Comparing these to the known right answers, notice that the results are now correct. This is shown in Figure 7.2(d).

Now I will consider a second example, shown in Figure 7.3. In this case we have created three elliptical clusters trending at about -45 degrees. This is a synthetic example of a class 3 AVO anomaly (Russell et al., 2002b).

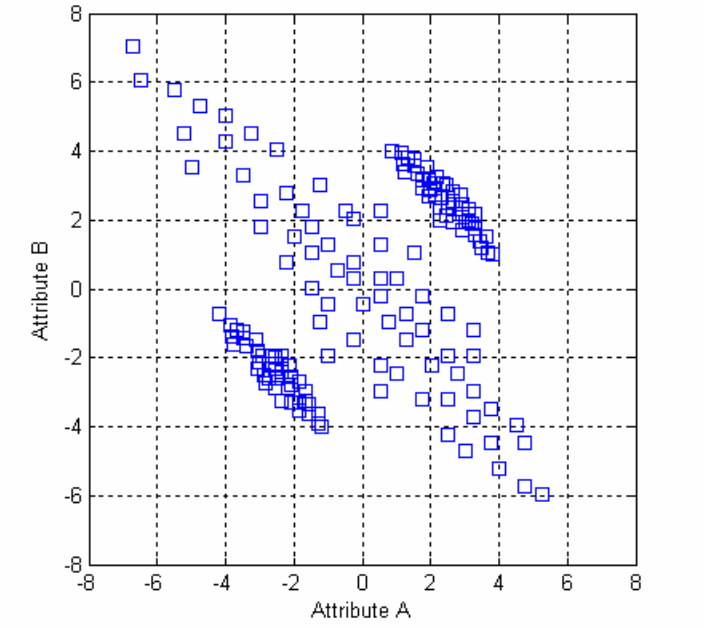


Figure 7.8: A second input dataset to the K-means clustering algorithm. This dataset simulates a typical class 3 AVO A-B crossplot.

It would appear that the job of partitioning this set of points into three clusters would be fairly trivial. The results of 20 iterations of *K*-means clustering on the example shown in Figure 7.8 are shown in Figure 7.9. Figure 7.9(a) shows the classified points, where the blue circles, red squares, and green diamonds show, respectively, the three clusters. Notice that the three obvious clusters from Figure 7.8 have not been correctly classified. The reason for this is shown in Figure 7.9(b), where black dots show the three means, and two circles have been drawn around each of the means. The circles have radii equal to half the distance between each pair of means.

Thus, the *K*-means algorithm has classified points into clusters that fall into circular groups, not the elongated ellipses seen in Figure 7.8. As I shall show in the next section, there is a theoretical reason for this, and we can make use of this theory to develop an algorithm that will correctly classify this example.

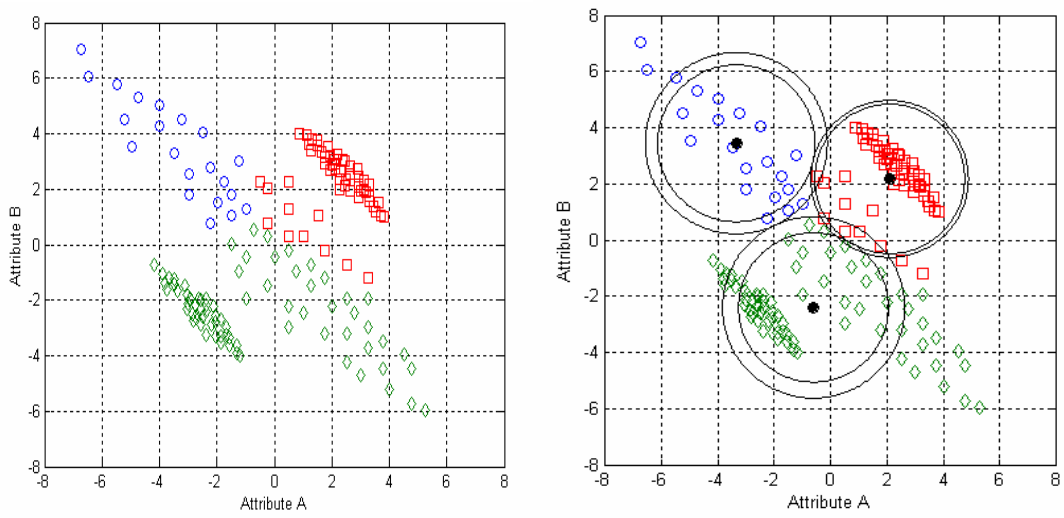


Figure 7.9: The application of the K-means clustering algorithm to the input dataset of Figure 7.8, showing (a) the three output clusters (blue circles, green diamonds, and red squares), and (b) the cluster centres (black circles) with circles indicating the mid-point distance between cluster centres.

7.4.3 Mahalanobis clustering

As discussed in the last section, *K*-means clustering is based on the Euclidean distance, which can be written

$$d_{ik}^2 = \left|\boldsymbol{x}_i - \boldsymbol{\mu}_k\right|^2 = (\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T(\boldsymbol{x}_i - \boldsymbol{\mu}_k),\ i = 1,\ldots,N;\ k = 1,\ldots,K, \quad (7.14)$$

where $\boldsymbol{x}_i^T = [x_{i1}, x_{i2}, \ldots, x_{iM}]$, *and* $\boldsymbol{\mu}_k^T = [\mu_{i1}, \mu_{i2}, \ldots, \mu_{iM}]$

Another type of distance is the statistical, or Mahalanobis, distance (Johnson and Wichern, 1998) from $\boldsymbol{x}$ to $\boldsymbol{\mu}$, which can be written

$$\Delta_{ik}^2 = (\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_k), \quad (7.15)$$

where $\boldsymbol{\Sigma}$ is the covariance matrix given by $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1M} \\ \vdots & \ddots & \vdots \\ \sigma_{M1} & \cdots & \sigma_{MM} \end{bmatrix}$, and the individual covariance values of $\boldsymbol{\Sigma}$ are computed from the outer product sum given by $\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^T$. Recall from Chapter 3 that the Mahalanobis distance could also be interpreted as the exponent in the multivariate Gaussian distribution, which is given by

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{M/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} exp\left[-\frac{\Delta^2}{2}\right]. \quad (7.16)$$

Also note that the covariance matrix with statistically independent variates and unit variances is equal to the identity matrix. That is, if

$$\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}, \quad (7.17)$$

then

$$\Delta_{ij}^2 = (\boldsymbol{x}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_j) = (\boldsymbol{x}_i - \boldsymbol{\mu}_j)^T(\boldsymbol{x}_i - \boldsymbol{\mu}_j), \quad (7.18)$$

which is the Euclidean distance. Thus, Mahalanobis distance can be seen as the generalization of Euclidean distance, and can be computed for each cluster if the covariances of the cluster are known.

After the application of the *K*-means clustering algorithm shown in Figure 7.9, we can then compute the means and covariances of each cluster, which gives us an initial estimate of these values. Then, by iterating through the same steps given at the start of the last section, but using Mahalanobis distance rather than Euclidean distance, we can improve the clustering. I refer to this as Mahalanobis clustering. (Although the author of this dissertation is not aware of any published theory on this method, it appears to be an intuitively obvious approach and has probably been implemented by others). The result of applying 20 iterations of Mahalanobis clustering is shown in Figure 7.10(a). Notice that the cluster values are now correctly assigned. Figure 7.10(b) shows the lines of equal bivariate gaussian amplitude, illustrating that the elliptical clusters have indeed been captured.

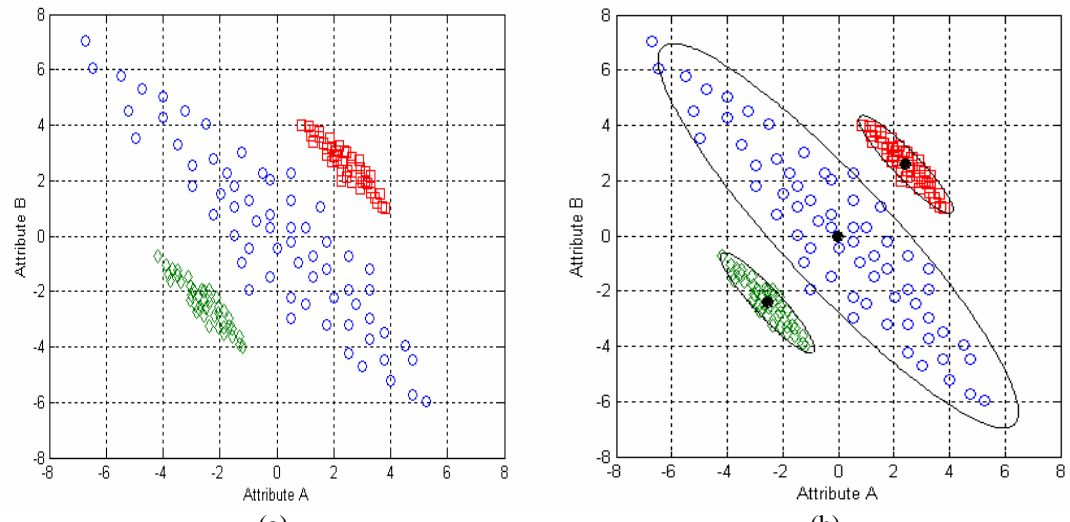


Figure 7.10: The application of the Mahalanobis clustering algorithm to the input dataset of Figure 7.8, where (a) shows the three output clusters (blue circles, green diamonds, and red squares), and (b) shows the cluster centres (black circles) with the ellipses showing lines of constant variance.

7.5 AVO Crossplot clustering

An obvious application of Mahalanobis clustering is to AVO crossplot analysis (Ross, 2000). To illustrate the method, I will use a data example from the Colony sand play in central Alberta (Russell and Lines, 2003). A seismic section across a known gas sand is shown in Figure 7.11. Although the gas sand is an obvious "bright spot", there are many false "bright spots" in the area due to thin limestone stringers. In Figure 7.11, the sonic log from the discovery well has been inserted at its correct location, after conversion from time to depth and the application of a check-shot correction.

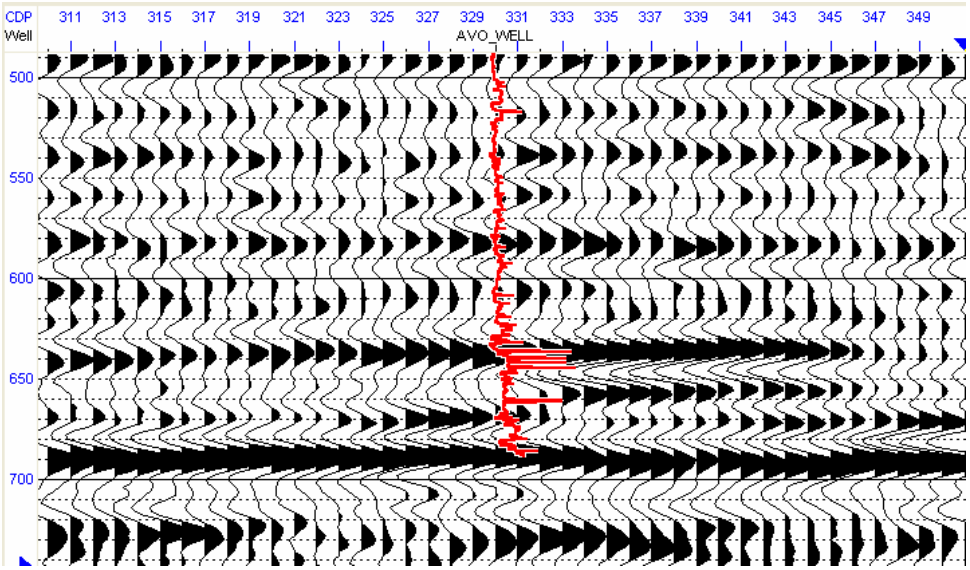
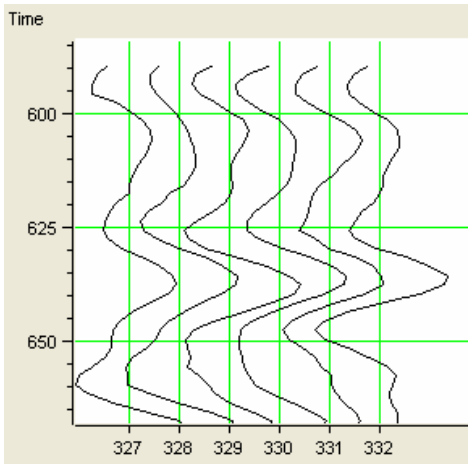


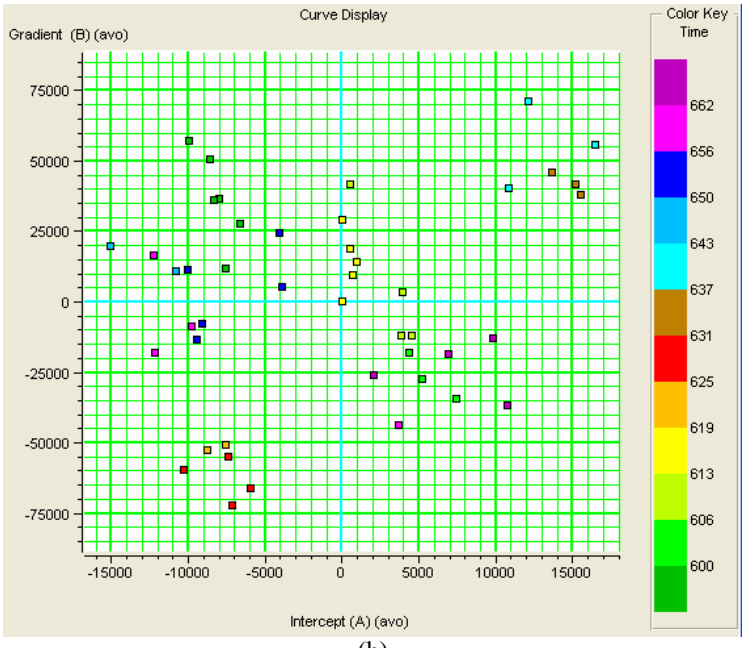
Figure 7.11: A seismic line over a know gas zone, with the sonic log from the discovery well overlain at CDP 330. The gas sand is indicated by the "bright spot" at a time of 630 ms.

A small window of the seismic section shown in Figure 7.11 is shown in Figure 7.12(a), with the intercept (A) versus gradient (B) crossplot from the peaks and troughs in this window shown in Figure 7.12(b). This window is between CDP 327 and 332 and encompasses a time window of 80 ms, around a time of 630 ms.

In the crossplot shown in Figure 7.12(b), the colour scale represents time. As discussed by Ross (2000) and Russell et al. (2002), the human interpreter would interpret this crossplot as a Class 3 AVO anomaly, with a "wet trend" visible as a cluster of points running along a -90 degree line, and two anomalous clusters in the first and third quadrants of the crossplot. These anomalies represent the top (third quadrant) and base (first quadrant) of the gas sand.



(a)



(b)

Figure 7.12: AVO intercept/gradient crossplot analysis over a window from the seismic section of Figure 7.17, where (a) is the seismic window, and (b) is the un-interpreted crossplot.

I will next apply the clustering algorithms described earlier to the crossplot shown in Figure 7.12(b). This results are shown in Figure 7.13, where Figure 7.13(a) shows the application of twenty iterations of the *K*-means algorithm, and Figure 7.13(b) shows the application of a further twenty iterations of the Mahalanobis algorithm to the output of the *K*-means algorithm.

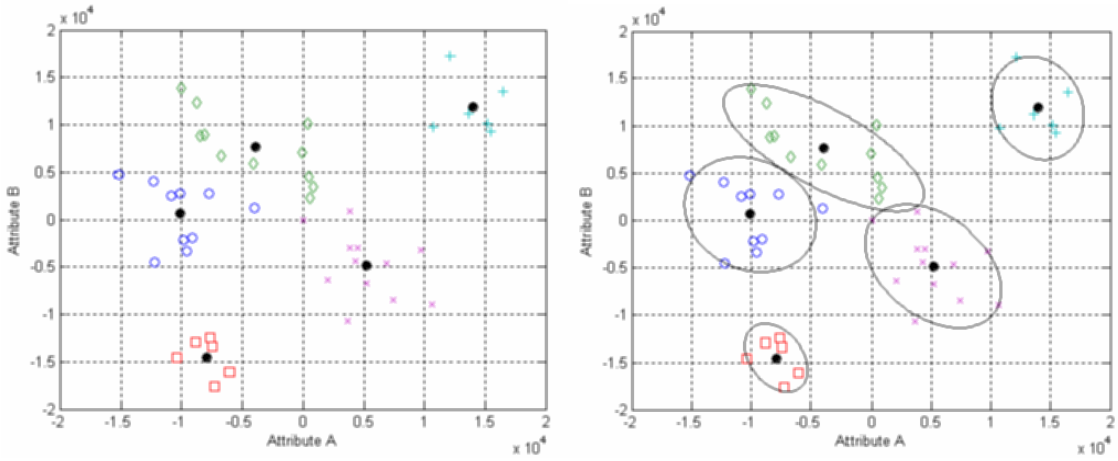
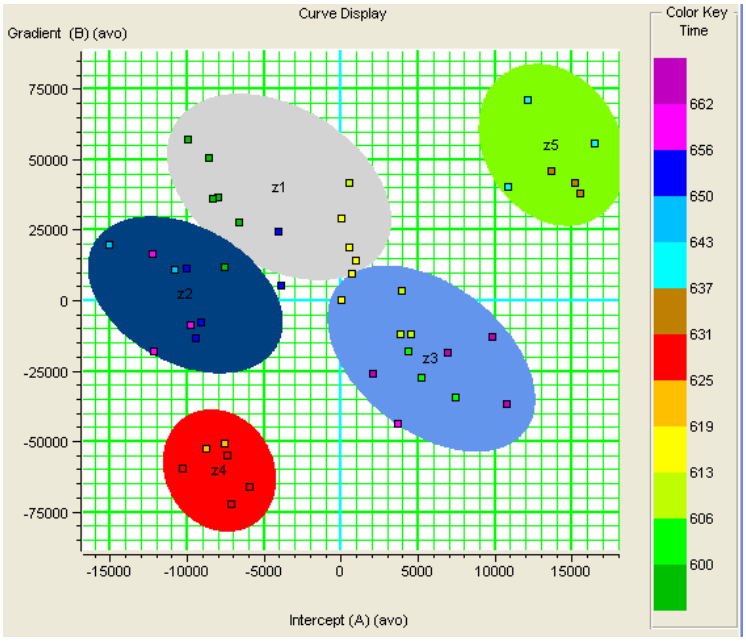


Figure 7.13: The application of (a) the *K*-means clustering algorithm, and (b) the Mahalanobis clustering algorithm to the crossplot of Figure 7.12(b), where the different shapes and colours indicate the five clusters, the black dots show the cluster centers, and the ellipses are equal variance lines enclosing the Mahalanobis clusters.

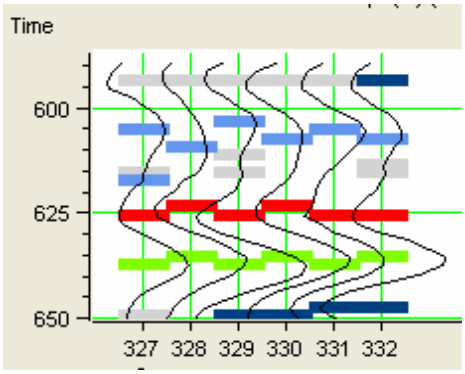
In Figure 7.13, notice that five clusters have been used in the training of the algorithm. This was due to the fact that there appeared to be more than three obvious clusters on the crossplot. (It should be pointed out that the number of clusters, or centers, to use is not a trivial issue, and has been discussed by many authors. For example, Chen et al. (1991) use an orthogonal least squares learning algorithm to determine the number of centers.)

The *K*-means clustering algorithm of Figure 7.13(a) has done an excellent job in finding five distinct clusters of points, and these are indicated by five different shapes and colours on the figure. When the output from the *K*-means algorithm was processed through twenty iterations of the Mahalanobis algorithm, as shown in Figure 7.13(b), the points in the clusters and the cluster centers do not change. In this case, the K-means

algorithm appears to have done the optimum job. However, the advantage of the Mahalanobis method in this case is that it defines the elliptical shapes (and their quantitative parameters) that can be used for the application of clustering back to the crossplot of Figure 7.12(b). The results are shown in Figure 7.14.



(a)



(b)

Figure 7.14: The application of the Mahalanobis clustering results shown in Figure 7.13(b) to the original crossplot of Figure 7.12(b), showing (a) the crossplot with the ellipse superimposed, and (b) application to the seismic traces of Figure 7.12(a).

In Figure 7.14(a) I have plotted the result of applying the elliptical shapes from the Mahalanobis clustering to the original crossplot of Figure 7.14(b), and in Figure

7.14(b) I have shown the equivalent peaks and troughs on the seismic window shown in Figure 7.12(a). Notice that the identified clusters appear to define coherent events in the seismic window. Although I have derived the crossplot clustering parameters over a very small window, we can apply the results of the clustering to the complete dataset. Figure 7.15 shows the application of the training to a portion of the complete line shown in Figure 7.11.

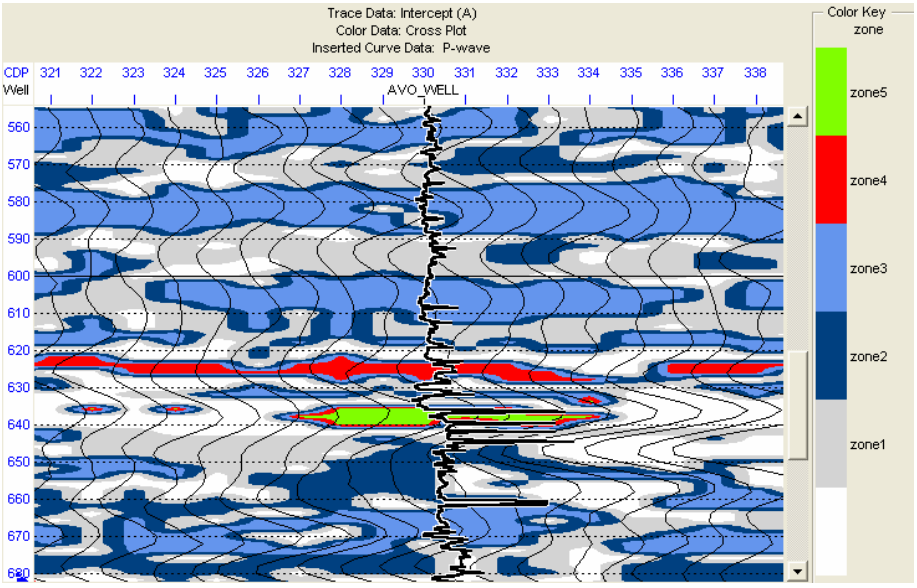


Figure 7.15: The application of the Mahalanobis clustering results shown to the complete seismic line of Figure 7.11.

In Figure 7.15, we clearly see the top of the gas sand (in red, called zone 4 on the colour bar) and the base of the gas sand (in green, called zone 3 on the colour bar). Also, the other three clusters, which are clearly on the wet trend in Figure 7.14(a), all plot on parts of the seismic line that are non-anomalous, as expected. Thus, the *K*-means/Mahalanobis clustering method has produced an excellent result in this case.

It is important to note that a human interpreter could do at least as good a job in identifying these anomalies on the crossplot. Also, the method depends on having a certain amount of separation between the anomalies and the wet trend. The real power of this method is when we apply to higher dimensional spaces, especially those having more than three dimensions. This extension of the *K*-means clustering algorithm will be

discussed in the next section, in which the method is applied to the radial basis function neural network used for predicting reservoir parameters.

7.6 K-means clustering with the radial basis function neural network

I will now compare the strict interpolation RBFN method with the RBFN method with centres to the channel sand case study which was discussed in sections 5.5, 6.4.2, and 6.8. This study was published by Russell and Lines (2003). Recall that this study involved the prediction of P-wave velocity in the Glauconitic reservoir of the Blackfoot field of central Alberta. The reservoir occurs at a depth of around 1550 m, where Glauconitic sand and shale fill valleys incised into the regional Mannville stratigraphy. The well log input consists of twelve wells, each with sonic and density logs. The base map showing the twelve wells, the inverted seismic data for line 95, and the impedance dataslice were shown in Figures 5.23 through 5.25. The multi-linear attribute training result was shown in Table 6.1. In this analysis, I used all twelve wells in the training, an operator length of seven points, and seven attributes. The training results of the strict interpolation RBFN algorithm is shown in Figure 7.16, where only the first three wells are displayed. The correlation coefficient is 0.7994 and the observed fit is very good. In the training of the full RBFN algorithm, I used a total training dataset of approximately 1000 points.

Next, I will apply the RBFN method with centers to the problem I have just considered. I reduce the number of points to 25 centres and see how well the results compare to the full RBFN. The same attributes will be used as shown in Table 6.1 and used in the full interpolation method. The centres are found using the *K*-means clustering algorithm. Since the vectors are 6-dimensional, it is impossible to actually visualize the clusters. The result of training with 25 centers is shown in Figure 7.17, where only the first three wells are displayed. Notice that the correlation coefficient is 0.634, less than the full RBFN value of 0.7994, but still a very good fit.

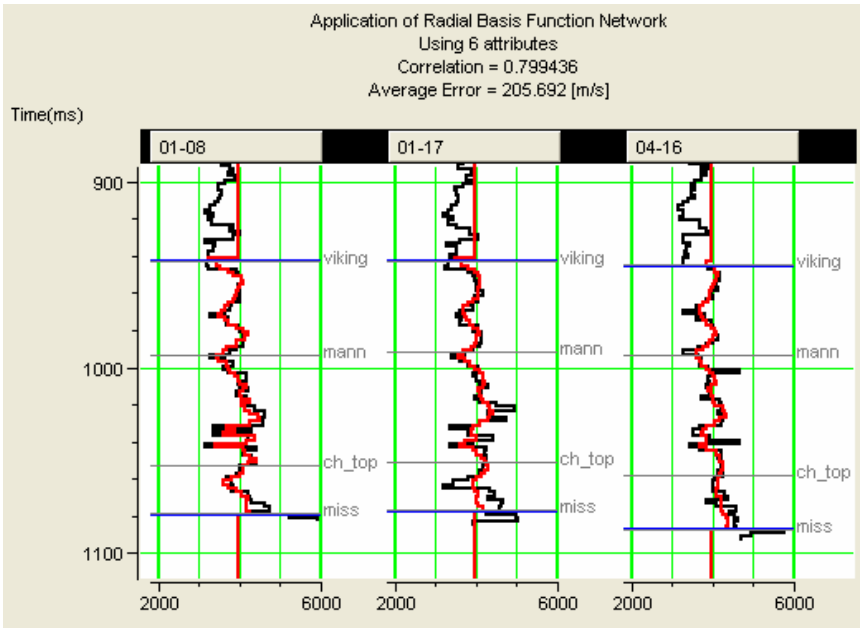


Figure 7.16: Training the RBFN algorithm, where all the training samples are used in the prediction. The black lines show the original curves and the red lines show the predicted curves.

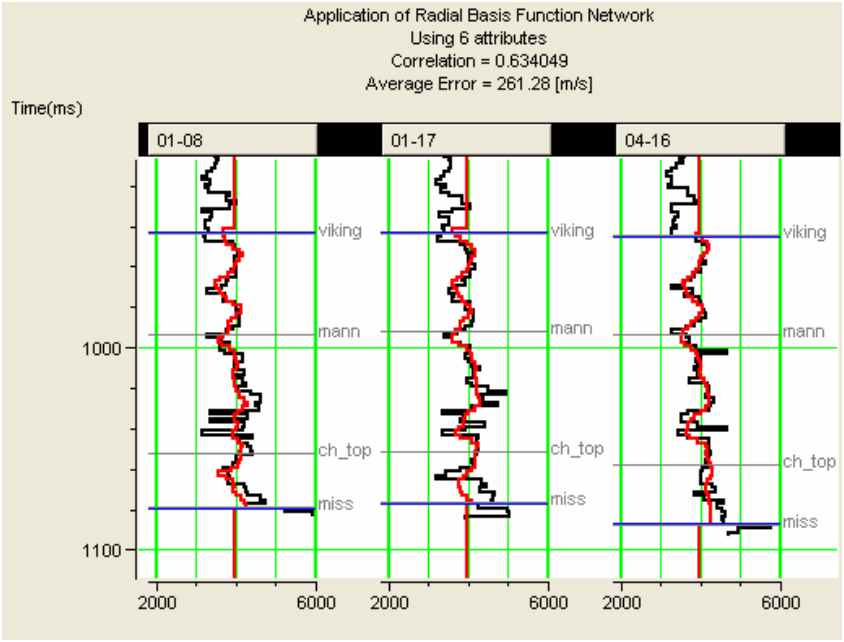


Figure 7.17: Training the RBFN algorithm using 25 centers. The black lines show the original curves and the red lines show the predicted curves.

The cross-validation of the full interpolation RBFN result is shown in Figure 7.18, where each log has been left out of the training and then "blindly" predicted. The black lines show the original curves and the red lines show the predicted curves. Only the first three P-wave sonic logs have been shown, although all twelve were used in the training. Notice that the cross-validation error is 0.6577, which is lower that the training error without cross-validation, but still quite good.

The cross-validation of the RBFN result with 25 centers is shown in Figure 7.19.. The black lines show the original curves and the red lines show the predicted curves. Only the first three P-wave sonic logs have been shown, although all twelve were used in the training. Notice that the cross-validation error is 0.602, which is less than the full RBFN result of 0.6558, but actually much closer than the full training result. That is, the difference between the training result with all wells and the cross-validation result is much smaller for RBFN with 25 centers than for full RBFN.

The results of the training from the full interpolation RBFN method is then applied to the seismic line shown in Figure 5.24, and is shown in Figure 7.20. Notice the excellent fit of the predicted P-wave sonic log at the well tie, and also the high frequency detail of the results as we move away from the well. There is also good lateral continuity of the predicted events.

Finally, the results of the training using the RBFN method with 25 centres is then applied to the seismic line shown in Figure 5.24, and is shown in Figure 7.21. Notice the excellent fit of the predicted P-wave sonic log at the well tie, and also the high frequency detail of the results as we move away from the well. There is also good lateral continuity of the predicted events. This detail should be compared with the results shown in Figure 7.19. Although the full result of Figure 7.19 is better, we have achieved almost as good a result using roughly 2.5 % of the original number of data values.

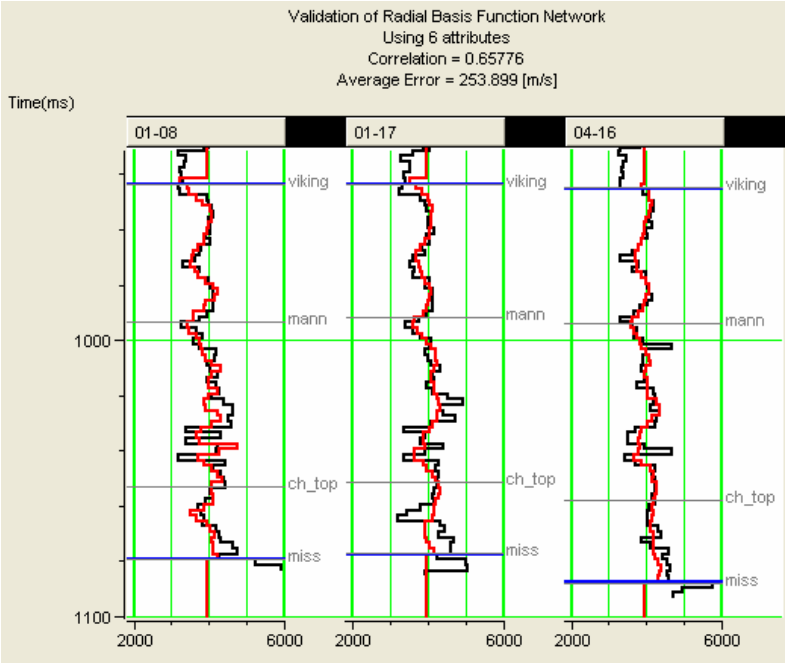


Figure 7.18: Validation of the RBFN, where the each log has been left out, in turn, of the training. The black lines show the original curves and the red lines show the predicted curves.

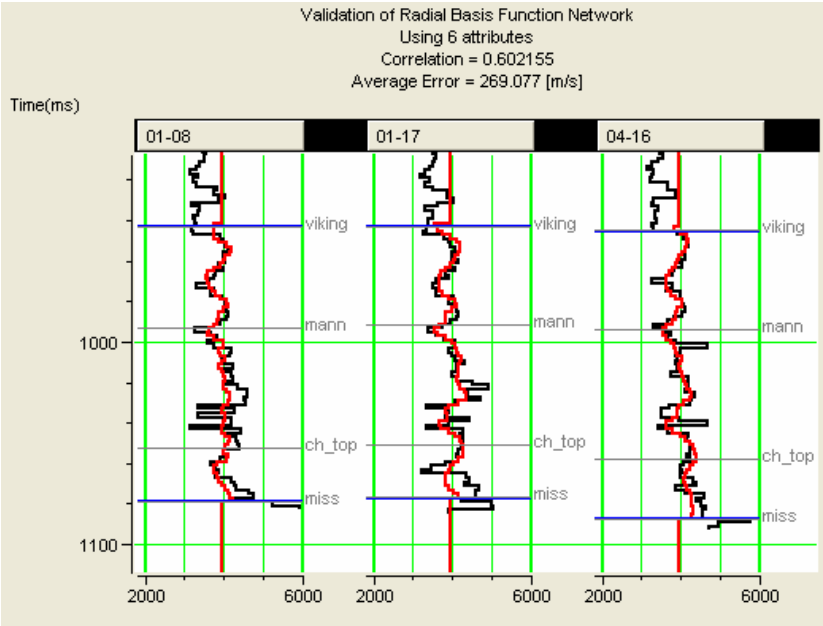


Figure 7.19: Validation of the RBFN using 25 centres, where the each log has been left out, in turn, of the training. The black lines show the original curves and the red lines show the predicted curves.

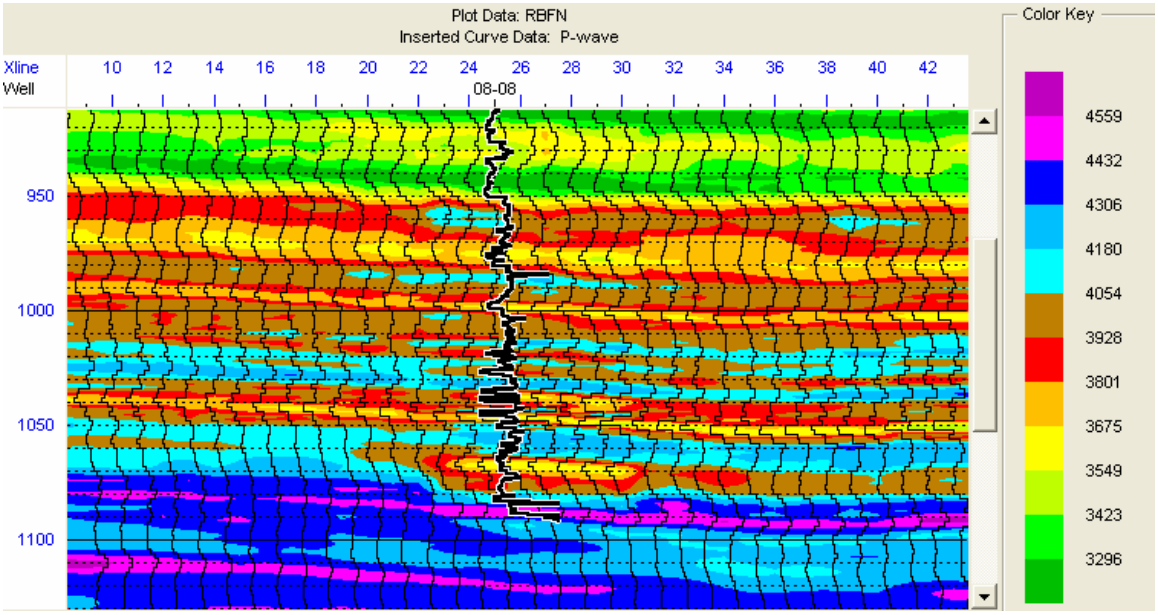


Figure 7.20: Application of the full RBFN algorithm to line 95 of the 3D volume.

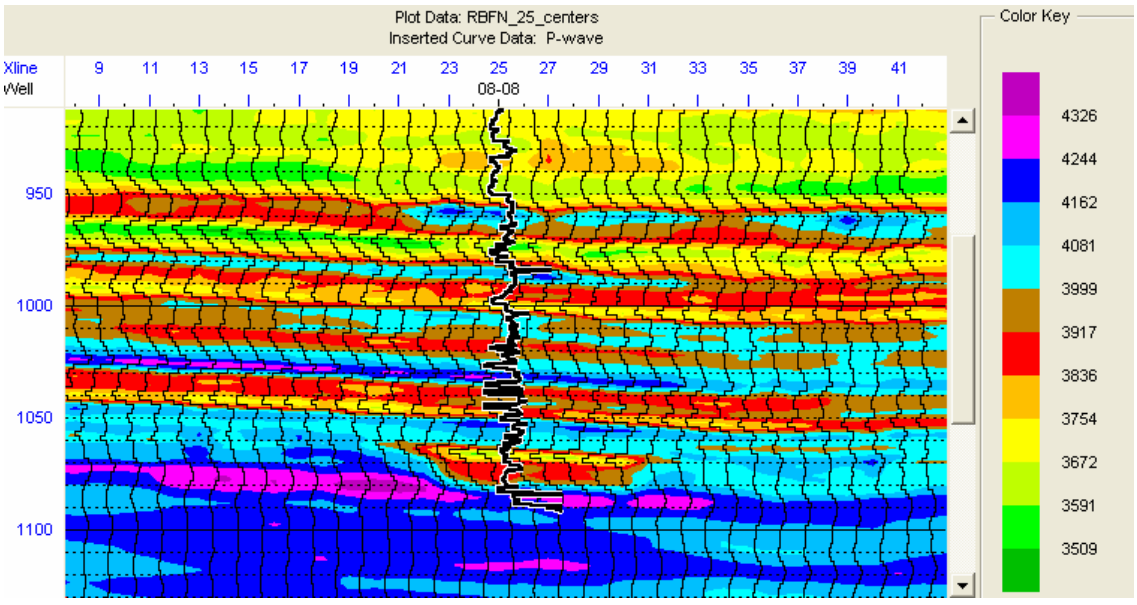


Figure 7.21: Application of the RBFN with 25 centers to line 95 of the 3D volume.

7.7 Mahalanobis clustering applied to the RBF network

In the previous section, I showed how *K*-means clustering could be used to improve the run time and stability of the RBFN algorithm by reducing the full interpolation algorithm to an algorithm that uses a limited number of centers (or mean vectors) in the design of the weight values. You will also recall that the scaling functions σ given in equation (7.5) were constant and were optimized by using a range of σ values and choosing the one that gives the lowest cross-validation error. If we return to the concept of statistical distance, notice that the full basis function equation can actually be written as follows (Bishop, 1995), using statistical distance rather than Euclidean distance:

$$\phi_{ik} = \exp\left[-\frac{1}{2}(\boldsymbol{s_i} - \boldsymbol{\mu_k})^T \boldsymbol{\Sigma_k^{-1}} (\boldsymbol{s_i} - \boldsymbol{\mu_k})\right], \tag{7.19}$$

where $\boldsymbol{\Sigma_k} = \begin{bmatrix} \sigma_{11}^{(k)} & \cdots & \sigma_{1M}^{(k)} \\ \vdots & \ddots & \vdots \\ \sigma_{M1}^{(k)} & \cdots & \sigma_{MM}^{(k)} \end{bmatrix}$ and M is the total number of attributes.

Equation (7.19) can be thought of as analogous to the multivariate Gaussian distribution (Johnson and Wichern, 1998). The Σ_k matrix represents the covariance matrix for the k^{th} cluster. The basis function in equation (7.19) can thus be seen to be a simplification of equation (7.4), in which the covariance matrix can be written in the simplified form

$$\boldsymbol{\Sigma_k} = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma^2 \end{bmatrix} = \sigma^2 I \ . \tag{7.20}$$

Using the Mahalanobis clustering method proposed in the previous section, we can obtain an estimate of each of these covariance matrices. However, this method has proved to be fairly unstable, and I decided to stay with the standard approach.

7.8 Parameter optimization in the RBFN with centres method

In the RBFN with centres method, there are three parameters that must be optimized: the σ term in the basis functions, the regularization parameter λ, and the number of centres K. Optimization of the σ term is done in the same way as discussed in section 6.5.2. In this section I will therefore discuss the optimization of the latter two parameters, λ and K, using the methods discussed by Orr (1996).

Recall that our objective is to find the optimal set of weights that will predict the training samples from the input attributes, where we have K weights and N inputs. The forward equation for this relationship was given in equation (6.6) and the inverse equation in equation (6.7). To check the error in the weights, we can apply them to the training values and compute an estimated output $\boldsymbol{y}$. By combining this operation with equation (6.7), we get

$$\boldsymbol{y} = \Phi \boldsymbol{w} = \Phi(A^{-1}\Phi^T \boldsymbol{t}), \tag{7.21}$$

where $\boldsymbol{y} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix}$, $\boldsymbol{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$, $\Phi = \begin{bmatrix} \phi_{11} & \cdots & \phi_{1K} \\ \vdots & \ddots & \vdots \\ \phi_{N1} & \cdots & \phi_{NK} \end{bmatrix}$, $A = \left[\Phi^T\Phi + \lambda I_K\right]$, I_K is the K dimensional identity matrix, and $\phi_{ij} = \exp\left[-\frac{\left|\boldsymbol{s}_i - \boldsymbol{\mu}_j\right|^2}{\sigma^2}\right]$. Using equation (7.21) the error between the observed and predicted values can be written

$$\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{y} = \boldsymbol{\Phi w} = (I_K - \boldsymbol{\Phi} A^{-1}\boldsymbol{\Phi}^T)\boldsymbol{t} = P\boldsymbol{t}, \tag{7.22}$$

where P is an N x N dimensional matrix called the projection matrix. The sum of squared error S is thus given by

$$S = \boldsymbol{e}^T\boldsymbol{e} = \boldsymbol{t}^T P^2 \boldsymbol{t} \ . \tag{7.23}$$

We can make use of the projection matrix and the sum of squared error to optimize λ. Orr (1996) shows that the optimum value of λ can be estimated iteratively using the formula

$$\hat{\lambda} = \frac{\boldsymbol{t}^{\boldsymbol{T}} P^2 \boldsymbol{t}\, \mathrm{tr}(A^{-1} - \hat{\lambda} A^{-2})}{\boldsymbol{w}^{\boldsymbol{T}} A^{-1} \boldsymbol{w}\, \mathrm{tr}(P)}, \tag{7.24}$$

where tr(·) indicates the trace of the matrix. To apply this procedure, we make an initial guess of $\hat{\lambda}$ and calculate the right hand side of equation (7.24). This gives us a new estimate of $\hat{\lambda}$ and we then iterate until convergence.

The problem of estimating the optimum number of basis functions is a more difficult task. The optimum number of centres is somewhere between *1* and *N*, the total number of training points. Orr (1996) shows that the optimum number of centres can be found by computing a large number of centres and then growing the network one centre at a time, each time computing the error difference. This error difference can be computed incrementally using the formula

$$S_k - S_{k+1} = \frac{\boldsymbol{t}^T P_k \boldsymbol{\phi}_J}{\boldsymbol{\phi}_J^T P_k \boldsymbol{\phi}_J}, \tag{7.25}$$

where the $\boldsymbol{\phi}_J$ functions are the columns of the design matrix, or $\Phi = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_K]$. The minimum error will indicate how many basis functions to use. In the approach taken in this study, equation (7.25) was found to be too costly in terms of computer time, and I therefore determined the number of basis functions by trial and error.

7.10 Conclusions

In this chapter, I discussed the RBFN method using basis centres. To compute the basis centres, I used the K-means clustering approach and presented a new approach to clustering, which I call Mahalanobis clustering. This method is an extension of *K*-means clustering in which we apply a second iteration which uses statistical, or Mahalanobis, distance to perform the clustering.

After a review of the *K*-means method, in which I illustrated the approach using a simple two-dimensional problem, I then presented a model dataset for which the *K*-means method did not converge to a correct answer. This model dataset consisted of elongated elliptical clusters. I then described the Mahalanobis clustering method and showed that this method converged to a correct answer for our model dataset.

I then applied the *K*-means and Mahalanobis clustering methods to a two-dimensional AVO crossplot example. Although both methods converged to the same answer, I showed that the Mahalanobis method was able to give us the correct elliptical shapes for interpreting the AVO crossplot. This was confirmed by the interpretation of the crossplot, in which the top and base of a gas zone were identified.

I then used *K*-means clustering to find the clusters to be used in the radial basis function neural network (RBFN) method with centers, and found that we could achieve a result that was very close to the full RBFN method using clusters representing only 3% of the original number of points. Although the *K*-means method performed well, it gave us no way of finding and refining the values for the covariance matrices of each cluster, which would allow us to optimize the scaling parameters in the weight determination. I thus proposed a method which uses Mahalanobis clustering to perform this task. Finally, I discussed ways of optimizing both the regularization parameter λ and the number of clusters, *K*.

CHAPTER 8 : GEOSTATISTICS AND MULTIATTRIBUTE TRANSFORMS

8.1 Introduction

The classical problem in seismic exploration and production is how to integrate seismic data, which are spatially closely sampled but of relatively low temporal resolution, with well log data, which are of high temporal resolution but are poorly sampled spatially. This can be solved using geostatistical methods such as cokriging and kriging with external drift (Doyen, 1988). In this approach, the well-log data are considered to be the primary dataset, and the seismic data provides a background trend. The advantage of this method is that the primary variable is honoured exactly at the well tie. The disadvantage is that this perfect tie often implies a less-than-perfect physical model. For example, if cokriging is applied to tying seismic structure to picked well depths, the implied velocities often show "bulls-eyes" around the well intersections.

We can use multilinear regression and neural networks to predict well log properties from seismic attributes, where the wells are considered to be the training points, and the method being used "learns" the relationship between the attributes and the well values. This relationship is then applied to the seismic volume to create a reservoir parameter volume. The advantage of this approach is a good overall fit to the parameter of interest. The disadvantage is that the fit at the well ties locations is not exact.

In this chapter, I combine the methods of geostatistics and multiattribute prediction and show a geological application of this approach. Our example will involve a map-based approach where the attributes are derived from an interval over the zone of interest of the channel sand. Part of this chapter has been published in the Journal of

Petroleum Geology (Russell et al., 2002c). Similar work, using multi-component data, has been published by Todorov et al. (1997, 1998) and Todorov (2000).

8.2 Channel Sand Case Study

Recall that our case study involves the prediction of porosity in the Blackfoot field of central Alberta. The reservoir occurs at a depth of around 1550 m, where Glauconitic sand and shale fill valleys incised into the regional Mannville stratigraphy. The objectives of the survey were to delineate the channel and distinguish between sand-fill and shale-fill. The well log input consisted of twelve wells, each with sonic, density, and calculated porosity logs. The top and base of the sand zone for each of the wells were picked, and the porosity was averaged between the top and base as input to the mapping procedure. Figure 8.1 shows the distribution of wells throughout the 3D survey area, as well as cross-line 18 from the P-wave seismic dataset.

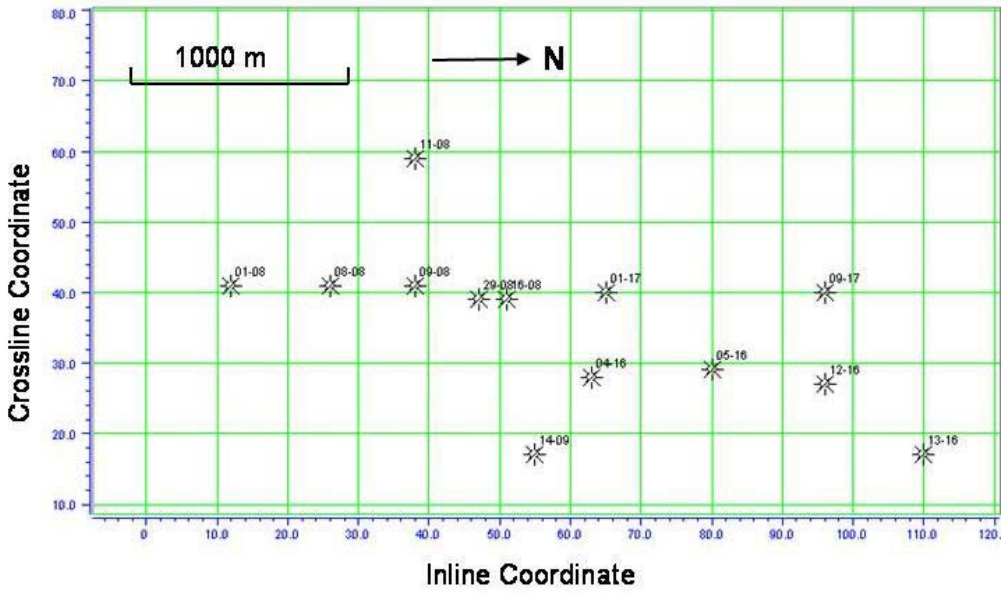


Figure 8.1: The distribution of wells within the 3D seismic survey area, Blackfoot, Alberta. The annotation shows inline and crossline numbers.

The seismic input consists of two 3D volumes. The first is the stacked P-wave seismic dataset from the survey, and the second is the acoustic impedance inversion of

the initial seismic volume. Figure 8.2 shows crossline 18 from the seismic survey, where (a) shows the input seismic line, and (b) shows the inverted line. Notice that this line ties well 14-09, a dry hole, and 13-16, a producer.

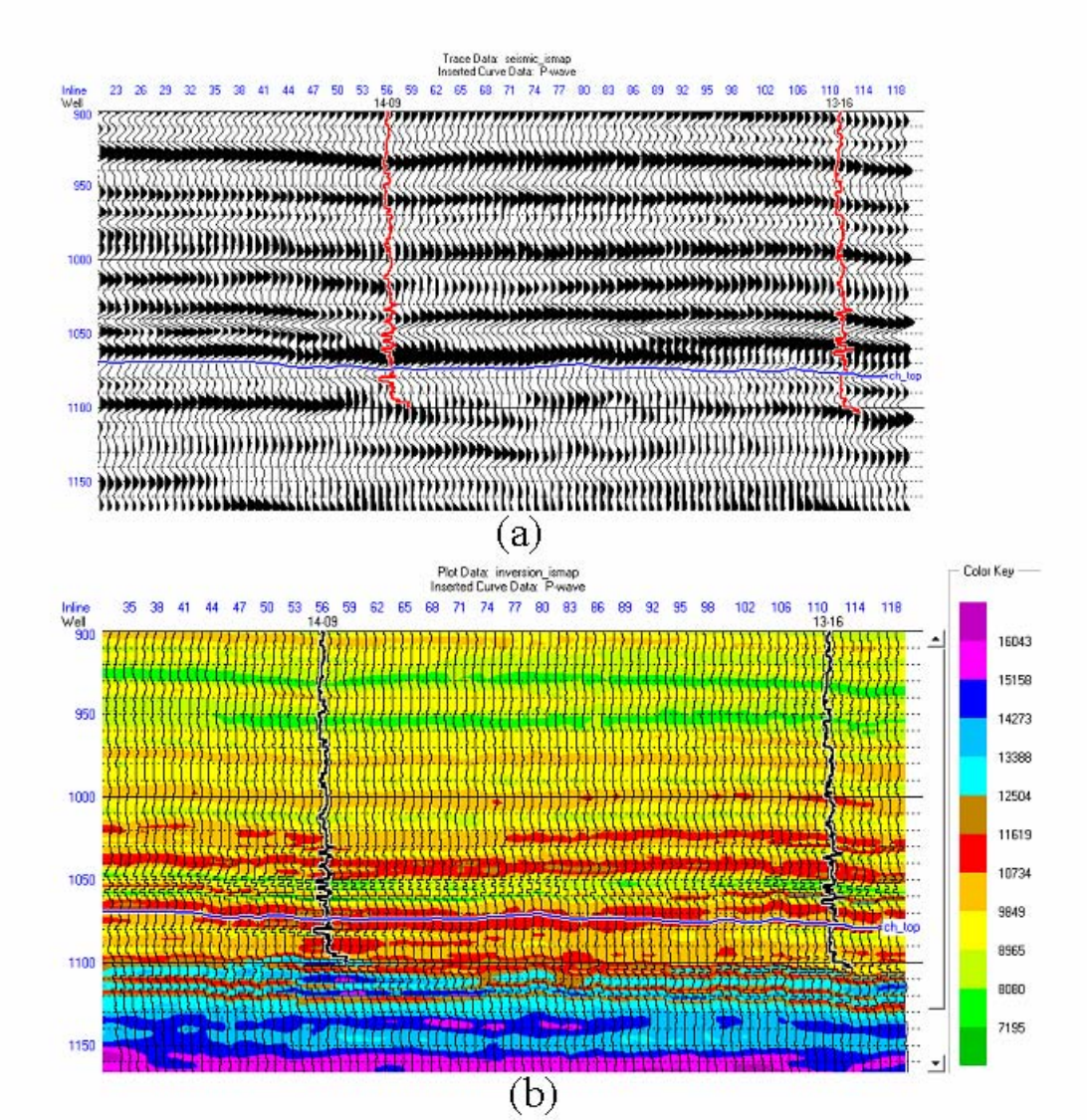


Figure 8.2. Cross-line 18 from the 3D seismic survey, where (a) shows the final CDP stack, and (b) shows the impedance inversion.

Another key to this method is the extraction of a stable seismic wavelet to be used in the inversion process. Figure 8.3 shows the sonic and porosity logs for well 14-09, along with the seismic picks and log tops, and a portion of the seismic data at the tie point. The figure also shows the synthetic tie at the well intersection.

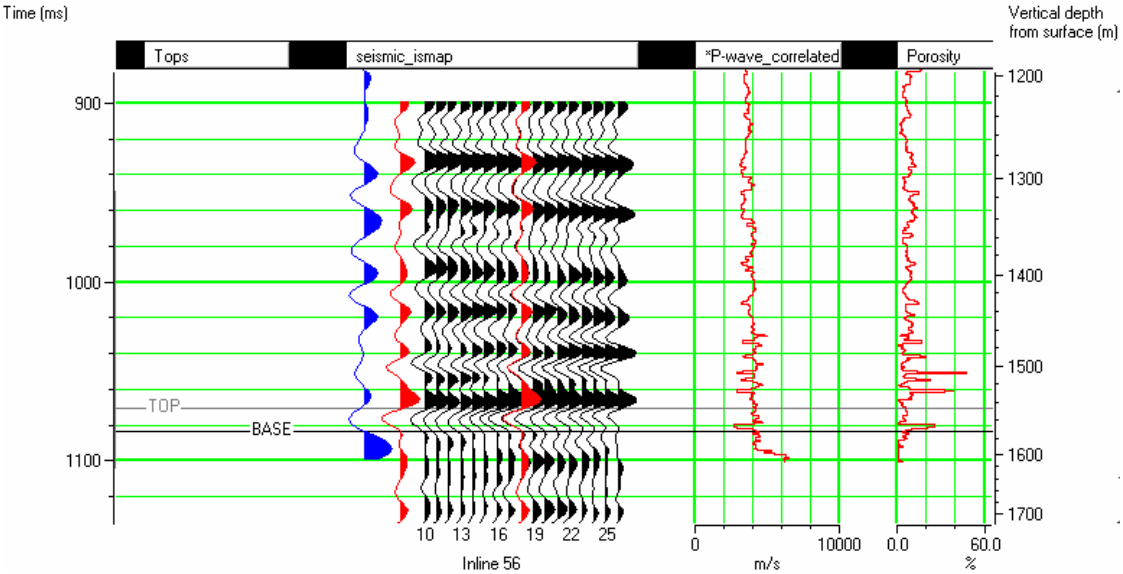


Figure 8.3: Sonic and porosity logs from well 14-09, together with synthetic tie, seismic picks, and tops.

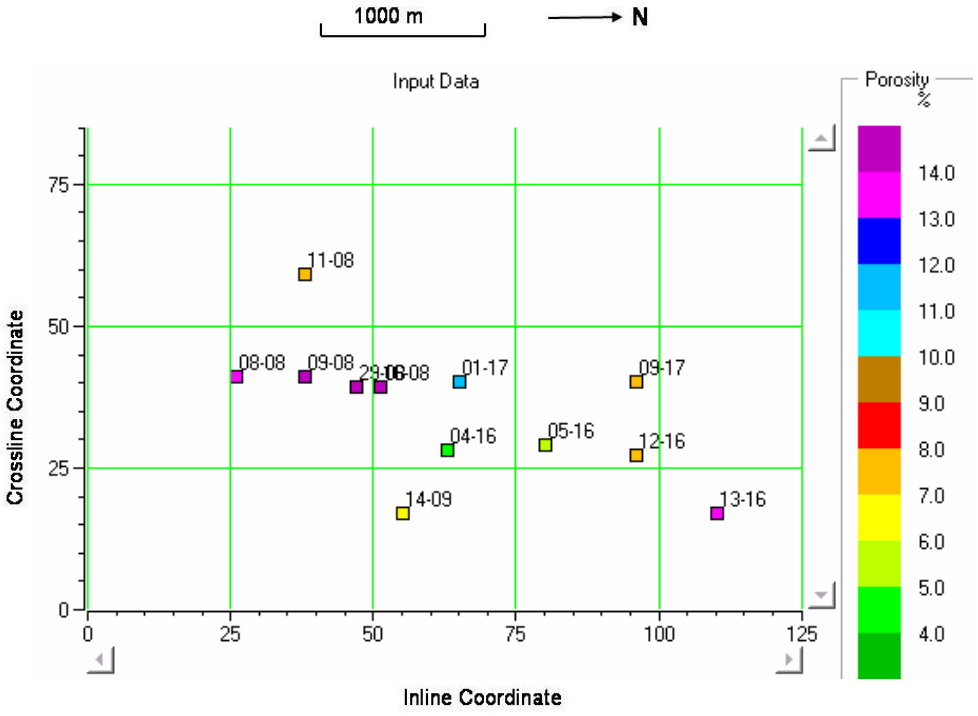


Figure 8.4: The porosity at each well location, averaged between the top and base of the channel.

The porosity value at each well was then averaged between the top and base of sand. The resulting values have been plotted and colour-coded on the map shown in Figure 8.4. Notice the high porosity values in the center left of the map.

Next, a dataslice was created through the inverted impedance volume to use as the secondary dataset. To create this slice, the channel top was picked (this pick is shown on a piece of the seismic data in Figure 8.3) and then an arithmetic average from a 10 ms window below the channel top was used to produce the slice. This resulting map slice is shown in Figure 8.5. Notice the low impedance values on the middle left side of this map, indicating the possible channel.

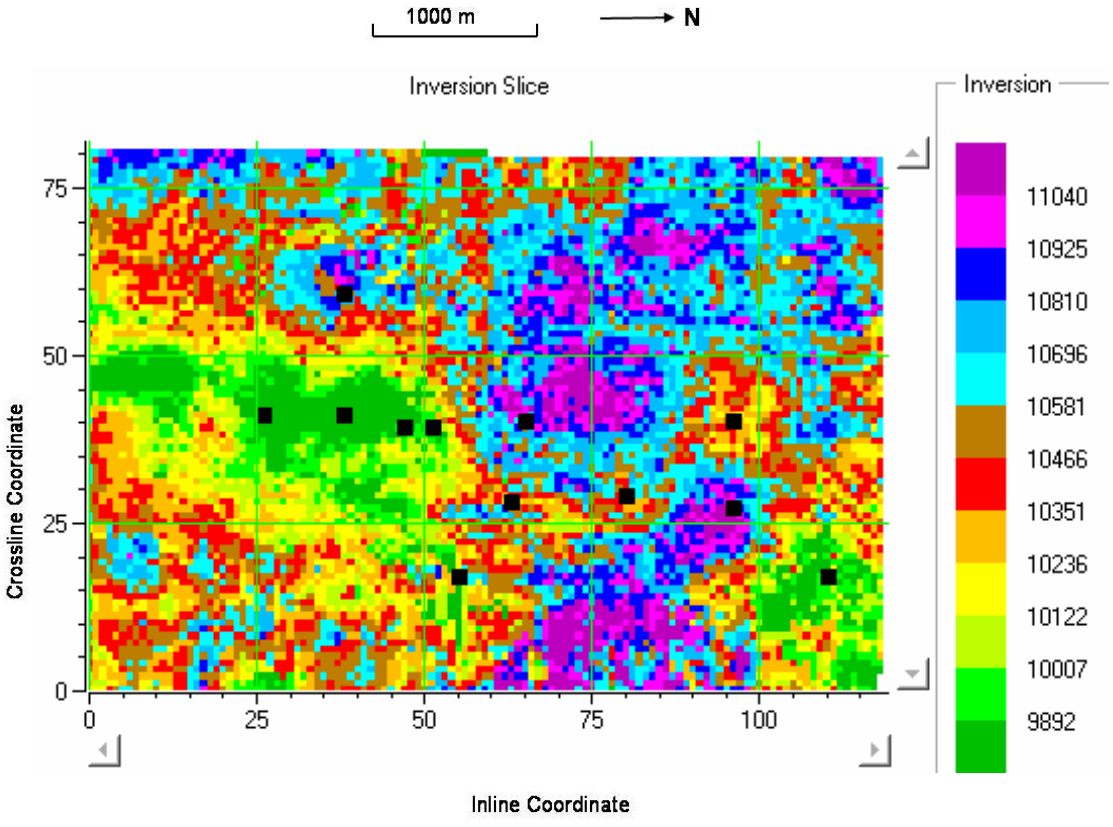


Figure 8.5: Map of average acoustic impedance over a 10 ms window below the picked channel top.

At the well locations, the average impedance values were then extracted from the map and crossplotted against the well porosity values. This crossplot is shown in Figure 8.6. Note that the correlation coefficient from the crossplot is equal to -0.65. The negative correlation was due to the fact that porosity varies as the inverse of impedance.

The regression fit from this crossplot was then applied to the impedance slice to convert it to pseudo-porosity for display purposes. The resulting porosity slice is shown in Figure 8.7. On this map, it is obvious that the well values do not tie the seismic values at the well locations, since the colour is indicative of porosity both at the wells and on the seismically derived map.

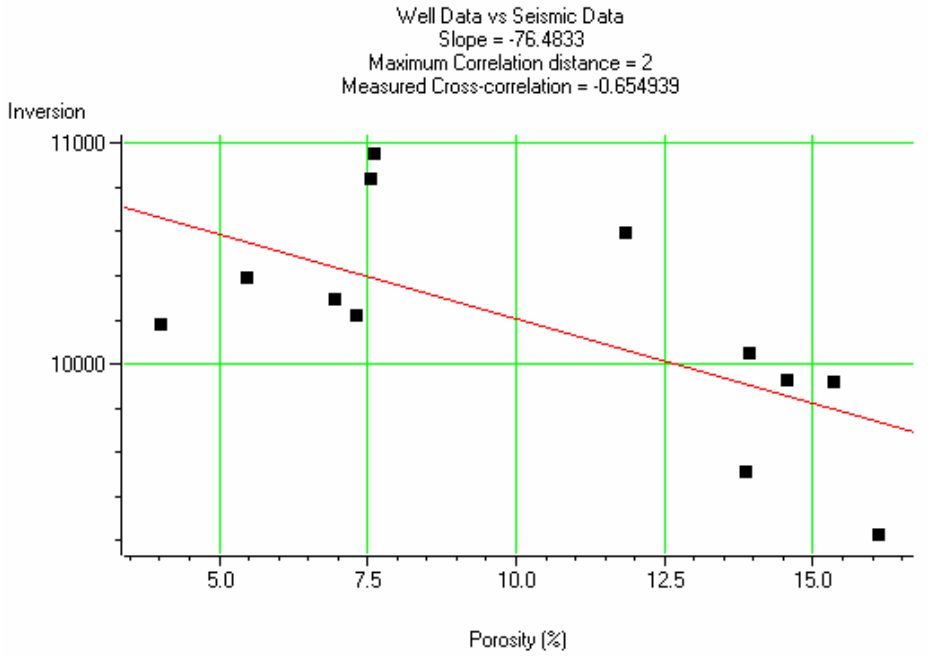


Figure 8.6: Plot of average well porosity against average impedance for all the wells in the survey area.

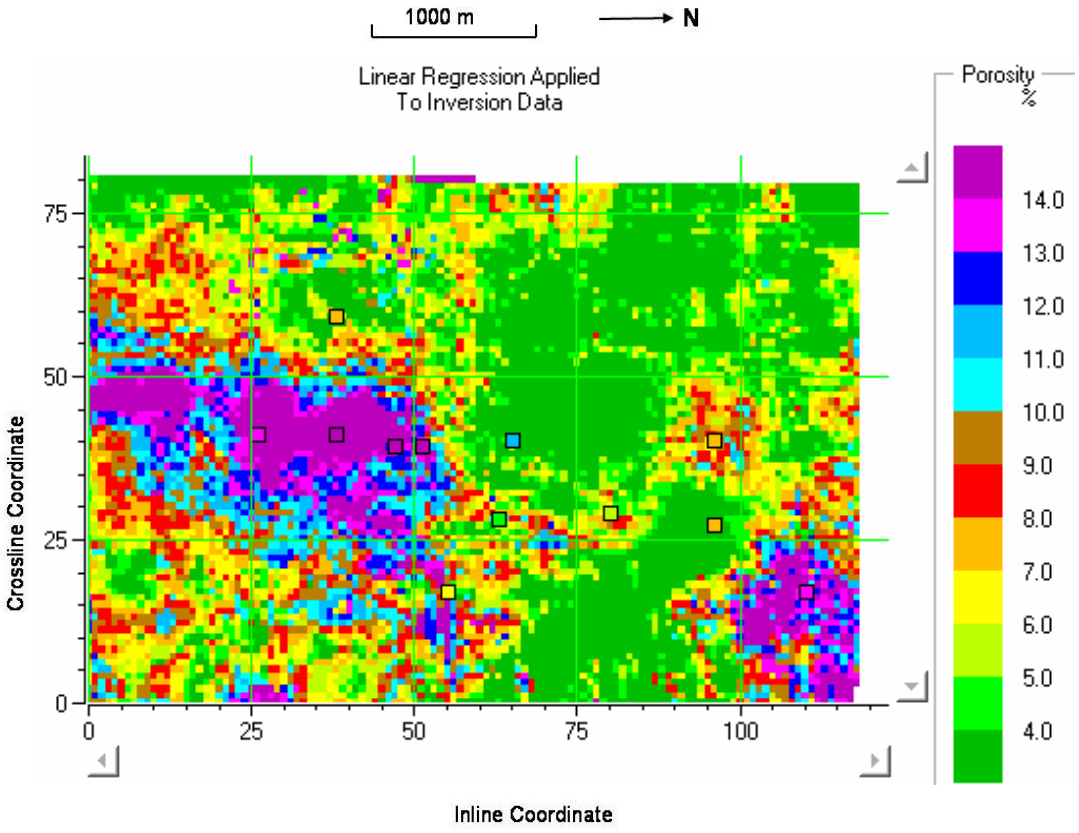


Figure 8.7: Map of porosity variations in the survey area. This was derived by the application of the regression fit from Figure 8.5 to the impedance slice of Figure 8.4.

8.3 Map-based geostatistics

Map-based geostatistics involves the integration of two related datasets. The primary dataset is generally a set of sampled values from well logs distributed throughout the map area. These values represent some reservoir parameter of interest such as porosity or depth. The secondary dataset is derived from a separate set of measurements, generally seismic data, which is related in some way to the primary dataset. For example, seismic amplitude or inverted seismic amplitude would be expected to correlate with porosity, whereas seismic structure time should correlate with measured well depth. To test the amount of correlation between the two datasets, a crossplot can be made between them (Fig. 8.6) and both the regression line and the correlation coefficient can be computed.

The next step is to derive variograms from the well data and seismic data alone, and also from the well-to-seismic comparison. The variogram (or more precisely, the semi-variogram) measures the spatial variability of the particular value being measured. That is, it tells us how this parameter changes as a function of distance. If there is very little spatial variability, the variogram will be close to zero, and if there is a lot of variability, the variogram will be large. In practice, the variogram is small for small distances close to zero, climbs rapidly, and flattens off to a uniform value. For the porosity case being considered in this section, we can write the variogram mathematically as

$$\gamma(h) = \frac{1}{2N} \sum_{h_{ij}=h} (\phi_i - \phi_j)^2 \,, \tag{8.1}$$

where h represents the offset between two porosity values and we collect all N pairs that are within an offset bin centered at h. The variogram is related to the covariance of the data by the following formula

$$C(h) = \gamma(\infty) - \gamma(h) = C(0) - \gamma(h) \,, \tag{8.2}$$

where $C(h)$ is the covariance at offset h and $C(0)$ is the zero offset covariance, which is identical to the variogram at an infinite distance.

Figure 8.8 shows two variograms, where (a) shows the well-to-well variogram and (b) shows the seismic-to-seismic variogram. In Figure 8.8, the black squares represent the computed variogram values from equation 8.1 and the red curve shows a mathematical fit to these points. This fit was done using the spherical function, written

$$\gamma(h) = \begin{cases} \gamma_0 + s\left[1.5\left(\frac{h}{a}\right) - 0.5\left(\frac{h}{a}\right)^3\right], h \le a \\ \gamma_0 + s, h > a \end{cases} , \tag{8.3}$$

where s is the sill, or the value at which the variogram flattens off; γ_0 is the nugget, or the starting value of the variogram; and a is the range, or the distance at which the variogram flattens off.

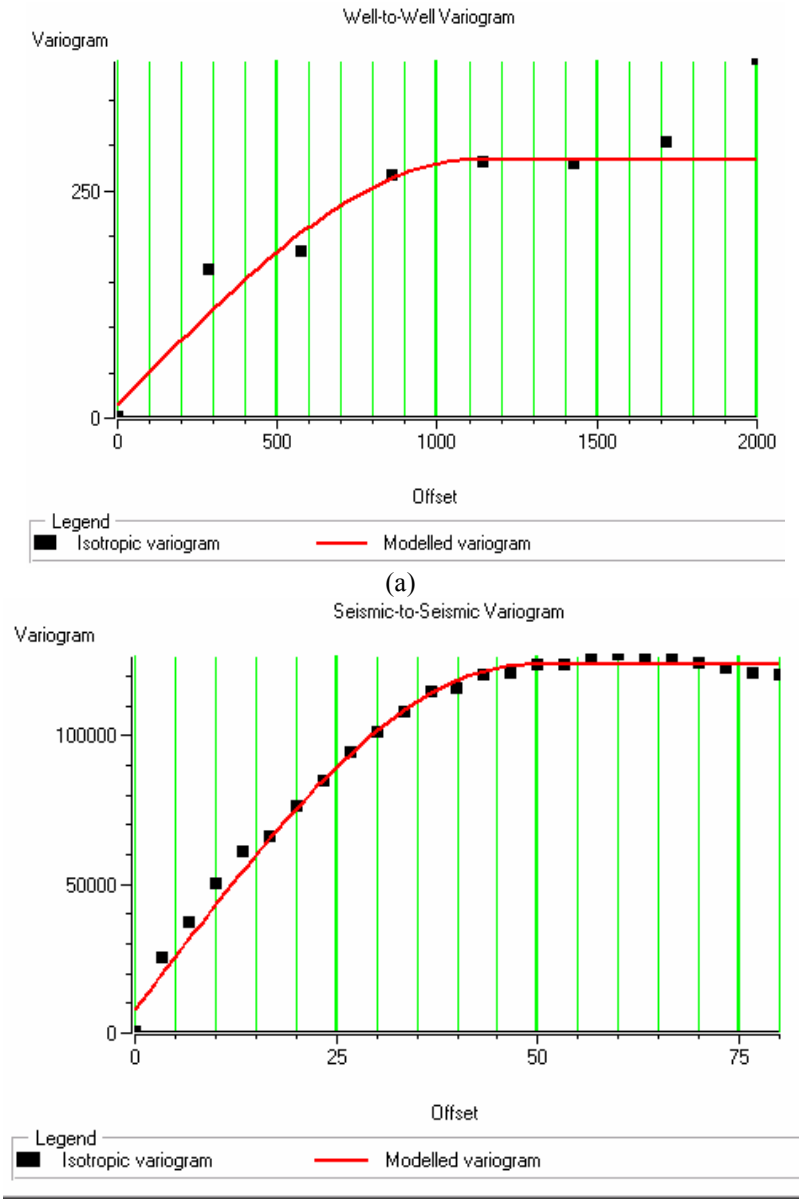


Figure 8.8: The spatial variogram of (a) the well values and (b) the seismic values. The variogram is the sum of the squared differences between all the pairs of points within a given offset value.

To understand how the variogram is used to create geostatistical maps, let us first consider the problem of creating a linear, unbiased estimate of an unknown map value given N input map values. The process involves finding the best N weights that will recreate the unknown sample, or

$$\phi(x_0, y_0) = w_1\phi(x_1, y_1) + w_2\phi(x_2, y_2) + \ldots + w_N\phi(x_N, y_N) = \sum_{i=1}^{N} w_i\phi_i \,, \tag{8.4}$$

where $\phi(x_0, y_0) = \phi_0$ is the unknown sample and ϕ_1 through ϕ_N are the known samples. As shown by Isaaks and Srivastava (1989), the weights can be estimated by using random variable theory and making the following two assumptions

(1) The mean between the estimated and true value is equal to zero.

(2) The error variance is minimized.

This leads to the following set of N equations in N unknowns:

$$\begin{bmatrix} C(h_{11}) & \cdots & C(h_{1N}) \\ \vdots & \ddots & \vdots \\ C(h_{N1}) & \cdots & C(h_{NN}) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} C(h_{10}) \\ \\ C(h_{N0}) \end{bmatrix}, \tag{8.5}$$

where $C(h_{ij})$ is the covariance function defined in equation (8.2). Equation (8.5) is referred to as simple kriging (SK) and the weighting coefficients can be computed using a standard matrix inversion technique, in which we note that we are dealing with a symmetric matrix. Note that the matrix on the left hand side of the equation contains the covariances between the known points, and the vector on the right hand side contains the covariances of the known points to the unknown points. These covariances are computed from the fitted variograms shown in Figure 8.8 and given in mathematical form in equation (8.3).

A problem with simple kriging is that we do not know the mean of the true values to be estimated. This problem can be avoided using the Lagrange multiplier technique, which results in the ordinary kriging (OK) systems of equations

$$\begin{bmatrix} C(h_{11}) & \cdots & C(h_{1N}) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ C(h_{N1}) & \cdots & C(h_{NN}) & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ \mu \end{bmatrix} = \begin{bmatrix} C(h_{10}) \\ \vdots \\ C(h_{N0}) \\ 1 \end{bmatrix}, \tag{8.6}$$

where μ is the Lagrange parameter.

As in simple kriging, the covariances are computed from the fitted variograms shown in Figure 8.8 and given in mathematical form in equation (8.3). Figure 8.9 shows the result of ordinary kriging using the well values for our case study. Notice that the map shows only the general trends, which consist of high porosity on the left side of the map and low porosity to the right side of the map. The detail that we would expect in a map of porosity is not present.

To estimate the statistical validity of our maps, we can use two approaches. The first is to use the kriging variance error, which is defined as:

$$\sigma_{sk}^2 = C(0) - \sum_{i=1}^{N} w_i C(h_{i0}) . \tag{8.6}$$

where *C(0)* represents the zero offset covariance. Figure 8.10 shows the kriging error in our case.

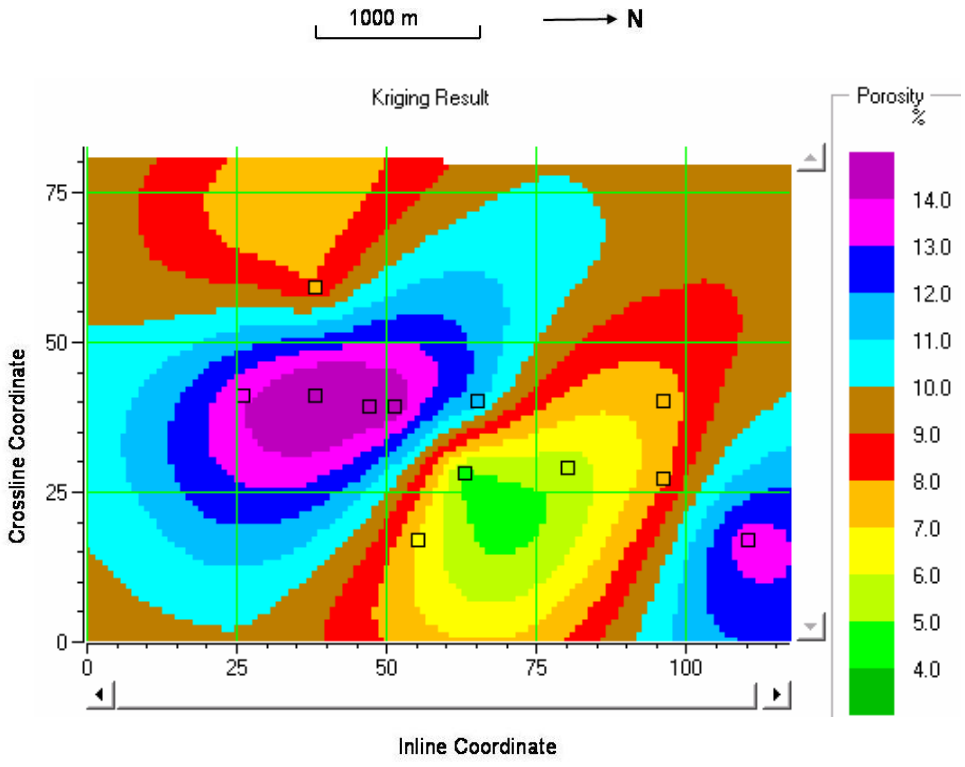


Figure 8.9: Map of the survey area produced by kriging the well porosities.

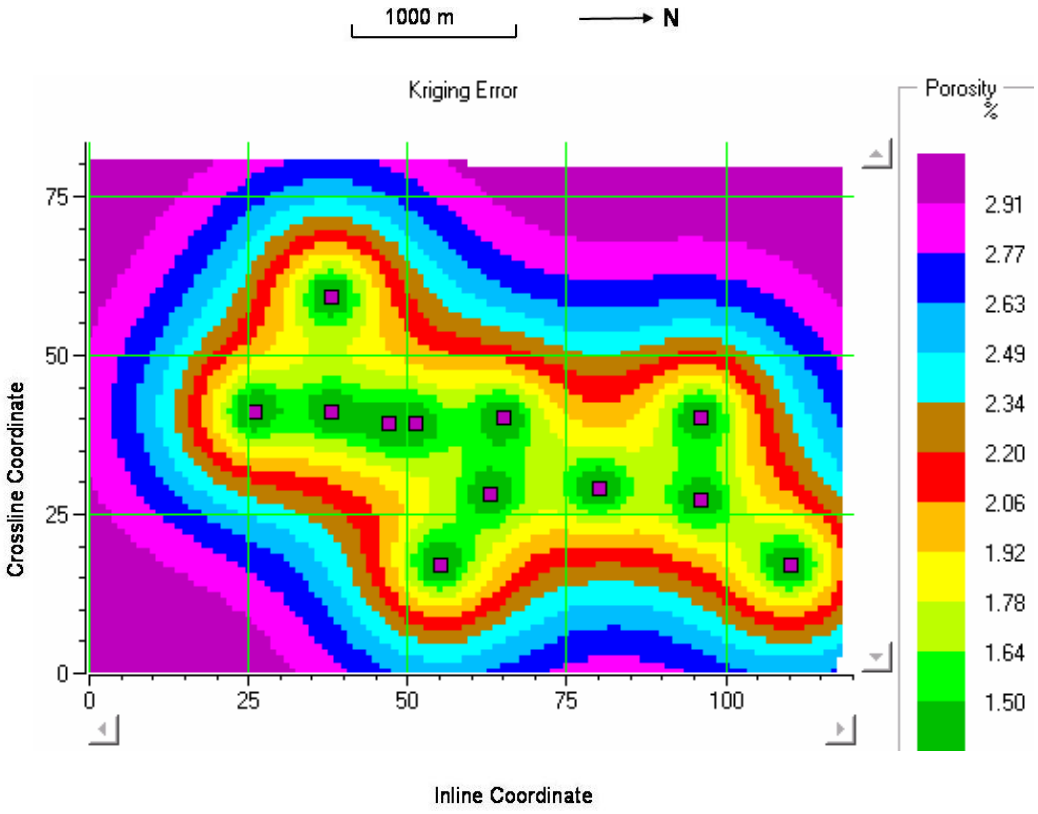


Figure 8.10: The kriging error for the kriged result of Figure 8.9

A second, more unbiased estimate measure of the error is the cross-validation error, which is found by successively leaving out wells and predicting their value. The cross-validation error for the kriged map is shown in Figure 8.11. The RMS average of these errors is 3.242 %.

If we use both datasets, the techniques of kriging with external drift (KED) or cokriging can be used to produce an optimal map. By optimal, I mean that we honour the well data at the well locations and the trend of the seismic data away from the well locations. For both these methods, the well log data is considered the primary dataset, and the seismic is the secondary dataset. The quality of the final maps can be determined either through the cross-validation technique in which we leave each well out in turn and blindly predict it value, or a display of the error variance at each estimated point

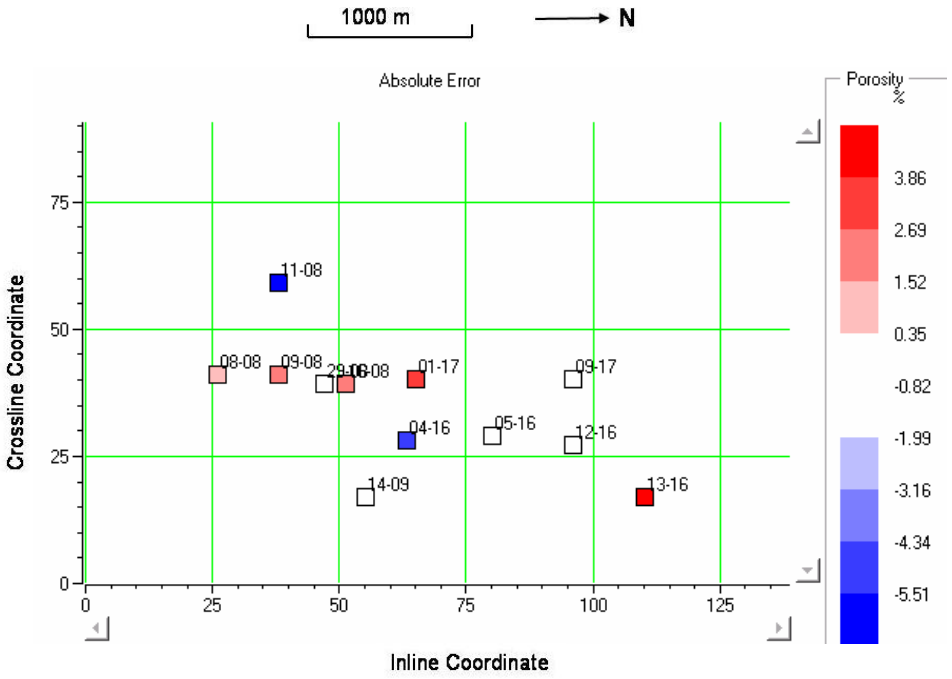


Figure 8.11: The cross-validation errors for the kriged map of Figure 8.9.

Cokriging is an extension of kriging that uses a weighted sum of both well and seismic values. In collocated cokriging, only the seismic value at the output location is used. The equation for collocated cokriging is given as:

$$\phi(x_0, y_0) = w_1\phi(x_1, y_1) + w_2\phi(x_2, y_2) + \ldots + w_N\phi(x_N, y_N) + w_{N+1}AI(x_0, y_0), \quad (8.7)$$

where AI is the acoustic impedance. Thus, we need to extend the theory of ordinary kriging to compute $N+1$ weights. Because we are including the seismic values, we now need to include both the seismic-to-seismic and seismic-to-well variograms. The final set of equations can be written in matrix form as

$$\begin{bmatrix} C_{ww}(h_{11}) & \cdots & C_{ww}(h_{1N}) & C_{ws}(h_{10}) & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ C_{ww}(h_{N1}) & \cdots & C_{ww}(h_{NN}) & C_{ws}(h_{N0}) & 1 & 0 \\ C_{ws}(h_{10}) & \cdots & C_{ws}(h_{N0}) & C_{ss}(0) & 0 & 1 \\ 1 & \cdots & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ w_{N+1} \\ \mu_1 \\ \mu_2 \end{bmatrix} = \begin{bmatrix} C_{ww}(h_{10}) \\ \vdots \\ C_{ww}(h_{N0}) \\ C_{ss}(0) \\ 1 \\ 1 \end{bmatrix}, \quad (8.8)$$

where $C_{ww}(h_{ij})$ represents the well-to-well covariance at the h_{ij} offset, $C_{ws}(h_{ij})$ represents the well-to-seismic covariance at the h_{ij} offset, and $C_{ss}(h_{ij})$ represents the well-to-well covariance at the h_{ij} offset, $C_{ws}(h_{ij})$ represents the seismic-to-seismic covariance at the h_{ij} offset. In the case of collocated cokriging, only the zero offset term is used for the seismic-to-seismic covariance. Note that for collocated cokriging there are two Lagrange parameters, one for the well terms and one for the well to seismic terms.

In equation (8.8), we show that three variograms are needed to compute the weighting coefficients: the well-to-well variogram, the well-to-seismic variogram, and the seismic-to-seismic variogram. However, in many cases we have much more confidence in the seismic-to-seismic variogram, which in the case of a 3D seismic dataset is derived from a much greater number of points than the well-to-well or well-to-seismic variograms. In this case, we can use the Markov-Bayes assumption, which assumes that there is a linear relationship between our datasets given by

$$AI(x,y) = a\phi(x,y) + b + noise, \tag{8.9}$$

where $AI(x,y)$ is the acoustic impedance at spatial locations x and y derived from the seismic data, $\phi(x,y)$ is the porosity derived from the well logs, and a and b are constants as determined in the crossplot of Figure 8.6.

This leads to the following two relationships among the three covariance functions

$$C_{SS}(\mathbf{h}) = a^2 C_{WW}(\mathbf{h}) + C_{NN}, \tag{8.10}$$

and

$$C_{WS}(\mathbf{h}) = a C_{WW}(\mathbf{h}), \tag{8.11}$$

where C_{NN} is the noise covariance, given by

$$C_{NN} = \frac{C_{WS}(0)}{\left(C_{WW}(0) \cdot C_{SS}(0)\right)^{1/2}}. \tag{8.12}$$

In Figure 8.12 the well and seismic data have been combined using collocated cokriging with the Markov-Bayes assumption being used to derive all three variograms from the seismic-to-seismic variogram.

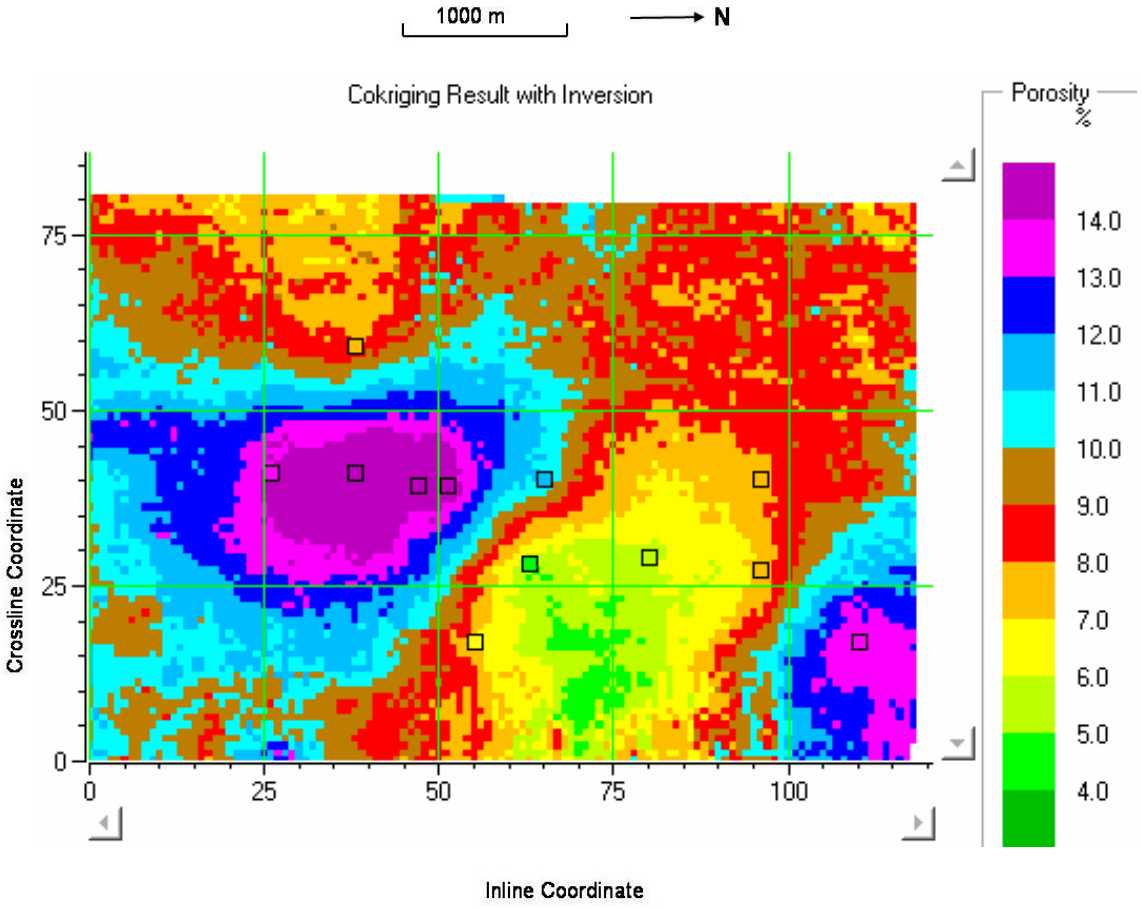


Figure 8.12: Map of porosity in the survey area produced by collocated cokriging between the averaged well porosities and the impedance slice.

Although the map in Figure 8.12 shows the imprint of the wells from the kriged result, the final look of the map is more realistic because of the inclusion of the seismic impedance data. Again, we can compute the error either from a similar technique defined for the kriged result, or by using the cross-validation error. In this case, the RMS average of the cross-validation error is 3.027%, which is lower than the error for kriging.

8.4 Map attributes

I will now discuss an improvement to the geostatistics approach, in which we pre-condition the seismic map using the multi-attribute approach discussed in this dissertation. The key difference between what I am doing in this chapter compared to what was done in earlier chapters is that I am using map attributes rather than volume attributes. That is, a series of time averaged slices were extracted from the seismic cube. These slices were averaged over a 10 ms window below the zone of interest using an RMS average, rather than an arithmetic average, since many of the attributes had zero mean (that is, both positive and negative values).

The six slices that were extracted consisted of seismic amplitude, amplitude envelope, instantaneous phase, cosine instantaneous phase, trace length, and integrated trace. In the case of trace length, no averaging was performed since this attribute measures the length of the trace over the zone of interest. These slices are shown in Figure 8.13. Notice that each slice shows the channel in a slightly different way, since each attribute looks at the information over the volume slice in a different way. For example, we see that the amplitude and amplitude envelope slices of Figure 8.13(a) and (b) are very similar, and show a narrow channel. However, the instantaneous phase and frequency slices of Figure 8.13(c) and (d) show a broader channel with a dominant phase of less than 112 degrees and a dominant frequency of greater than 69 Hz. The integrated trace slice of Figure 8.13(e) again shows a broad channel, but the trace length slice of Figure 8.13(f) does not show the channel very well.

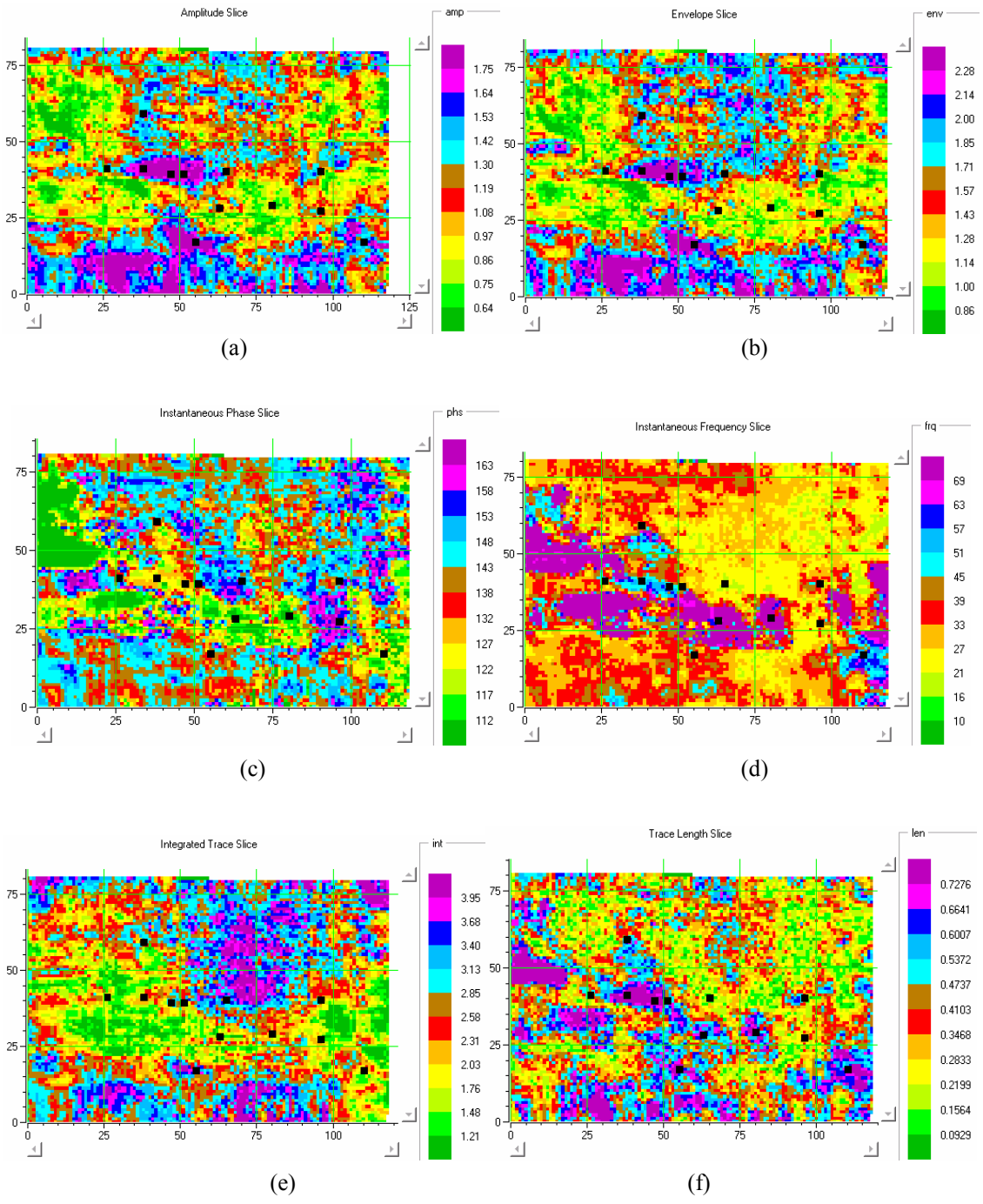


Figure 8.13: Attribute slices derived from the original seismic volume, where (a) shows seismic amplitude, (b) shows amplitude envelope, (c) shows instantaneous phase, (d) shows instantaneous frequency, (e) shows integrated trace, and (f) shows total trace length. The first five attributes consist of an RMS average over a 10 ms window below the picked channel top.

8.5 The multiattribute transform

The multiattribute transform method involves the same input data as in the geostatistical method just described, except that multiple secondary sets of seismic attributes are used. Recall that seismic attributes were discussed at length in Chapter 2. Another key difference between this method and the geostatistical method is that the multiattribute transform does not force an exact solution at the well-to-seismic intersections. Instead, a "best-fit" relationship is derived at the well-tie points, which is then applied to the multiple input attributes to produce the reservoir volume. Two approaches can be used to derive this relationship: multilinear regression and neural network analysis. I will discuss the multilinear regression approach in this section and the neural network approach in a later section. Using multilinear regression we seek a set of weights which, when applied to the attribute maps, will produce the reservoir parameter map. A pictorial illustration of this approach is shown in Figure 8.14.

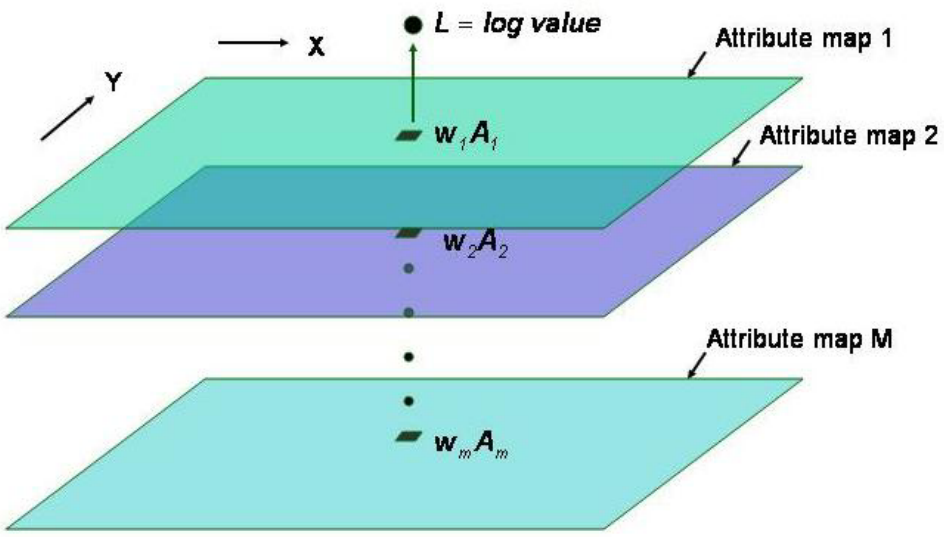


Figure 8.14: A pictorial illustration of the multilinear regression method of combining map attributes.

Mathematically, we model the log parameter map $L(x,y)$ as a weighted sum of the M attribute maps $A_j(x,y)$ by the linear equation

$$L(x,y) = w_0 + w_1 A_1(x,y) + \cdots + w_M A_M(x,y)\,. \tag{8.13}$$

In equation (8.13) both the estimated log values and the attributes are a function of the map coordinates *x* and *y*, rather than of time, *t*, as in our applications in the earlier chapters. As also discussed in earlier chapters, we can expand the linear approach by applying nonlinear transforms to the attributes, such as the logarithm, the square root, etc.

In the multilinear regression approach we will compute the number and order of the statistically valid attribute slices using the cross-validation method, which was also discussed at length in previous chapters. In the cross-validation approach, the prediction error is computed by leaving out the target well for each of the input points, calculating the resulting least-squares error, and then summing the results. The validation error will reach a minimum at a relatively small number of attributes, usually less than five.

The correlation between each slice and the porosity map is shown in Table 8.1, where the inversion slice has the highest correlation coefficient, the amplitude (amp) slice has the second highest correlation coefficient, and so on down to the integrated trace length slice (int), which has the lowest correlation coefficient.

Well Log Data	Seismic Data	Correlation
Porosity	Inversion	-0.654939
Porosity	amp	0.405974
Porosity	env	0.320297
Porosity	frq	-0.274947
Porosity	phs	0.153302
Porosity	len	0.110205
Porosity	int	0.0295412

Table 8.1: The correlation coefficients for the seven attribute slices.

A multiattribute analysis was then performed at the well locations using the multilinear regression algorithm. The cross-validation results for this analysis are shown in Figure 8.15. In this figure the bottom curve (in black) shows the result of using all the wells in the training, and the top curve (in red) is the cross-validation result, in which the wells are left out of the computation and then predicted. It is clear in Figure 8.15 that only the first three attributes are statistically significant, since the error on the top curve increases after the third attribute. Table 8.2 shows a numerical summary of these attributes. Note that the three attributes were, in order, the inverse of impedance, instantaneous phase, and integrated trace. Also note that the well parameter is the square root of porosity.

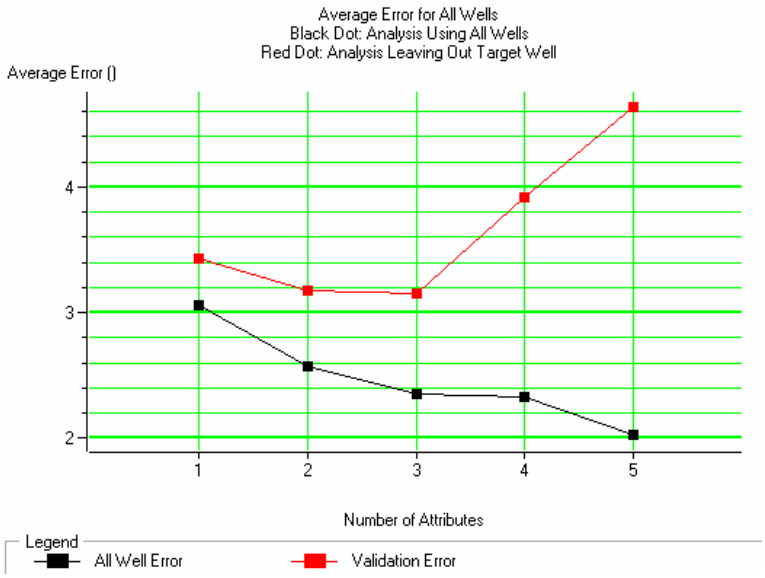


Figure 8.15: The average error for the best five attribute found by multi-linear regression, where the bottom curve (black) shows the total error and the top curve (red) shows the validation error.

	Target	Final Attribute	Training Error	Validation Error
1	Sqrt(Porosity)	1 / (Inversion)	3.062634	3.428392
2	Sqrt(Porosity)	1 / (phs)	2.578627	3.173227
3	Sqrt(Porosity)	1 / (int)	2.353296	3.150217
4	Sqrt(Porosity)	(X-coordinate)**2	2.328632	3.731974
5	Sqrt(Porosity)	1 / (env)	2.286716	4.521664

Table 8.2. The training and validation errors for the attribute slices used in the multiattribute computation.

Table 8.3 then shows the derived weights used in the multiattribute transform. These weights are the coefficients for equation 8.13.

Attribute Name	Attribute Transform	Weight
Inversion	1/X	110045
phs	1/X	-265.721
int	1/X	-1.24719
Constant		-5.05961

Table 8.3. The weights used in the multiattribute computation.

The map that results from applying the derived weights to the attribute slices is shown in Figure 8.16. Note the increased resolution of the high porosity sand channel.

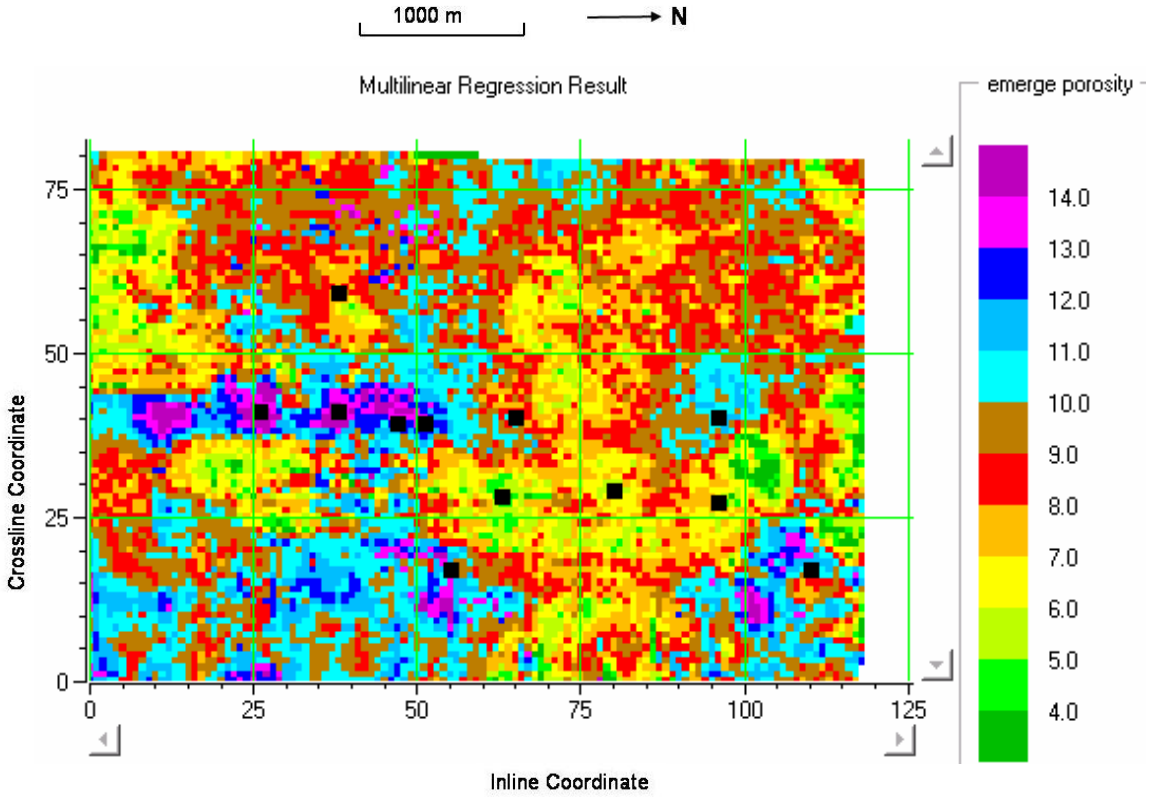


Figure 8.16: The application of multi-linear regression using the weights and attributes shown in Table 8.3.

To see how well we have done statistically, Figure 8.17 is a crossplot of the actual porosity values against the predicted porosity values. Notice that the correlation coefficient has now increased to 0.85, compared with the -0.65 value for the best attribute slice, the impedance.

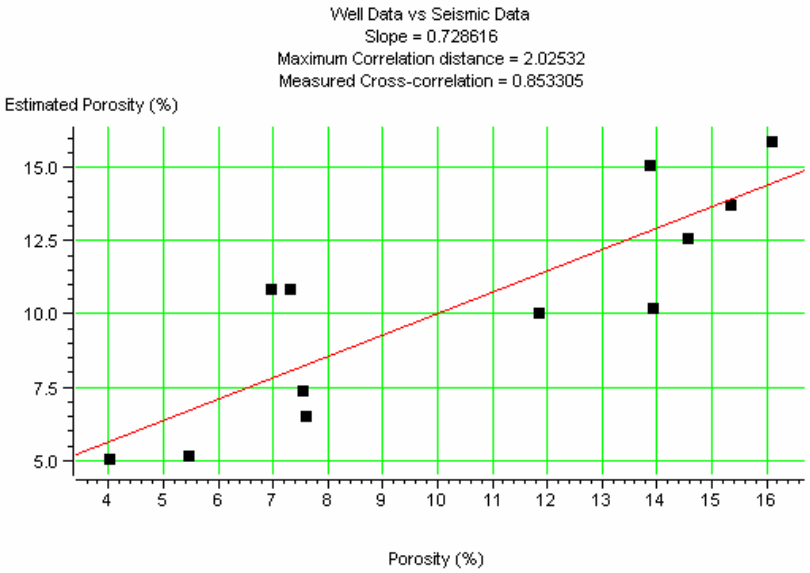


Figure 8.17: The crossplot of the actual well-log derived porosity (horizontal axis) against the estimated porosity in the result shown in Figure 8.15. Note that the correlation coefficient is 0.85.

Although I have described the geostatistical method before the multiattribute method, the actual order of application is the reverse of this. That is, we first use the multiattribute method to produce an improved map as the secondary attribute for the geostatistical method. We next apply the technique of collocated cokriging to this new map. As mentioned previously, we must first re-compute the variogram, and this is shown in Figure 8.18. In the previous variogram, a spherical function was used, which was given in equation (8.3). In this case, a better fit was obtained with an exponential function, which can be written

$$\gamma(h) = \gamma_0 + s\left[1 - \exp\left(-\frac{h}{a}\right)\right], \tag{8.14}$$

where s is the sill, γ_0 is the nugget and a is the range, all as defined previously.

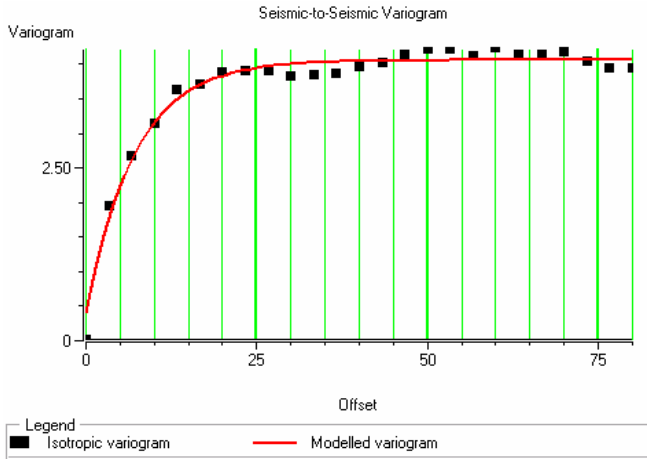


Figure 8.18: The recomputed seismic variogram used for collocated cokriging with the multi-linear result of Figure 8.15 as the secondary dataset.

The cokriged porosity result is shown in Figure 8.19. Notice that the channel sand is now clearly delineated, and the fit to the wells is very good. In this case, the RMS average of the cross-validation error is 2.534%, which is lower than the result found using cokriging with the impedance attribute.

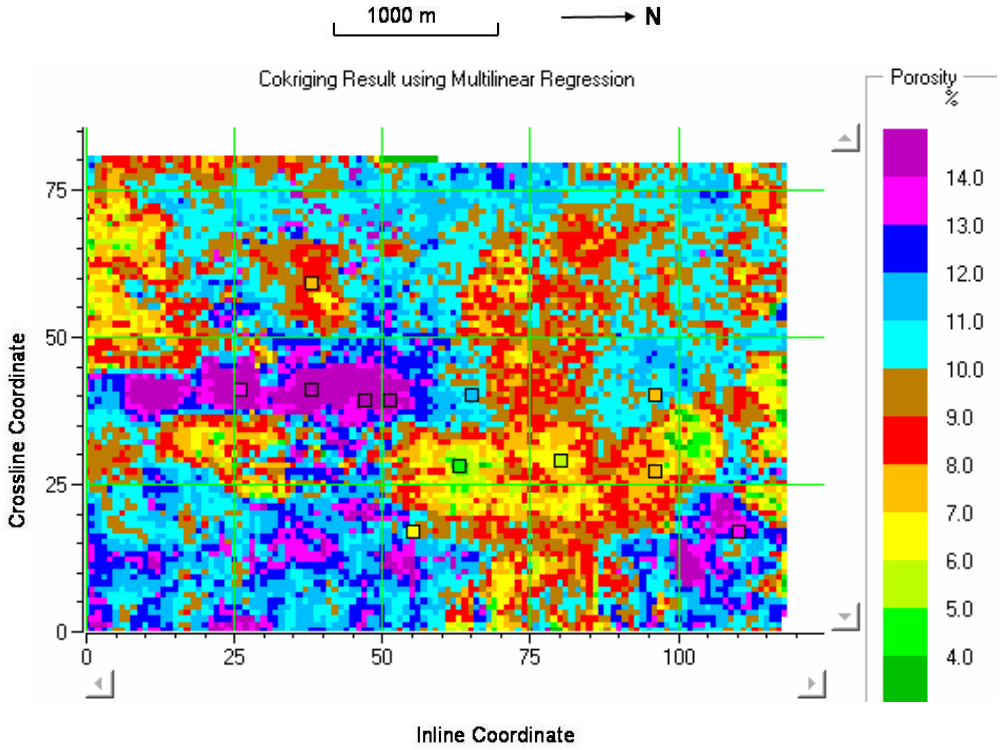


Figure 8.19: The result of applying cokriging to the multi-linear regression result.

8.6 Neural network mapping

I will now apply several neural network algorithms that have been to the map dataset. Specifically, I will use the three key algorithms that have been discussed in this dissertation: the multi-layer perceptron (MLP), the generalized regression neural network (GRNN), and the radial basis function neural network (RBFN). As input to these algorithms I will use the attributes computed using multi-linear regression with cross-validation analysis as shown in Figure 8.15 and Table 8.2.

I will start with the multi-layer perceptron, which was fully described in Chapter 5 of this dissertation. Figure 8.20 shows the result of applying the multi-layer perceptron with 5 nodes in the hidden layer and 10 iterations in the weight computation.

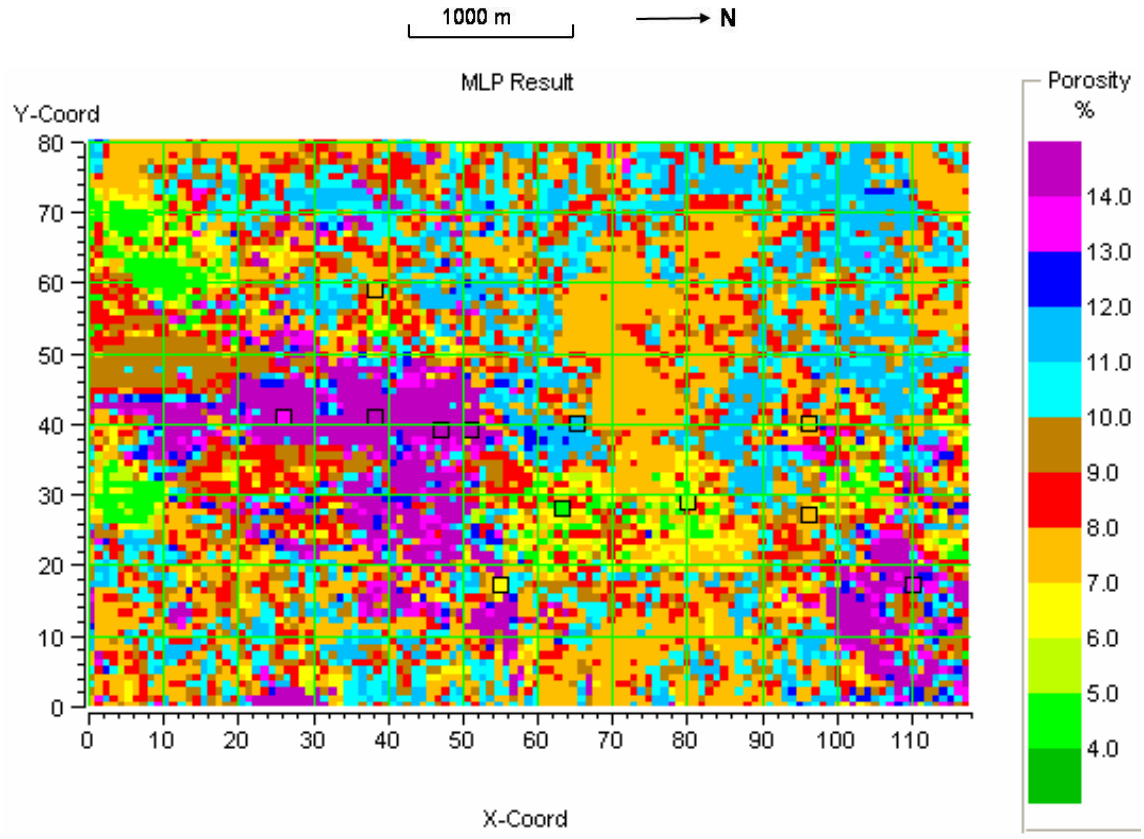


Figure 8.20: The computed porosity map using the multi-layer perceptron applied to the first three attributes shown in Table 8.2.

Although the main trend of the high porosity channel has been revealed in Figure 8.20, there appears to be a lot of "noise" in the result, which suggests that the MLP has been overtrained. This overtraining is indeed in evidence when we look at the cross plot of the actual well log porosity values against the predicted values, shown in Figure 8.21. Notice that the fit is too good for most of the points, since they fall on a straight line, and not good enough for the rest of the points, since they all have the same value of 15%. The overall correlation coefficient is very close to 1.0, which implies perfect correlation.

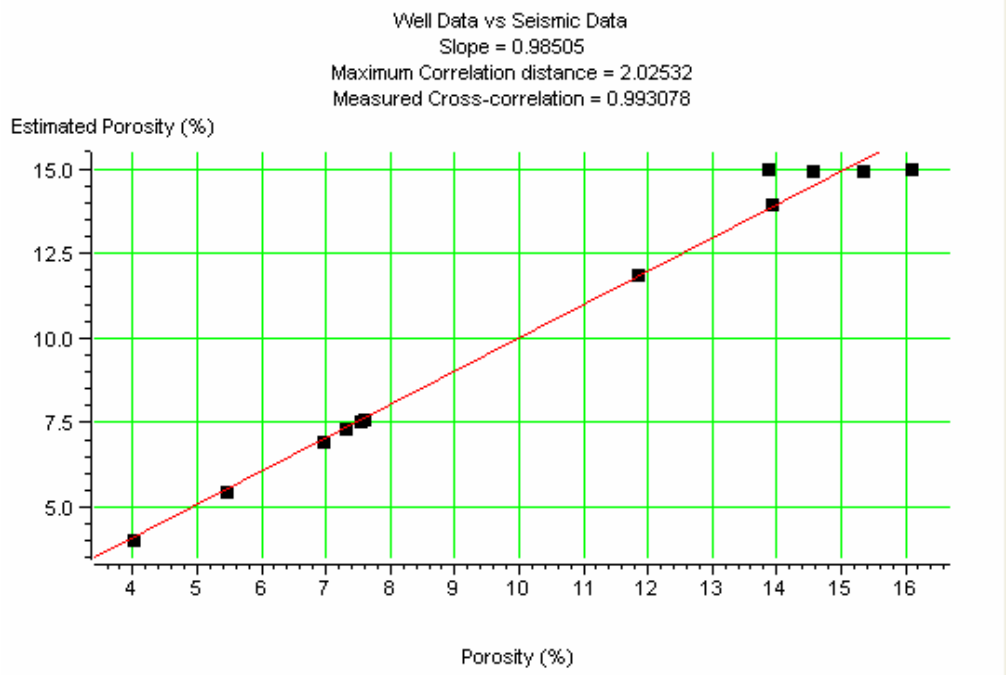


Figure 8.21: The crossplot of the actual well-log derived porosity (horizontal axis) against the estimated porosity for the MLP result shown in Figure 8.19. Note that the correlation coefficient is 0.85.

Next, I will apply the generalized regression neural network (GRNN) algorithm, which was fully described in Chapter 6 of this dissertation. You will recall that the GRNN algorithm involves performing a weighted sum of the input values multiplied by a set of Gaussian functions of the square of the distance between the attribute vectors of the output values and those of the input values. The key parameter that needs to be modified is σ, which controls the width of the Gaussian functions.

Figure 8.22 shows the result of applying the GRNN algorithm, in which σ was optimized using cross-validation, and allowed to vary as a function of the three attributes used as input. These attributes are the first three attributes shown in Table 8.2. As with the MLP algorithm, the main trend of the high porosity channel has been revealed. However, there again appears to be a lot of "noise" in the result, which suggests that the GRNN has been overtrained.

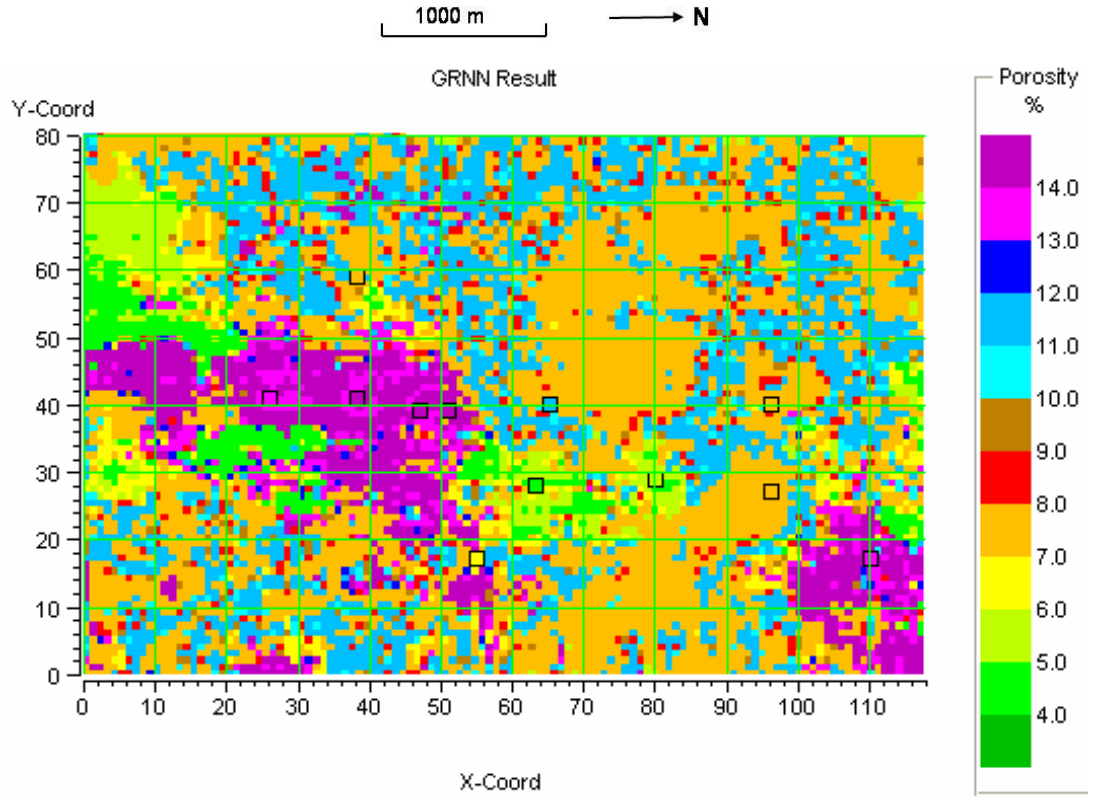


Figure 8.22: The computed porosity map using the generalized regression neural network (GRNN) applied to the first three attributes shown in Table 8.2.

As shown with the MLP result, this overtraining is obvious when we look at the cross plot of the actual well log porosity values against the predicted values, shown in Figure 8.23. In this figure we see that the points fall on the line of perfect correlation and have a correlation coefficient almost identical to 1.0. The initial results of applying neural networks to the map case are thus not too encouraging.

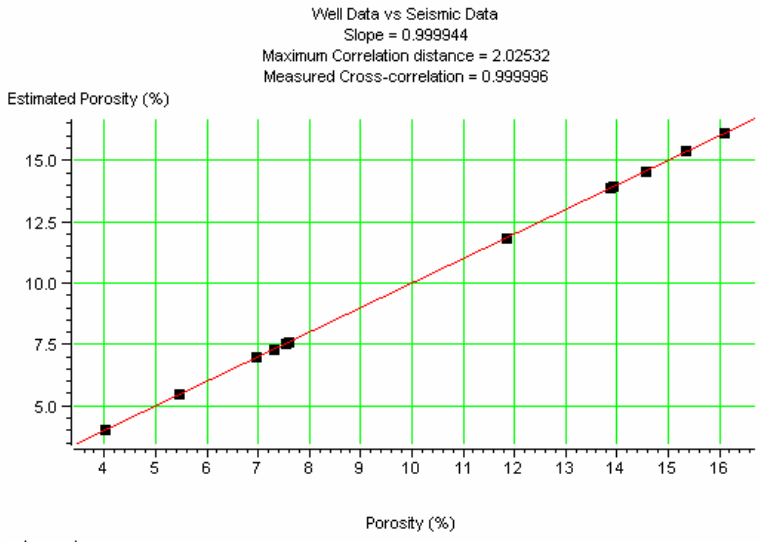


Figure 8.23: The crossplot of the actual well-log derived porosity (horizontal axis) against the estimated porosity for the GRNN result shown in Figure 8.22. Note that the correlation coefficient is almost perfect, but this suggests overtraining.

I will next apply the radial basis function neural network (RBFN), which was fully discussed in Chapter 6. As with the GRNN approach, the RBFN uses Gaussian functions of distance. But, unlike GRNN, a set of weights are pre-computed from the training data, and are then applied to the attribute distance functions. The key parameter to optimize is again σ, which controls the widths of the Gaussian basis functions.

Figure 8.24 shows the result of applying the RBFN algorithm, in which σ was optimized using the parabolic search technique described in section 6.5.2. These attributes are the first three attributes shown in Table 8.2. Unlike our results using the MLP and GRNN algorithms, the map shown in Figure 8.24 does not show any noise bursts and is actually quite similar to the multi-linear result shown in Figure 8.16. Notice that the high porosity channel has been very well delineated. To see if there is any improvement over the multi-linear result, we will again crossplot the actual and predicted porosity values. This crossplot is shown in Figure 8.25, and shows a correlation coefficient of 0.869. This value is an improvement over the value of 0.85 found using multi-linear regression, but is not indicative of over-training, as were the crossplots for both the GRNN and MLP results, shown earlier.

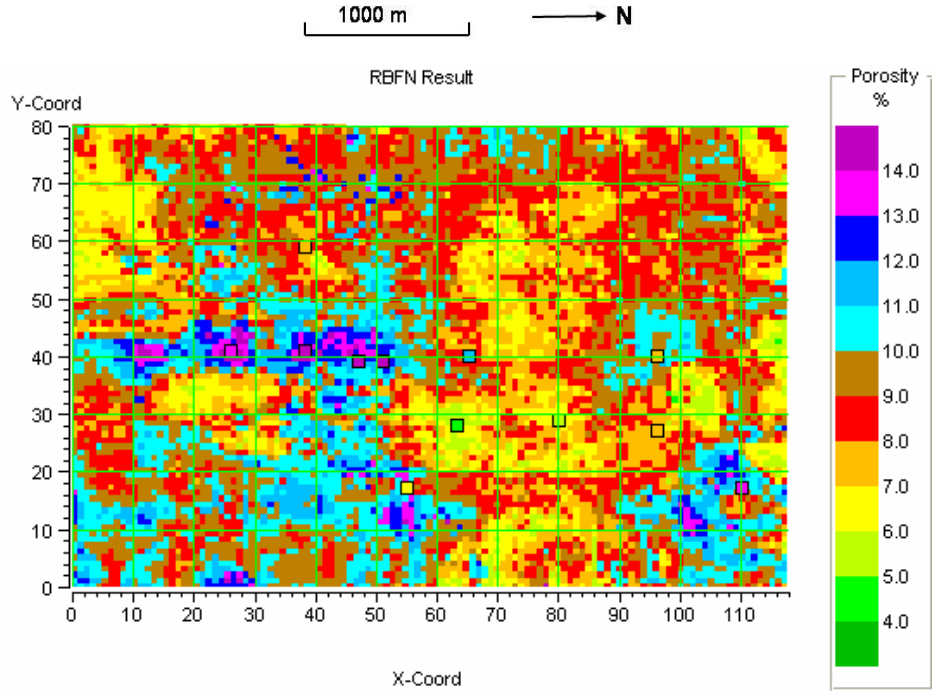


Figure 8.24: The computed porosity map using the radial basis function neural network (RBFN) applied to the first three attributes shown in Table 8.2.

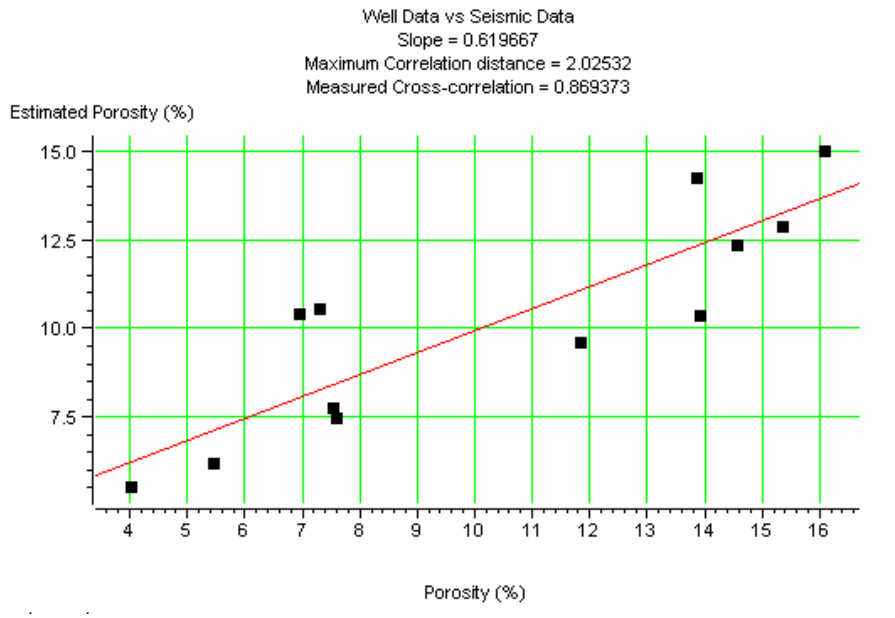


Figure 8.25: The crossplot of the actual well-log derived porosity (horizontal axis) against the estimated porosity for the RBFN result shown in Figure 8.23. Note that the correlation coefficient is equal to 0.869.

I next apply the technique of collocated cokriging to this new map produced by the RBFN algorithm. As mentioned previously, we must first re-compute the variogram,

and this is shown in Figure 8.26. As with the multi-linear regression map, the best fit was obtained with an exponential function, which was written in equation (8.14).

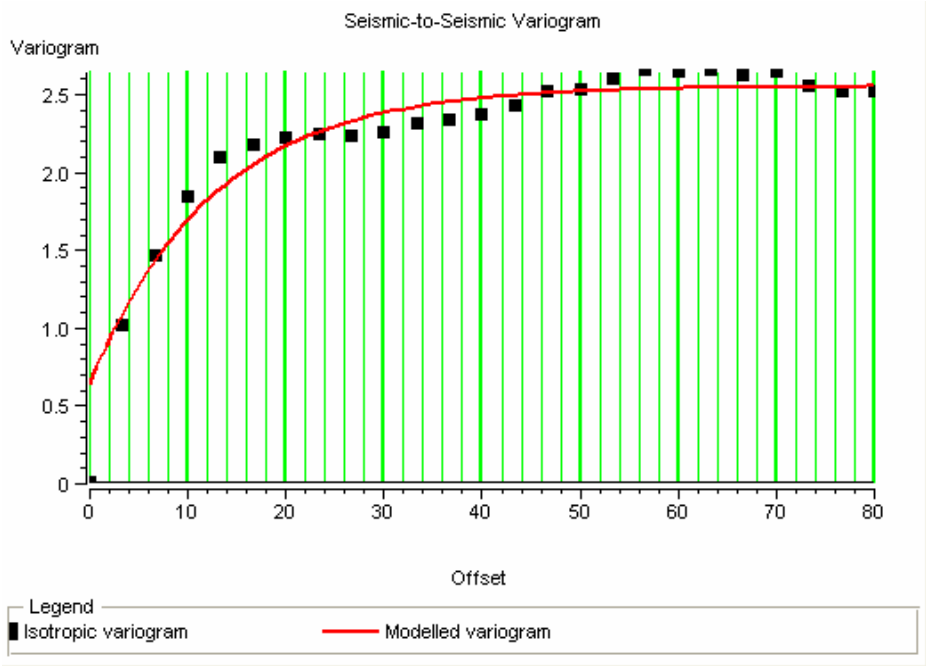


Figure 8.26: The recomputed seismic variogram used for collocated cokriging with the RBFN result of Figure 8.24 as the secondary dataset.

The final cokriged porosity result is shown in Figure 8.27, in which the RBFN result of Figure 8.25 has been used as the secondary input. Notice that the channel sand is again clearly delineated and that the fit to the wells is very good. This result is very similar to the result obtained using multilinear regression, but gives a slightly improved cross-validation result. In this case, the RMS average of the cross-validation error is 2.516%, which is lower than the result found using cokriging with the multi-linear attribute. As a final summary, Table 8.3 lists the cross-validation results at each of the wells, and their RMS averages, for all of the methods discussed in this chapter.

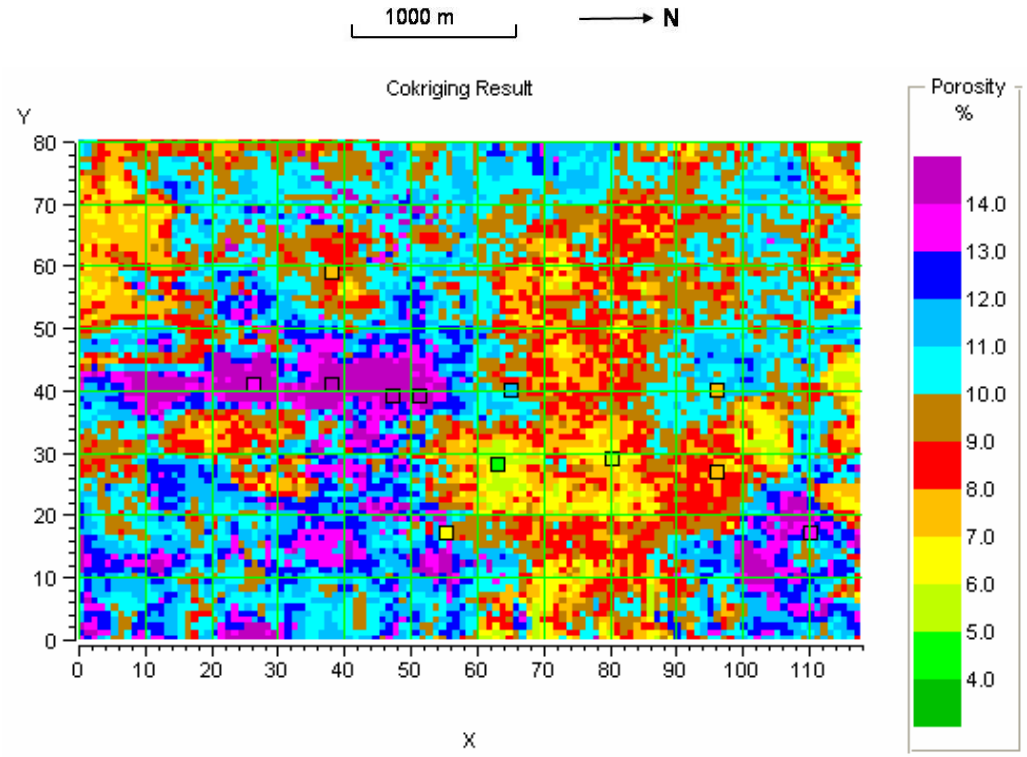


Figure 8.27: The result of applying collocated cokriging using the RBFN result of Figure 8.24 as the secondary dataset.

			Cokriged:		
X	Y	Kriged	Impedance	Multilinear	RBFN
80	29	-0.662	-1.338	-1.694	-1.842
55	17	-0.665	-1.367	-5.171	-4.213
96	27	-1.190	-0.372	-0.160	-1.043
110	17	5.040	4.287	2.801	3.042
63	28	-5.354	-5.835	-2.944	-3.518
65	40	3.318	4.258	2.847	3.054
96	40	-1.322	-1.753	-3.884	-3.489
26	41	1.032	-0.222	-1.610	-1.334
38	41	2.552	1.186	-0.333	0.456
47	39	-0.427	-0.593	-0.203	0.201
38	59	-6.652	-5.170	-2.066	-2.518
51	39	2.233	1.996	0.843	1.497
	RMS:	3.242	3.027	2.534	2.516

Table 8.4: A cross-validation analysis of the errors at each well using each of the mapping methods. The units in the table are % porosity.

8.7 Conclusions

In this chapter I have presented a new approach to the integration of well log and seismic data, which combines the methods of seismic inversion, geostatistics and multiattribute transforms. Inversion is used as the starting point, since I have found that the inversion results correlate much better with geology than the original seismic. However, by combining multiple seismic attributes I was able to bring in extra information to enhance the final well tie. It is also important to use the technique of validation to make sure that we are not adding spurious attributes to the final solution. Finally, geostatistics gives us a powerful set of tools for producing our final map, which combines the multiattribute transformed map with the well values, giving priority to the well information. The geostatistical results produced in this chapter were created using a commercial implementation of the GSLIB Geostatistical Software Library (Deutsch and Journel, 1992) from Hampson-Russell Software called ISMap.

Our approach was tested using a channel sand example from Alberta. I used four separate approaches to build a map of well porosity: kriging with the wells values alone, collocated cokriging with the impedance as the secondary dataset, collocated cokriging with the multi-linear regression output as the secondary dataset, and collocated cokriging with the RBFN output as the secondary dataset. I found that the RMS error of the cross-validation results progressively improved from 3.242 for kriging to 2.516 for the RBFN method, and that the final result clearly delineated the channel sand and provided an excellent match to the wells. I also found that the MLP and GRNN neural networks tended to be overtrained with this dataset, leading to noisy results. The fact that the RBFN network worked best is due to the fact that RBFN is the best method to use as the number of training points goes down. Since we only had 12 training points in this case, this is the ultimate test of these methods.

CHAPTER 9 : SUMMARY AND CONCLUSIONS

9.1 Summary

In this dissertation, I have examined the relationship between seismic attributes and reservoir parameters such as porosity. Figure 1.5 shows this relationship for the case that was considered in Chapter 2. Note that we are trying to predict the target log on the left from the three attributes on the right. In Figure 1.5, the target log $\boldsymbol{t}$ represents the known seismic reservoir parameter and the attribute vectors $\boldsymbol{a_i}$ represent the seismic attributes from a seismic volume that correspond spatially and temporally to this reservoir parameter, where i goes from 1 to M. Although only three attribute vectors are shown in Figure 1.5, we can use as many in our analysis as are statistically valid. The sample vectors $\boldsymbol{x_j}$, where j ranges over the number of seismic samples, are associated with the individual training samples t_j, and can be considered as the transpose of the attribute vectors (see Appendix 1 for the mathematical details).

Throughout this study, the relationship between the seismic attributes and the reservoir parameter was quantified using various linear and nonlinear mappings derived from multivariate statistics and neural network theory. We can write this relationship as

$$t_j = f(\boldsymbol{w}, \boldsymbol{x}_j), \tag{9.1}$$

where $\boldsymbol{w}$ is a set of weights. Once this mapping has been found, we can apply to the unknown seismic points $\boldsymbol{s}$ to find a new reservoir parameter value y, using the relationship

$$y = f(\boldsymbol{w}, \boldsymbol{s}), \tag{9.2}$$

The most general form of such a mapping using the training data is

$$t_j = f(W^{(q)}, \boldsymbol{x}_j) = \phi\left(W^{(2)T}\left(\phi\left(W^{(1)T}\boldsymbol{x}_j\right)\right)\right), \tag{9.3}$$

where j = 1, ..., N, the number of seismic samples, $W^{(1)}$ and $W^{(2)}$ are weight matrices given by

$$W^{(2)} = \begin{bmatrix} \boldsymbol{w}_1^{(2)T} \\ \vdots \\ \boldsymbol{w}_K^{(2)T} \end{bmatrix} \text{ and } W^{(1)} = \boldsymbol{w}^{(1)T} = [w_1^{(1)} \quad \cdots \quad w_M^{(1)}],$$

the input is given by $\boldsymbol{x}_j^T = [x_{1j} \quad \cdots \quad x_{Mj}]$, and ϕ is a nonlinear function. Note that K represents the number of summing nodes. If we let ϕ be a sigmoid function, equation (9.3) represents the multilayer perceptron (MLP) that was discussed in Chapter 5. If we let $\boldsymbol{\phi(x)} = \boldsymbol{x}$ it was shown in section 6.6 that equation (9.3) reduces to

$$t_j = f(\boldsymbol{w}, \boldsymbol{x}_j) = \boldsymbol{w}^T \boldsymbol{x}_j, \tag{9.4}$$

where $\boldsymbol{w}$ is equal to $W^{(1)}$ in equation (9.1) and $\boldsymbol{x_j}$ is as defined above.

The linear multi-regression approach of equation (9.4) was extensively discussed in Chapter 3, and was proved to be an extremely robust way of mapping attributes into reservoir parameters. In Chapter 4 I discussed the related concept of linear classification, in which the linear functions defined the boundaries between classes.

In both chapters 3 and 4, we found that the limitation of linear methods is their inability to perform nonlinear tasks, such as prediction of data points that do not fall on a linear trend, or separation of classes for which there is a nonlinear boundary. This led us to the multi-layer perceptron, which I have just discussed, and to the radial basis function neural network. The radial basis function neural network combines the ease of implementation of multi-linear regression and the power of the multi-layer perceptron. Mathematically, we can write the radial basis equation as

$$t_j = f(\boldsymbol{w}, \boldsymbol{x}_j) = \sum_{k=1}^{K} w_k \phi(\boldsymbol{x}_j), \tag{9.5}$$

where $\boldsymbol{w}$ is now a K-valued weight vector, $\boldsymbol{x_j}$ is as defined above, $\boldsymbol{\phi(x_j)}$ is a Gaussian basis function given by

$$\phi(\boldsymbol{x}_j) = \exp\left[-\frac{\left|\boldsymbol{x}_j - \boldsymbol{\mu}_k\right|^2}{\sigma^2}\right], \tag{9.6}$$

the $\boldsymbol{\mu}_k$ are a set of mean values, and σ is a scaling value. Note that equation (9.5) is the formulation for RBFN with centres, and can be expanded to the full RBFN method by letting *K* go to *N*, and replacing the $\boldsymbol{\mu}_k$ values with the *N* seismic samples. The radial basis function approach was discussed in Chapters 6 and 7. In Chapter 6, I discussed the general theory and showed that there are three closely related methods which use Gaussian basis functions: the probabilistic neural network (PNN), the generalized regression neural network (GRNN), and the radial basis function network (RBFN), and in Chapter 7 I discussed the RBFN with centres method.

In Chapters 2 through 7, I considered the mapping procedure as one in which we transform a set of 3D seismic attribute volumes into a volume of reservoir values. In Chapter 8 I applied these methods to map analysis, and combined this with the theory of geostatistics.

9.2 Conclusions

Throughout this dissertation I have used both model and real datasets to illustrate the theory behind the prediction of reservoir parameters from seismic attributes. This has lead me to a number of conclusions about the effectiveness of each method, which I will now discuss.

The first technique that was applied to the prediction of reservoir parameters was multilinear regression. This is a well established technique that is fully discussed in the multivariate statistics literature, and a summary of this theory was given in Chapter 3. I found that multilinear regression gave extremely robust, reproducible results but that the error between the known training values and the predicted results tended to be large. An extension of the classical multilinear regression technique, suggested by Hampson et al.

(2001), was to use convolutional filters instead of single points in multilinear regression. As I show mathematically in section 3.5.4, this is equivalent to creating a set of new seismic attributes that are the time-shifted versions of the original attributes at the shifts corresponding to the convolutional filters. I also show in Appendix 3 that this approach can be derived from the more general multichannel filter (Treitel, 1970). Another way in which the results of multilinear regression can be improved, also suggested by Hampson et al. (2001), is to apply nonlinear functions such as log, square root, etc, to either the attributes or the target log or both before analysis. Using this approach, I was able to build a new relationship between S-wave velocity and two other well log attributes: the P-wave sonic log and the square root of the gamma ray log.

As a measure of the statistical validity of the multilinear regression method, I used the cross-validation approach, in which we successively drop known values out of the training dataset and "blindly" predict these values. This method gave us an excellent way of ordering the attributes and removing those that did not improve the overall error. This approach is used to find the order and number of attributes for each of the nonlinear methods used in subsequent chapters.

In Chapter 4, linear classification methods were discussed and were shown to be closely related to multilinear regression. Two applications of these methods were considered. The first was linear discriminant analysis, in which we divided a set of reservoir parameters into a number of classes based on the linear separation between these classes. This approach was applied to a well log and seismic dataset from the Blackfoot area of Alberta, in which three classes of porosity (low, medium, and high) were classified. The results indicated the presence of a known high porosity channel, but also contained some spurious high porosity zones. The second application of linear classification was the single layer perceptron (SLP). When applied to a class 3 AVO anomaly, the SLP was able to correctly identify either the top or base of the sand, but not both simultaneously, since this was a nonlinear problem.

In Chapter 5, I extended the single layer perceptron to the multi-layer perceptron (MLP), which is also often called the multi-layer feedforward neural network. Continuing the work on the class 3 AVO anomaly, I was able to show that the MLP was able to simultaneously identify the top and base of the sand. I then discussed the mathematics behind error backpropagation in the MLP and applied this theory to both the AVO classification problem and a sine wave prediction problem. The MLP was then applied to the prediction of P-wave velocity in the Blackfoot dataset. In our applications to both real and model datasets, we found that the MLP was able to predict the training data quite accurately, but often displayed the following undesirable characteristics

- the solution of the weights was highly dependent on the initial guess.
- the solution could be very slow to converge.
- the result were often "overtrained", meaning that there was a good fit to the training data but poor generalization to the unknown values.

Before discussing the last three chapters of this dissertation, which contain my key argument, it is important to recall the discussion of the "bias versus variance" dilemma which was presented in section 1.5. Let me quote from that section:

> "On one hand, we do not want a model that is too simple and does not have the flexibility to ever fit our data points. This is called a model with high bias. On the other hand, we do not want a model that is too complex and has such a high degree of flexibility that it overfits the data. This is called a model with high variance. The optimal model would be complex enough to fit the data values reasonably well, but not so complex that it fits the noise in the data."

In the light of this quote, let me summarize the first two approaches I used for the prediction of reservoir parameters. I found that the multilinear regression approach tended to have high bias because the linear model is too simple to predict earth

properties. However, I found that the multilayer perceptron was too complex of an algorithm and tended to have high variance. We needed to find a better method.

The answer was to find a method that combines the best properties of both methods, while improving on their weaknesses. This method was supplied by using Gaussian basis functions. In Chapter 6, three methods were discussed that used Gaussian functions of attribute distance: the probabilistic neural network (PNN), the generalized regression neural network (GRNN), and the radial basis function neural network (RBFN). The first two methods (PNN and GRNN) were derived using statistical theory, and the third method (RBFN) was derived using regularization theory. Both theoretically and by using a simple sine wave and square wave example, we saw that the GRNN can be seen as a subset of the RBFN method. I then applied both methods to the Blackfoot seismic data example. The results indicated that both methods gave about the same fit to the training data, but that the RBFN method tended to give higher-frequency results. However, the high frequencies did not appear to be from overtraining, as in the MLP approach, but appeared to bring out spatial continuity within the dataset.

The one problem with the full RBFN approach is that it requires the inversion of a symmetric $N \, x \, N$ matrix, where N is the number of training points. This can result in long processing times. The GRNN method is faster because training does not involve the inversion of a matrix. In Chapter 7, I looked for an approach that would vastly improve the run time of the full interpolation RBFN method, called RBFN with centres. This method was first applied to the AVO classification example, with results that were identical to the MLP approach. To apply the method to real data required the computation of a reduced set of data centres, which was done using *K*-means clustering and a refinement to *K*-means clustering which I termed "Mahalanobis" clustering. The resulting clustering method was applied to the Blackfoot seismic dataset and the results were very close in accuracy to the full interpolation RBFN method, but using only 2% of the number of values needed in the former approach.

Finally, in Chapter 8, I discussed the related problem of transforming map averages derived from a 3D volume into maps of reservoir parameters. I showed how to combine the multi-linear and neural network concepts discussed throughout the dissertation with traditional map-based geostatistics. This was done by creating an improved attribute map first and then using collocated cokriging to create a final map that honoured both the wells and the seismic volume. A summary of this work was presented in Table 8.4, which has been reproduced below as Table 9.1.

			Cokriged:		
X	Y	Kriged	Impedance	Multilinear	RBFN
80	29	-0.662	-1.338	-1.694	-1.842
55	17	-0.665	-1.367	-5.171	-4.213
96	27	-1.190	-0.372	-0.160	-1.043
110	17	5.040	4.287	2.801	3.042
63	28	-5.354	-5.835	-2.944	-3.518
65	40	3.318	4.258	2.847	3.054
96	40	-1.322	-1.753	-3.884	-3.489
26	41	1.032	-0.222	-1.610	-1.334
38	41	2.552	1.186	-0.333	0.456
47	39	-0.427	-0.593	-0.203	0.201
38	59	-6.652	-5.170	-2.066	-2.518
51	39	2.233	1.996	0.843	1.497
	RMS:	3.242	3.027	2.534	2.516

Table 9.1: A cross-validation analysis of the porosity error at each well from Figure 8.1 using each of the mapping methods discussed in Chapter 8. The units in the table are % porosity.

Table 9.1 represents a summary of this complete dissertation, since it contains all of the elements that have been presented. This includes the concept of the seismic attribute, discussed in Chapter 2, since we are using seismic impedance as our primary attribute. Also, I used multilinear regression and neural network (RBFN) to combine a group of map attributes to predict porosity as our reservoir parameter. I then used the geostatistical technique of collocated cokriging to "fine-tune" our prediction of porosity. Finally, the technique of cross-validation was used as an independent statistical evaluation of the accuracy of each method. The numbers at the bottom of each list of

cross-validated porosities is the RMS average of the porosity values. It can be seen in this example that the best "blind" prediction of porosity is from the RBFN method.

In addition, a number of secondary ideas were developed as a "by-product" of my primary research. These included the development of an approach to fluid estimation that combined the ideas of Biot, Gassmann, and AVO analysis (Russell et al., 2003), which was discussed in section 2.9; the development of a multilinear regression equation for the prediction of S-wave logs from a combination of P-wave sonic logs and gamma ray logs (Russell et al., 2004), which was discussed in section 3.7; and the development of Mahalanobis clustering and its application to AVO crossplotting (Russell et al., 2003), which was discussed in section 7.5.

9.3 Suggestions for future research

I think it is fair to say that no dissertation ever covers all of the research that the student envisaged covering when he or she started the project. The same is true for this work. In particular, I see the following three areas in which future work can be done:

(1) Optimization of the σ value in the RBFN method so that σ is a function of each attribute, as is done in the GRNN and PNN methods.
(2) Incorporation of the multivariate statistical parameters estimated using Mahalanobis clustering into the RBFN method with basis centres.
(3) Estimation of regularization parameters, which vary as a function of each weight.

Having said this, I feel that this dissertation advances research in the area of reservoir parameter prediction, and hope that the ideas presented here will be of use to my colleagues and to future geophysical students.

REFERENCES

Abdi, H., Valentin, D., and Edelman, B., 1999, Neural Networks: Sage Publications, Inc., Thousand Oaks, California.

Aki, K., and Richards, P.G., 2002, Quantitative Seismology, 2nd Edition: W.H. Freeman and Company.

Anderson, J.A., 1972, A simple neural network generating an interactive memory: Mathematical Biosciences, 14, 197-220.

Anderson, T.W., 1984, An Introduction to Multivariate Statistical Methods (2nd ed.): John Wiley, New York.

Bahorich, M. and Farmer, S., 1995, 3-D seismic discontinuity for faults and stratigraphic features: The coherence cube: THE LEADING EDGE, 14**,** no. 10, 1053-1058.

Batzle, M., and Wang, Z., 1992, Seismic properties of fluids: Geophysics, 57, 1396-1408.

Biot, M. A., 1941, General theory of three-dimensional consolidation: Journal of Applied Physics, 12, 155-164.

Bishop, C.M, 1995, Neural Networks for Pattern Recognition. Oxford: Oxford University Press.

Bortfeld, R., 1961, Approximations to the reflection and transmission coefficients of plane longitudinal and transverse waves: Geophys. Prosp., Eur. Assn. Geosci. Eng., 09**,** 485-502.

Bracewell, R.N., 1965, The Fourier Transform and its Applications: New York, McGraw-Hill Book Co., Inc.

Castagna, J. P., Batzle, M. L., and Eastwood, R. L., 1985, Relationships between compressional-wave and shear-wave velocities in clastic silicate rocks, Geophysics, **50**, 571-581.

Castagna, J. P., Batzle, M. L. and Kan, T. K., 1993, Rock physics - The link between rock properties and AVO response, *in* Backus, M. M., Ed., Offset-dependent reflectivity - theory and practice of AVO analysis: Soc. of Expl. Geophys., 135-171.

Chen, Q. and Sidney, S., 1997, Seismic attribute technology for reservoir forecasting and monitoring: THE LEADING EDGE, **16,** no. 05, 445-456.

Chen, S., Cowan, F.N., and Grant, P.M., 1991, Orthogonal least squares learning algorithm for radial basis function networks: IEEE Transactions on Neural Networks, 2, 302-309.

Claerbout, J, 1976, Fundamentals of Geophysical Data Processing, McGraw Hill, Inc.

Deutsch, C.V. and Journel, A.G., 1992, GSLIB: Geostatistical Software Library and User's Guide. Oxford University Press, New York, NY.

Doyen, P.M., 1988, Porosity from seismic data – A geostatistical approach: Geophysics, 53, no. 10, 1263-1275.

Duda, R.O., Hart, P.E., and Stork, D.G., 2001, Pattern Classification, Second Edition. New York, John Wiley and Sons.

Dufour, J., Squires, J., Goodway, W.N., Edmunds, A. and Shook, I., 2002, Integrated geological and geophysical interpretation case study, and Lamé rock parameter extractions using AVO analysis on the Blackfoot 3C-3D seismic data, southern Alberta, Canada: Geophysics, 67**,** 27-37.

Fatti, J., Smith, G., Vail, P., Strauss, P., and Levitt, P., 1994, Detection of gas in sandstone reservoirs using AVO analysis: a 3D Seismic Case History Using the Geostack Technique, Geophysics, 59, 1362-1376.

Fisher, R. A., 1936, The use of multiple measurements in taxonomic problems: Annals of Eugenics, 7, 179-188.

Fletcher, R., and Reeves, C.M., 1964, Function minimization by conjugate gradients: Computer Journal, 7, 149-154.

Gassmann, F., 1951, Uber die Elastizitat poroser Medien, Vierteljahrsschrift der Naturforschenden Gesellschaft in Zurich, 96, 1-23.

Gersztenkorn, A. and Marfurt, K. J., 1999, Eigenstructure-based coherence computations as an aid to 3-D structural and stratigraphic mapping: GEOPHYSICS, 64**,** 1468-1479.

Gill, P.E., Murray, W., and Wright, M.H., 1981, Practical Optimization: New York, Academic Press.

Goodway, W., Chen, T., and Downton, J., 1997, Improved AVO fluid Detection and Lithology Discrimination Using Lamé Petrophysical Parameters: Extended Abstracts, SEG 67th Annual International Meeting, Denver.

Hagan, M.T., Demuth, H.B., and Beale, M., 1996. Neural Network Design. Boston: PWS.

Hampson, D. and Russell, B., 1990, AVO inversion: Theory and practice: Annual Meeting Abstracts, Society of Exploration Geophysicists, 1456-1458.

Hampson, D. and Russell, B., 1992, Strata Course Notes: unpublished.

Hampson, D. and Russell, B., 1998, Emerge Course Notes: unpublished.

Hampson, D., Schuelke, J.S., and Quirein, J.A., 2001, Use of multi-attribute transforms to predict log properties from seismic data: Geophysics, 66, 220-231.

Haykin, S., 1999, Neural Networks, A Comprehensive Foundation, 2nd Ed. Englewood Cliffs, New Jersey: Prentice Hall.

Hebb, D., 1949, The Organization of Behaviour: New York, Wiley.

Hedlin, K., 2000, Pore Space Modulus and Extraction using AVO, Extended Abstracts, Soc. Expl. Geophys., 70th Annual Meeting, Calgary.

Hilterman, F. J., 2001, Seismic Amplitude Interpretation, 2001 Distinguished Instructor Short Course, Distinguished Instructor Series, No. 4, Soc. Expl. Geophys.

Hogg, R.V. and Craig, A.T., 1995, Introduction to Mathematical Statistics, 5th Edition. Englewood Cliffs, New Jersey: Prentice Hall.

Isaacs, E. and Srivastava, R., 1989, An Introduction to Applied Geostatistics. Oxford University Press, New York, NY.

Johnson, R.A. and Wichern, D.W., 1998. Applied Multivariate Statistical Analysis, Upper Saddle River, N.S.: Prentice Hall.

Kohonen, T., 1972, Correlation matrix memories: IEEE Transactions on Computers, volume C-21, 353-359.

Kohonen, T., 2001, Self Organizing Maps: Springer Series in Information Sciences, 20, Springer Verlag, Berlin.

Krief, M., Garat, J., Stellingwerff, J., and Ventre, J., 1990, A petrophysical interpretation using the velocities of P and S waves, The Log Analyst, Nov-Dec, 355-369.

Lindseth, R.O., 1979, Synthetic sonic logs-A process for stratigraphic interpretation: Geophysics, 44, 3-26.

Marfurt, K. J., Kirlin, R. L., Farmer, S. L. and Bahorich, M. S., 1998, 3-D seismic attributes using a semblance-based coherency algorithm: GEOPHYSICS, 63**,** 1150-1165.

Marfurt, K. J. and Kirlin, R. L., 2000, 3-D broad-band estimates of reflector dip and amplitude: GEOPHYSICS, 65**,** 304-320.

Masters, T., 1993, Practical Neural Network Recipes in C++: Academic Press, Inc.

Masters, T., 1995, Advanced algorithms for neural networks: John Wiley & Sons, Inc.

Mavko, G., Mukerji, T., and Dvorkin, J., 1998, The rock physics handbook – Tools for seismic analysis in porous media, Cambridge University Press.

McClelland, J.L. and Rumelhart, D. E., 1981, An interactive model of context effects in letter perception: part 1: An account of the basic findings, Psychological Review 88: 375-407.

McCulloch, W. S. and Pitts, W., 1943, A logical calculus of the ideas immanent in nervous activity: Bulletin of Mathematical Biophysics, **5**, 115-133.

Minsky, M.L., and Papert, S.A., 1969, Perceptrons: MIT Press, Cambridge, Mass.

Murphy, W., Reischer, A., and Hsu, K., 1993, Modulus Decomposition of Compressional and Shear Velocities in Sand Bodies, Geophysics, 58, 227-239.

Nadaraya, E.A., 1964, On estimating regression: Theory of Probability and its Applications 9 (1), 55-59.

Oldenburg, D. W., Scheuer, T. and Levy, S., 1983, Recovery of the acoustic impedance from reflection seismograms: Geophysics, 48**,** 1318-1337.

Orr, M., 1996, Introduction to radial basis function neural networks: Research Report for the Institute of Adaptive and Neural Computation, University of Edinburgh.

Partyka, G., Gridley, J. and Lopez, J., 1999, Interpretational applications of spectral decomposition in reservoir characterization: THE LEADING EDGE, 18**,** no. 3, 353-360.

Parzen, E. (1960), On estimation of a probability density function and mode, Annals of Mathematical Statistics, 33, 1065-1076.

Poggio, T., and Girosi, F., 1990, Networks for approximation and learning: Proceedings of the IEEE 78 (9), 1481-1497.

Powell, M.J.D., 1987, Radial basis functions for multivariable interpolation: a review. In J.C. Mason and M.G. Cox (Eds.), Algorithms for Approximation, 143-167. Oxford: Clarendon Press.

Press, W.H., Flannery, B., Teukolsky, S., and Vetterling, W., 1992, Numerical Recipes in C. Cambridge University Press, New York.

Richards, P. G. and Frasier, C. W., 1976, Scattering of elastic waves from depth-dependent inhomogeneities: Geophysics, 41**,** 441-458.

Ronen, S., Schultz, P.S., Hattori, M., and Corbett, C., 1994, Seismic-guided estimation of log properties, Parts 1, 2, and 3: The Leading Edge, 13, 305-310, 674-678, and 770-776.

Rosenblatt, M., 1958, The perceptron: A probabilistic model for information storage and organization in the brain: Psychological Review, 65, 386-408.

Ross, C. P., 2000, Effective AVO crossplot modeling: A tutorial: Geophysics, 65**,** 700-711.

Rummelhart, D.E., Hinton, G.E., and Williams, R.J., 1986, Learning representations of back-propagation errors: Nature, 323, 533-536.

Russell, B., and Lindseth, R.O., 1982, The information content of synthetic sonic logs – a frequency domain approach: abstracts of the 44th Mtg., EAEG.

Russell, B., 1988, Introduction to seismic inversion methods: Society of Exploration Geophysicists. (Course notes from SEG Continuing Education course)

Russell, B. and Hampson, D., 1991, A comparison of post-stack seismic inversion methods: Annual Meeting Abstracts, Society Of Exploration Geophysicists, 876-878.

Russell, B., Hampson, D., Schuelke, J. and Quirein, J., 1997, Multiattribute seismic analysis: The Leading Edge, 16, no. 10, 1439-1443.

Russell, B.H., Lines, L.R., and Ross, C.P., 2002a, AVO classification using neural networks: A comparison of two methods: CREWES Research Report 14.

Russell, B.H., Ross, C. P. and Lines, L.R., 2002b, Neural networks and AVO: THE LEADING EDGE, 21**,** no. 3, 268-277.

Russell, B., Hampson D.P., Lines L.R., and Todorov, T., 2002c, Combining geostatistics and multiattribute transforms: A channel sand case study: Journal of Petroleum Geology, 25, 97-117.

Russell, B., Hedlin, K., Hilterman, F. and Lines, L. R., 2003, Fluid-property discrimination with AVO: A Biot-Gassmann perspective: Geophysics, 68**,** 29-39.

Russell, B.H., Lines, L.R., and Hampson, D.P., 2003, Application of the radial basis function neural network to the prediction of log properties from seismic data: Exploration Geophysics, 34, 15-23.

Russell, B.H., and Lines, L.R., 2003, Mahalanobis clustering, with applications to AVO classification and seismic reservoir parameter estimation: CREWES Research Report 15.

Russell, B.H., Hampson, D.P, and Lines, L.R., 2004, A case study in the local estimation of shear-wave logs: accepted for presentation at the 74th Annual meeting of the Society of Exploration Geophysicists, Denver.

Rutherford, S. R. and Williams, R. H., 1989, Amplitude-versus-offset variations in gas sands: Geophysics, 54**,** 680-688.

Shuey, R.T., 1985, A simplification of the Zoeppritz equations: Geophysics 50, 609-614.

Smith, G.C., and Gidlow, P.M., 1987, Weighted stacking for rock property estimation and detection of gas: Geophys. Prosp., 35, 993-1014.

Specht, D.F., 1990, Probabilistic neural networks: Neural Networks 3 (1), 109-118.

Specht, D.F., 1991, A general regression neural network: IEEE Transactions on Neural Networks, 2(6), 568-576.

Strang, G., 1988, Linear Algebra and its Applications: Harcourt Brace and Company.

Taner, M.T., Koehler, F., and Sheriff, R.E., 1979, Complex seismic trace analysis: Geophysics, **44**, 1041-1063.

Tikhonov, A. N., and Arsenin, V.Y., 1977, Solutions of Ill-posed Problems: V.H. Winston, Washington, D.C.

Todorov, T., Yang, G., and Stewart, R., 1997, Geostatistical analysis of 3C-3D seismic data for depth, isopachs, and sand/shale distribution: 1997 Technical Abstract Book, Canadian SEG, 52.

Todorov, T. I., Stewart, R. R., Hampson, D. P. and Russell, B. H., 1998, Well log prediction using attributes from 3C-3D seismic data: Annual Meeting Abstracts, Society Of Exploration Geophysicists: 1574-1576.

Todorov, T. I., 2000, Integration of 3C-3D seismic data and well logs for rock property estimation: M.Sc. Dissertation, University of Calgary.

Treitel, S., 1970, Principles of digital multichannel filtering: Geophysics, 35**,** 785-811.

Walden, A. T., 1991, Making AVO sections more robust: Geophysical Prospecting, 39**,** 915-942.

Watson, G.S., 1964, Smooth regression analysis: Sankhya, the Indian Journal of Statistics. Series A 26, 359-372.

Widrow, B., and Stearns, S., 1985, Adaptive Signal Processing. New York: Prentice-Hall.

Wiggins, R., Kenney, G.S., and McClure, C.D., 1983, A method for determining and displaying the shear-velocity reflectivities of a geological formation: European Patent Application 0113944.

Zoeppritz, K.,1919, Erdbebenwellen VIIIB, On the reflection and propagation of seismic waves: Gottinger Nachrichten, I, 66-84.

APPENDIX 1: A note on terminology

This appendix has been added to clarify the terminology used in the main body of this dissertation.

Fundamentally, there are four different datasets to consider: the input values, the weighting coefficients, the output values, and the target values. For a single input set of values, the transform relationship is given by

$$y = w_0 + \boldsymbol{w}^T \boldsymbol{x}\,, \tag{A1-1}$$

where y is the output value, a scalar, w_0 is a bias term, $\boldsymbol{w}^T = [w_1 \quad w_2 \quad \cdots \quad w_M]$ is a weight vector of M values, and $\boldsymbol{x}^T = [x_1 \quad x_2 \quad \cdots \quad x_M]$ is an input vector also of M values. The superscript ^{T} indicates the transpose operation, in which a column vector becomes a row vector. The bias term can be absorbed into the weight and input vectors by replacing equation (A1-1) by

$$y = \boldsymbol{w}^T \boldsymbol{x}\,, \tag{A1-2}$$

where $\boldsymbol{w}^T = [w_0 \quad w_1 \quad \cdots \quad w_M]$ and $\boldsymbol{x}^T = [1 \quad x_1 \quad \cdots \quad x_M]$. Equation (A1-2) is often referred to as the dot, or scalar, product.

In the more general case where there are N input vectors, we can write

$$y_j = \boldsymbol{w}^T \boldsymbol{x}_j\,, \quad j = 1,\ldots,N\,, \tag{A1-3}$$

where $\boldsymbol{x}_j^T = [1 \quad x_{1j} \quad \cdots \quad x_{Mj}]$. Equation (A1-3) can be replaced by the matrix equation

$$\boldsymbol{y}^T = \boldsymbol{w}^T X\,, \tag{A1-4}$$

where $\boldsymbol{y}^T = [y_1 \quad y_2 \quad \cdots \quad y_N]$ and $X = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_N] = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_{11} & x_{12} & \ldots & x_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \ldots & x_{MN} \end{bmatrix}$.

By taking the transpose of both sides of equation A1-4, we get the equation

$$\boldsymbol{y} = X^T \boldsymbol{w}, \tag{A1-5}$$

which we will also write as

$$\boldsymbol{y} = A\boldsymbol{w}, \tag{A1-6}$$

where

$$A = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_M \end{bmatrix} = \begin{bmatrix} 1 & a_{11} & \dots & a_{1M} \\ 1 & a_{21} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{N1} & \dots & a_{NM} \end{bmatrix} = X^T = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \dots & x_{M1} \\ 1 & x_{12} & \dots & x_{M2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1N} & \dots & x_{MN} \end{bmatrix}, \text{ and}$$

$$\boldsymbol{a}_i = \begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{Ni} \end{bmatrix}, i = 1,\dots,M .$$

The distinction between vectors $\boldsymbol{x_j}$ and $\boldsymbol{a_i}$ is important. The $\boldsymbol{a_i}$ vectors represent M input seismic attribute traces, each with N samples, whereas the $\boldsymbol{x_j}$ vectors represent N input vectors of seismic attributes, each with M samples. The elements of matrix A can be written as a_{ji}, and the elements of matrix X can be written as x_{ij}, again emphasizing that the two matrices are a transposed pair. We can also rewrite equation A1-6 as

$$\boldsymbol{y} = w_0\boldsymbol{a}_0 + w_1\boldsymbol{a}_1 + \ldots + w_M\boldsymbol{a}_M, \tag{A1-7}$$

where the weights now represent a linear regression on the M attribute vectors. The vector $\boldsymbol{y}$ is the output after applying the weight vector to the input. To determine the weights, we often use a training vector $\boldsymbol{t}$, which can be physically interpreted as the well log associated with the input attributes. The vectors $\boldsymbol{y}$ and $\boldsymbol{t}$ are of the same length, and the transpose of $\boldsymbol{t}$ can be written:

$$\boldsymbol{t}^T = \begin{bmatrix} t_1 & t_2 & \cdots & t_N \end{bmatrix}. \tag{A1-8}$$

In the still more general case where we have K weight vectors, equation (A1-5) becomes the matrix equation

$$Y = W^T X, \tag{A1-9}$$

where X is as given above, W^T is the K x M dimensional weight matrix given by

$$W^T = \begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_K^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{M1} \\ w_{12} & w_{22} & \cdots & w_{M2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1K} & w_{2K} & \cdots & w_{MK} \end{bmatrix},$$

and Y is the K x N dimensional output matrix given by

$$Y = \begin{bmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_N \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1N} \\ y_{21} & y_{22} & \cdots & y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{K1} & y_{K2} & \cdots & y_{KN} \end{bmatrix}.$$

Thus, we have transformed the M x N dimensional matrix X into the K x N dimensional matrix Y, using the K x M dimensional transform W^T. Note that the values in X can be written as x_{ij}, where subscript i refers to the attribute number, and goes from *1* to *M,* and subscript j refers to the sample in the time series, and goes from *1* to *N*. The values in W^T can be written w_{ki}, where subscript k represents the order of the weights (for example, the weights for the k^{th} node in a multi-layer perceptron), and goes from *1* to *K,* and subscript i again represents the attribute number, and goes from *1* to *M*. The values of Y can be written y_{kj}, where subscript k refers to the output from a particular set of weights (e.g. from a given node in a single layer perceptron), and goes from *1* to *K*, and subscript j represents the output from a particular sample, and goes from *1* to *N*.

Until now, all of our indices have been in subscript form. The last case that we will consider involves an index in superscript form. This is the index that keeps track of the layer number (that is, the level of nesting) in a multilayer perceptron. Since superscripts can be confused with exponential powers, I have chosen to bracket this index. Thus, we will write:

$$w^{(q)},\ q = 1, 2, \ldots, Q \tag{A1-10}$$

for the weight matrix of the q^{th} layer, where there are Q layers in total. The complete input-output relationship for a two layer *MLP* network could therefore be written as:

$$z^{(2)} = f^{(2)}\{W^{(2)T}[f^{(1)}(W^{(1)T}X)]\} \tag{A1-11}$$

Notice that equation (A1-11) does not include a bias term. This can been included in the weight and input matrices by appending a zeroth term to each of vectors in the matrices of equation A1-5, or $\boldsymbol{x}_j^T = \left(x_{0j}, x_{1j}, \ldots, x_{Mj}\right)$, and $\boldsymbol{w}_k^T = \left(w_{0k}, w_{1k}, \ldots, w_{Mk}\right)$, where $x_{0j} = 1$ and w_{0k} = the bias term.

APPENDIX 2: The least-squares method

A2.1 A geometrical development of the least-squares method

One of the basic problems discussed in this dissertation has been the prediction an output training vector, $\boldsymbol{t}$, from M input vectors, $\boldsymbol{a_i}$, where both the input and training vectors are N-dimensional. In equation form, we can write:

$$\boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} = w_1\boldsymbol{a_1} + w_2\boldsymbol{a_2} + \ldots + w_M\boldsymbol{a_M} = w_1\begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{1N} \end{bmatrix} + w_2\begin{bmatrix} a_{21} \\ a_{22} \\ \vdots \\ a_{2N} \end{bmatrix} + \ldots + w_M\begin{bmatrix} a_{M1} \\ a_{M2} \\ \vdots \\ a_{MN} \end{bmatrix}. \qquad \text{(A2.1)}$$

This is an inverse problem in which we want to find the weights which best reconstruct the training vector. Once the weights have been found the actual computed output will be the vector $\boldsymbol{y}$, which is usually not identical to the target vector. Equation (A2.1) can be written more compactly as

$$\boldsymbol{t} = A\boldsymbol{w}, \qquad \text{(A2.2)}$$

where $A = \begin{bmatrix} a_{11} & \ldots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{N1} & \ldots & a_{NM} \end{bmatrix}$. If $M = N$ and the input vectors are independent we can find a solution using the matrix inverse

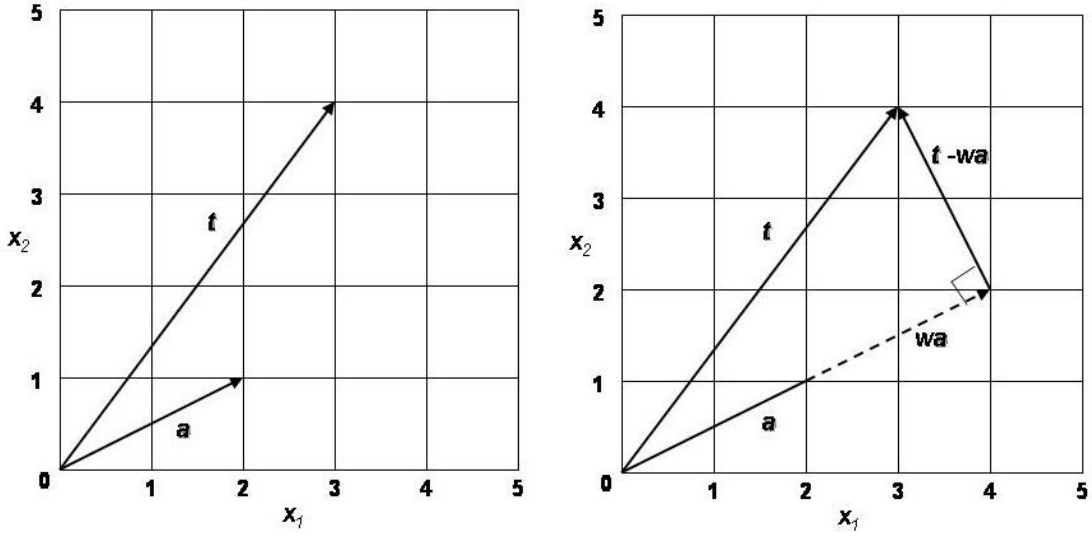
$$\boldsymbol{w} = A^{-1}\boldsymbol{t}. \qquad \text{(A2.2)}$$

However, we typically find that $M < N$, and an exact solution cannot be found. This problem is called overdetermined, since we have more information than we need. As the simplest example of this, we will let $M = 1$ and $N = 2$, and we can rewrite equation (A2.1) as

$$\boldsymbol{t} = w\boldsymbol{a}, \qquad \text{(A2.3)}$$

where $\boldsymbol{t} = [t_1 \quad t_2]^T$, and $\boldsymbol{a} = [a_1 \quad a_2]^T$. We can find an exact solution only if $\boldsymbol{a}$ is linearly dependent on $\boldsymbol{t}$. In the more general case, shown in Figure 2.1(a), we can never match $\boldsymbol{t}$ exactly by applying a weight to $\boldsymbol{a}$, and the optimum solution is to scale $\boldsymbol{a}$ so that it comes as close as possible to $\boldsymbol{t}$. This is shown in Figure 2.1(b), where the vector that joins the product $w\boldsymbol{x}$ to $\boldsymbol{t}$ is orthogonal to $\boldsymbol{t}$. Since this distance is given by $\boldsymbol{t} - w\boldsymbol{a}$, and we know that the scalar product between two orthogonal vectors is equal to zero, we can write

$$(\boldsymbol{t} - w\boldsymbol{a}) \cdot \boldsymbol{a} = 0 \Rightarrow \boldsymbol{t} \cdot \boldsymbol{a} = w(\boldsymbol{a} \cdot \boldsymbol{a}) \Rightarrow w = \frac{\boldsymbol{t} \cdot \boldsymbol{a}}{\boldsymbol{a} \cdot \boldsymbol{a}} = \frac{t_1 a_1 + t_2 a_2}{a_1^2 + a_2^2} \tag{A2.4}$$



(a) (b)

Figure A2.1: A geometrical illustration of least-squares, where (a) shows the input and target vectors $\boldsymbol{a}$ and $\boldsymbol{t}$, and (b) shows that $w\boldsymbol{a}$ is the closest scaled version of $\boldsymbol{a}$ to $\boldsymbol{t}$.

To demonstrate using an example, notice that for $\boldsymbol{a} = (2, 1)^T$ and $\boldsymbol{t} = (3, 4)^T$, as shown in Figure A2.1(a), the weight would be

$$w = \frac{3 \cdot 2 + 4 \cdot 1}{2^2 + 1^2} = \frac{10}{5} = 2 .$$

The output $\boldsymbol{y}$ thus can be written

$$\boldsymbol{y} = w\boldsymbol{a} = 2\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix},$$

which is geometrically as close as we can get to the right answer.

A2.2 Alternate derivations of the least-squares method

A second way to derive the identical result is to minimize the error between the vectors $\boldsymbol{t}$ and $w\boldsymbol{a}$. If we define E as the squared distance between $\boldsymbol{t}$ and $w\boldsymbol{a}$, we can write

$$E = \left|w\boldsymbol{a} - \boldsymbol{t}\right|^2 = \left(wa_1 - t_1\right)^2 + \left(wa_2 - t_2\right)^2 . \tag{A2.5}$$

Setting the derivative of the error with respect to the weight equal to zero gives

$$\frac{dE}{dw} = 2\left(wa_1 - t_1\right)a_1 + 2\left(wa_2 - t_2\right)a_2 = 0 . \tag{A2.6}$$

Simplifying, we get $w\left(a_1^2 + a_2^2\right) = a_1 t_1 + a_2 t_2 \Rightarrow w = \dfrac{a_1 t_1 + a_2 t_2}{a_1^2 + a_2^2}$. This result is identical to the result that we obtained using a geometrical argument. However, this approach explains why the method is also called the *least-squares technique*.

There is yet a third way of deriving the result we have just derived using geometry and calculus. If we start with the vector equation given in equation (A2.3) and simply pre-multiply both sides by the vector transpose of the input vector $\boldsymbol{a}$, we get

$$\boldsymbol{a}^T\boldsymbol{t} = w(\boldsymbol{a}^T\boldsymbol{a}) \Rightarrow w = \frac{\boldsymbol{a}^T\boldsymbol{t}}{\boldsymbol{a}^T\boldsymbol{x}} = \frac{a_1 t_1 + a_2 t_2}{a_1^2 + a_2^2} \tag{A2.7}$$

Let us now move to higher dimensions. If we let N increase, while leaving $M = 1$, the solution is the same as just given, finding a single weight which minimizes the distance from one vector in N-dimensional space to another vector in N-dimensional space. The next level of difficulty is therefore to let $N = 3$ and $M = 2$. In this case, we need to find the solution to the problem

$$\boldsymbol{t} = w_1\boldsymbol{a_1} + w_2\boldsymbol{a_2}, \tag{A2.8}$$

where we now have the vectors $\boldsymbol{t} = \begin{bmatrix} t_1 & t_2 & t_3 \end{bmatrix}^T$, $\boldsymbol{a_1} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix}^T$, and $\boldsymbol{a_2} = \begin{bmatrix} a_{21} & a_{22} & a_{23} \end{bmatrix}^T$. Geometrically, we now want to find the weighted vector sum of the inputs which gives the shortest distance to the training vector. This is given as the orthogonal projection of $\boldsymbol{t}$ onto the vector $w_1\boldsymbol{a_1} + w_2\boldsymbol{a_2}$. Following Bishop (1995), if we

call this orthogonal projection $\boldsymbol{t}_{orth}$, then we can decompose $\boldsymbol{t}$ into the two components $\boldsymbol{t} = \boldsymbol{t}_{orth} + \boldsymbol{t}_{para}$, where $\boldsymbol{t}_{para}$ is the parallel component of $\boldsymbol{t}$. Notice that the least-squares error can now be written

$$E = \left| \sum_{j=1}^{M} w_j \boldsymbol{a}_j - \boldsymbol{t} \right|^2 \tag{A2.9}$$

and we can therefore arrive at a least-squares solution by setting the derivative of E with respect to each of the weights equal to zero, or:

$$\frac{\partial E}{\partial w_j} = \boldsymbol{a}_j^T (\boldsymbol{y} - \boldsymbol{t}) = 0 \tag{A2.10}$$

Substituting, we find that

$$\boldsymbol{a}_j^T (\boldsymbol{y} - (\boldsymbol{t}_{orth} + \boldsymbol{t}_{para})) = 0 \tag{A2.11}$$

But, since $\boldsymbol{a}_j^T \boldsymbol{t}_{orth} = 0$, we find that $\boldsymbol{y} - \boldsymbol{t}_{para} = 0$, or $\boldsymbol{y} = \boldsymbol{t}_{para}$.

The matrix solution to the full least-squares problem can be found by generalizing our geometrical argument. That is, expanding equation (A2.7) using the M vectors shown in equation (A2.1), we can write the following M linear equations by taking the dot product of each of the M input vectors $\boldsymbol{a}_i$ with the target vector $\boldsymbol{t}$, or

$$\begin{aligned}
\boldsymbol{a}_1 \cdot \boldsymbol{t} &= w_1(\boldsymbol{a}_1 \cdot \boldsymbol{a}_1) + w_2(\boldsymbol{a}_1 \cdot \boldsymbol{a}_2) + \ldots + w_M(\boldsymbol{a}_1 \cdot \boldsymbol{a}_M), \\
\boldsymbol{a}_2 \cdot \boldsymbol{t} &= w_1(\boldsymbol{a}_2 \cdot \boldsymbol{a}_1) + w_2(\boldsymbol{a}_2 \cdot \boldsymbol{a}_2) + \ldots + w_M(\boldsymbol{a}_2 \cdot \boldsymbol{a}_M) \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
\boldsymbol{a}_M \cdot \boldsymbol{t} &= w_1(\boldsymbol{a}_M \cdot \boldsymbol{a}_1) + w_2(\boldsymbol{a}_M \cdot \boldsymbol{a}_2) + \ldots + w_M(\boldsymbol{a}_M \cdot \boldsymbol{a}_M)
\end{aligned} \tag{A2.12}$$

Note that equation (A2.12) can be written in the following matrix form:

$$\begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_M^T \end{bmatrix} \boldsymbol{t} = \left\{ \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_M^T \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_1 \end{bmatrix} \right\} \boldsymbol{w}, \tag{A2.13}$$

Or more compactly as

$$A^T \boldsymbol{t} = (A^T A) \boldsymbol{w} \tag{A2.14}$$

where A is as defined in equation (A2.2)

.

Note that the matrix product A^TA is now a square matrix, so we simply perform the matrix equivalent of division by multiplying by the inverse of A^TA. This leads to the pseudo-inverse solution

$$\boldsymbol{w} = (A^T A)^{-1} A^T \boldsymbol{t}, \tag{A2.15}$$

where the product A^TA now gives us a square matrix which is invertible if the input vectors are all independent. (The non-invertible case will be discussed in section A2.4).

A2.3 Adding a zero weight to the least-squares problem

Let us next consider the effect of adding a zero weight, w_0, to an underdetermined set of equations. As an example, consider again the case of $N=2$ and $M=1$. The two linear equations can now be written:

$$\begin{aligned} t_1 &= w_0 + w_1 a_1 \\ t_2 &= w_0 + w_1 a_2 \end{aligned} \tag{A2.16}$$

or

$$\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 1 & a_1 \\ 1 & a_2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \Rightarrow \boldsymbol{t} = A\boldsymbol{w}, \tag{A2.17}$$

with solution

$$\boldsymbol{w} = A^{-1}\boldsymbol{t} \Rightarrow \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 1 & a_1 \\ 1 & a_2 \end{bmatrix}^{-1} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \Rightarrow \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \frac{1}{a_2 - a_1} \begin{bmatrix} a_2 & -a_1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \dfrac{t_1 a_2 - t_2 a_1}{a_2 - a_1} \\ \dfrac{t_2 - t_1}{a_2 - a_1} \end{bmatrix}$$

When we substitute our last example ($\boldsymbol{x} = (2, 1)^T$ and $\boldsymbol{t} = (3, 4)^T$), note that this gives $w_0 = 5$ and $w_1 = -1$, which leads to

$$\boldsymbol{y} = w_0 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_1 \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

By adding a zero weight we have thud solved the problem exactly, without needing a second set of measurements. However, this approach will fail if $\boldsymbol{a}$ is itself a scaled unit vector, that is, if $a_1 = a_2$. In this case, the determinant of the matrix is undefined, and the inverse does not exist. In general, as the dimensions of the problem increase, but $M = N\text{-}1$, we can solve the problem exactly by adding a zero weight, and thus a unit vector, as long as none of the input vectors is equal to a scaled unit vector. If $M < N\text{-}1$, the problem is improved by adding a zero weight but not solved exactly, again assuming that none of the input vectors are equal to a scaled unit vector. A more practical solution is to add pre-whitening to the problem.

A2.4 The effect of pre-whitening

In the case of an undefined matrix inverse caused by two or more linearly dependent input vectors, a practical solution is to add pre-whitening. This can be written

$$\boldsymbol{w} = (A^T A + \lambda I)^{-1} A^T \boldsymbol{t}, \tag{A2.18}$$

where λ is the pre-whitening factor and I is the M x M identity matrix.

As an example, if our earlier problem was re-defined so that we were trying to predict the target vector $\boldsymbol{t} = (3, 4)^T$ using the input vector $\boldsymbol{a} = (1, 1)^T$, then the matrix solution using a zero weight would be

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 4 \end{bmatrix},$$

for which the matrix inverse is undefined. By adding the pre-whitening term $\lambda \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix, we get

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \left\{ \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right\}^{-1} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1+\lambda & 1 \\ 1 & 1+\lambda \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

Using the explicit form of the matrix solution, we get:

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \frac{1}{(1+\lambda)^2 - 1} \begin{bmatrix} 1+\lambda & -1 \\ -1 & 1+\lambda \end{bmatrix} \begin{bmatrix} 3 \\ 4 \end{bmatrix},$$

which is obviously invertible as long as $\lambda > 0$. Let us now try a few values of λ. If $\lambda = 0.1$ (10% pre-whitening), we get:

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \frac{1}{0.21}\begin{bmatrix} 1.1 & -1 \\ -1 & 1.1 \end{bmatrix}\begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} -3.333 \\ 6.667 \end{bmatrix} \Rightarrow \mathbf{y} = -3.333 + 6.667\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.333 \\ 3.333 \end{bmatrix}.$$

If $\lambda = 0.01$ (1% pre-whitening), we get:

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \frac{1}{0.02}\begin{bmatrix} 1.01 & -1 \\ -1 & 1.01 \end{bmatrix}\begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} -48.26 \\ 51.74 \end{bmatrix} \Rightarrow \mathbf{y} = -48.26 + 51.74\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.483 \\ 3.483 \end{bmatrix}$$

Recall that the least-squares solution to the problem is given in equation (A2.11), which gives in this case

$$w = \frac{\mathbf{a}^T\mathbf{t}}{\mathbf{a}^T\mathbf{a}} = \frac{a_1t_1 + a_1t_1}{a_1^2 + a_2^2} = \frac{3+4}{2} = 3.5 \Rightarrow y = w\mathbf{a} = 3.5\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 3.5 \end{bmatrix}.$$

Comparing the last three computations, it is obvious that the pre-whitening solution with the linear dependent vectors is converging to the least-squares solution. To see this explicitly, let us re-write the matrix solution as

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \frac{1}{(1+\lambda)^2 - 1}\begin{bmatrix} 1+\lambda & -1 \\ -1 & 1+\lambda \end{bmatrix}\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \frac{1}{\lambda^2 + 2\lambda}\begin{bmatrix} t_1 - t_2 + \lambda t_1 \\ t_2 - t_1 + \lambda t_2 \end{bmatrix} = \begin{bmatrix} \frac{t_1 - t_2}{\lambda^2 + 2\lambda} + \frac{t_1}{\lambda + 2} \\ \frac{t_2 - t_1}{\lambda^2 + 2\lambda} + \frac{t_2}{\lambda + 2} \end{bmatrix}$$

Notice that in this case each output value y_i is equal to the sum of the two weights, which gives

$$y_1 = y_2 = \frac{t_1 + t_2}{\lambda + 2}.$$

Therefore, we can see from the above derivation that, as λ approaches 0, y_1 and y_2 approach $(t_1 + t_2)/2$.

APPENDIX 3: Digital Multichannel Filtering

In this appendix, we will discuss the theory of the multichannel digital filter (Treitel, 1970), which can be seen as the generalization of the concepts of the linear perceptron model and multilinear regression with convolutional weights. Let us first re-write the general form of the multichannel digital filter from Treitel (1970), using the notation described in Appendix 1:

$$\begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_K \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_{11} & \boldsymbol{w}_{12} & \cdots & \boldsymbol{w}_{1M} \\ \boldsymbol{w}_{21} & \boldsymbol{w}_{22} & \cdots & \boldsymbol{w}_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{K1} & \boldsymbol{w}_{K2} & \cdots & \boldsymbol{w}_{KM} \end{bmatrix} * \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_M \end{bmatrix}, \tag{A3.1}$$

where $\boldsymbol{a}_i = \begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{Ni} \end{bmatrix}$, $\boldsymbol{w}_{ki} = \begin{bmatrix} w_{ki}(1) \\ w_{ki}(2) \\ \vdots \\ w_{ki}(L) \end{bmatrix}$, *and* $\boldsymbol{y}_k = \begin{bmatrix} y_{1k} \\ y_{2k} \\ \vdots \\ y_{(N+L-1)k} \end{bmatrix}$, and * denotes convolution.

In this notation, $\boldsymbol{a_i}$ is an input attribute of N samples, $\boldsymbol{w_{ki}}$ is set of filter weights of length L, and $\boldsymbol{y_k}$ is an output reservoir parameter of length $N+L-1$. When K and M are both equal to one, equation (A3.1) reduces to the classical single-channel filter, which can be written without subscripts as $\boldsymbol{y = w * a}$. If we let both the filter and input have two points, the complete transient convolutional matrix can be written as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} w(1) & 0 \\ w(2) & w(1) \\ 0 & w(2) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} w(1)a_1 \\ w(2)a_1 + w(1)a_2 \\ w(2)a_2 \end{bmatrix} \tag{A3.2}$$

This can be written as the two-point ($N = L = 2$), two input/output ($K = M = 2$) case considered by Treitel (1970) as

$$\begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_{21} & \boldsymbol{w}_{12} \\ \boldsymbol{w}_{21} & \boldsymbol{w}_{22} \end{bmatrix} * \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \end{bmatrix}, \tag{A3.3}$$

which can be expanded to give

$$\boldsymbol{y}_1 = \boldsymbol{w}_{11} * \boldsymbol{a}_1 + \boldsymbol{w}_{12} * \boldsymbol{a}_2,$$
$$\boldsymbol{y}_2 = \boldsymbol{w}_{21} * \boldsymbol{a}_1 + \boldsymbol{w}_{22} * \boldsymbol{a}_2,$$

where

$$\boldsymbol{a}_i = \begin{bmatrix} a_{1i} \\ a_{2i} \end{bmatrix}, \quad \boldsymbol{w}_{ki} = \begin{bmatrix} w_{ki}(1) \\ w_{ki}(2) \end{bmatrix}, \quad \text{and } \boldsymbol{y}_k(t) = \begin{bmatrix} y_{1k} \\ y_{2k} \\ y_{3k} \end{bmatrix}.$$

We can then write this equation in a more complete way as

$$\boldsymbol{y} = W\boldsymbol{a} \Rightarrow \begin{bmatrix} y_{11} \\ y_{21} \\ y_{31} \\ y_{12} \\ y_{22} \\ y_{32} \end{bmatrix} = \begin{bmatrix} w_{11}(1) & 0 & w_{12}(1) & 0 \\ w_{11}(2) & w_{11}(1) & w_{12}(2) & w_{12}(1) \\ 0 & w_{11}(2) & 0 & w_{11}(2) \\ w_{21}(1) & 0 & w_{22}(2) & 0 \\ w_{21}(2) & w_{21}(1) & w_{22}(2) & w_{22}(1) \\ 0 & w_{21}(2) & 0 & w_{22}(2) \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \end{bmatrix}. \tag{A3.4}$$

Equation (A3.4) can obviously be generalized for any number of filters and inputs, of any length. Substituting the numerical example from Treitel (1970), we find that

$$\begin{bmatrix} y_1(0) \\ y_1(1) \\ y_1(2) \\ y_2(0) \\ y_2(1) \\ y_2(2) \end{bmatrix} = \begin{bmatrix} 6 & 0 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 7 & 0 & 1 & 0 \\ 1 & 7 & 3 & 1 \\ 0 & 1 & 0 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ -5 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 12 \\ -20 \\ 1 \\ 18 \\ -18 \\ 4 \end{bmatrix}. \tag{A3.5}$$

For the inverse problem, we can write the generalized inverse solution as $\boldsymbol{a} = (W^T W)^{-1} W^T \boldsymbol{y}$, or $\boldsymbol{a} = R^{-1}\boldsymbol{c}$, where R is the autocorrelation matrix and $\boldsymbol{c}$ is the cross-correlation between the output and the filter. This gives us:

$$\begin{bmatrix} a_1(0) \\ a_1(1) \\ a_2(0) \\ a_2(1) \end{bmatrix} = \begin{bmatrix} 87 & 13 & 12 & 1 \\ 13 & 87 & 33 & 12 \\ 12 & 33 & 14 & 3 \\ 1 & 12 & 3 & 14 \end{bmatrix}^{-1} \begin{bmatrix} 160 \\ -241 \\ -76 \\ -4 \end{bmatrix} = \begin{bmatrix} 2 \\ -5 \\ 4 \\ 3 \end{bmatrix}. \tag{A3.6}$$

As expected, the inverse gives us the exact answer in this case. The time domain solution shown above is not given in Treitel (1970), since that paper focuses on the Z-transform solution to the multichannel filter.

Next, consider the case of a single-point filter, which is the case used in neural network theory. We can now drop the time index on the filter, to get

$$\begin{bmatrix} y_1(0) \\ y_1(1) \\ y_1(0) \\ y_1(1) \end{bmatrix} = \begin{bmatrix} w_{11} & 0 & w_{12} & 0 \\ 0 & w_{11} & 0 & f_{12} \\ w_{21} & 0 & w_{22} & 0 \\ 0 & w_{21} & 0 & w_{22} \end{bmatrix} \begin{bmatrix} a_1(0) \\ a_1(1) \\ a_2(0) \\ a_2(1) \end{bmatrix}. \tag{A3.7}$$

Re-writing equation (A3.7) in the same form as equation (A3.3), we get

$$\begin{aligned} \boldsymbol{y}_1(t) &= w_{11} I\, \boldsymbol{a}_1(t) + w_{12} I\, \boldsymbol{a}_2(t), \text{ and} \\ \boldsymbol{y}_2(t) &= w_{21} I\, \boldsymbol{a}_1(t) + w_{22} I\, \boldsymbol{a}_2(t), \end{aligned} \tag{A3.8}$$

where I is the identity matrix, $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Since $I\, \boldsymbol{a}_i(t) = \boldsymbol{a}_i(t)$, we can rewrite equation (A3.8) as

$$\begin{aligned} \boldsymbol{y}_1(t) &= w_{11}\, \boldsymbol{a}_1(t) + w_{12} \boldsymbol{a}_2(t), \\ \boldsymbol{y}_2(t) &= w_{21} \boldsymbol{a}_1(t) + w_{22} \boldsymbol{a}_2(t), \end{aligned} \tag{A3.9}$$

which can be re-arranged in matrix form as

$$\begin{bmatrix} \boldsymbol{y}_1^T(t) \\ \boldsymbol{y}_2^T(t) \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_1^T(t) \\ \boldsymbol{a}_2^T(t) \end{bmatrix}, \tag{A3.10}$$

where the superscript ^{T} indicates the vector transpose.

Equation (A3.10) can be expanded to give

$$\begin{bmatrix} y_1(0) & y_1(1) \\ y_2(0) & y_2(1) \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} a_1(0) & a_1(1) \\ a_2(0) & a_2(1) \end{bmatrix}. \tag{A3.11}$$

We can therefore re-write the single point filter equivalent of the general case shown in equation (A3.1) as

$$\begin{bmatrix} \mathbf{y}_1^T(t) \\ \mathbf{y}_2^T(t) \\ \vdots \\ \mathbf{y}_K^T(t) \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1M} \\ w_{21} & w_{22} & \cdots & w_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KM} \end{bmatrix} \begin{bmatrix} \mathbf{a}_1^T(t) \\ \mathbf{a}_2^T(t) \\ \vdots \\ \mathbf{a}_M^T(t) \end{bmatrix}. \tag{A3.12}$$

Equation A3.12 is the standard way of writing the first layer calculation for a multi-layer perceptron with M inputs and K perceptrons. This is obviously a special case of the multichannel digital filter in which the filters have only a single point, or weight. However, because of the transpose operation in equation (A3.12), there is a fundamental difference in the interpretation of the two approaches which becomes more obvious when we expand equation (A3.12) in the same way that we expanded equation (A3.10). That is, we get:

$$\begin{bmatrix} y_1(0) & y_1(1) & \cdots & y_1(n-1) \\ y_2(0) & y_2(1) & \cdots & y_2(n-1) \\ \vdots & \vdots & \ddots & \vdots \\ y_K(0) & y_K(1) & \cdots & y_K(n-1) \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1M} \\ w_{21} & w_{22} & \cdots & w_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KM} \end{bmatrix} \begin{bmatrix} a_1(0) & a_1(1) & \cdots & a_1(n-1) \\ a_2(0) & a_2(1) & \cdots & a_1(n-1) \\ \vdots & \vdots & \ddots & \vdots \\ a_M(0) & a_M(1) & \cdots & a_M(n-1) \end{bmatrix}. \tag{A3.13}$$

Thus, each value in the output matrix can be thought of as the product of a horizontal weight vector and a constant time vector. Equation (A3.13) can therefore be written:

$$\begin{bmatrix} \mathbf{y}(0) & \mathbf{y}(1) & \cdots & \mathbf{y}(n-1) \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_K^T \end{bmatrix} \begin{bmatrix} \mathbf{a}(0) & \mathbf{a}(1) & \cdots & \mathbf{a}(n-1) \end{bmatrix}, \tag{A3.14}$$

where:

$$\boldsymbol{y}(j)=\begin{bmatrix} y_1(j) \\ y_2(j) \\ \vdots \\ y_K(j) \end{bmatrix}, \boldsymbol{w}_k=\begin{bmatrix} w_{k1} \\ w_{k2} \\ \vdots \\ w_{kM} \end{bmatrix}, \text{and } \boldsymbol{a}(j)=\begin{bmatrix} a_1(j) \\ a_2(j) \\ \vdots \\ a_M(j) \end{bmatrix}.$$

Note that I have used a bold letter to denote the weight vector, or $\boldsymbol{w}_k$. This new weight vector should not be confused with the multi-point convolutional filter defined in equation (A3.1). In that case, the filter was convolved with the full input time series at each channel. In this case, we take the dot product of the weight vector with a vector containing a single sample of all the input channels, which is a multiplexed operation.

As a further simplification, if we let $K = 1$ (a single perceptron), equation (A3.14) reduces to

$$\boldsymbol{y}^T(t)=\begin{bmatrix} w_1, & w_2, & \cdots, & w_M \end{bmatrix}\begin{bmatrix} \boldsymbol{a}_1^T(t) \\ \boldsymbol{a}_2^T(t) \\ \vdots \\ \boldsymbol{a}_M^T(t) \end{bmatrix}. \tag{A3.15}$$

In equation (A3.15) the k subscript has been dropped and now the weight values are not written in bold, since these are scalar values rather than vectors. Equation (A3.15) can also be written in transposed form as

$$\begin{aligned} \boldsymbol{y}(t) &= \begin{bmatrix} \boldsymbol{a}_1(t), & \boldsymbol{a}_2(t), & \cdots, & \boldsymbol{a}_M(t) \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}. \\ &= w_1\boldsymbol{a}_1(t)+w_2\boldsymbol{a}_2(t)+\ldots+w_M\boldsymbol{a}_2(t) \end{aligned} \tag{A3.16}$$

Equation (A3.16) can be seen to be the common form of the multilinear regression equation.

The multilinear regression equation in equations (A3.14) through (A3.16) is the same approach we use in Chapter 3 to initially find the optimum set of attributes to use in the neural network approach. Again, this is simply a special case of the multichannel digital filter.

Finally, consider the case of multilinear regression with convolutional weights, which is also discussed in Chapter 3. This again is a special case of equation (A3.1), where $K = 1$. That is:

$$y(t) = \begin{bmatrix} w_1(t) & w_2(t) & \cdots & w_M(t) \end{bmatrix} * \begin{bmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_M(t) \end{bmatrix}. \tag{A3.18}$$

APPENDIX 4: Bayes' Theorem

A4.1 Introduction

One of the cornerstones of statistical theory is Bayes' Theorem, first derived by the Reverend Thomas Bayes in 1764. Bayes' Theorem can be written succinctly as

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}, \tag{A4.1}$$

where: $P(A)$ = the unconditional probability of event A, $P(B)$ = the probability of event B $P(A \mid B)$ = the conditional probability of A given B, and $P(B \mid A)$ = the conditional probability of B given A. Although equation A4.1 is a correct statement of Bayes' Theorem, it is extremely abstract for those not familiar with statistics. In the next section, I will explain Bayes' Theorem with a simple geologic example. In the last section, I will consider an example using probability distributions.

A4.2 A simple example of Bayes' Theorem

Let us consider a geological example in which we have determined the lithology and porosity of samples taken from ten different reservoirs. We will assume that there are only two possible lithologies, sandstone (SS) and limestone (LS), and two possible porosities, low porosity (LP) and high porosity (HP). A summary of the ten samples is shown in Table A4.1, where we note that there are one low porosity sandstone sample, four high porosity sandstone samples, three low porosity limestone samples, and two high porosity limestone samples. Note that the table also contains the sums of the columns and rows, and shows that there are an equal number of sandstone and limestone samples, but more high porosity than low porosity cores. To relate this table back to the form of Bayes' Theorem given in equation A4.1, we will refer to the porosity and lithology as *events*, and have arbitrarily called the lithology event A and the porosity event B.

Lithology (A)	Porosity (B)		
	LP	*HP*	*Sum*
SS	1	4	5
LS	3	2	5
Sum	4	6	10

Table A4.1: A table of ten cores samples as a function of porosity and lithology, where SS = Sandstone, LS = Limestone, LP = Low porosity, and HP = High porosity.

Table A4.1 contains three fundamentally different types of probabilities: the joint probability, the unconditional probability, and the conditional probability. The *joint probability* is the probability of two events happening simultaneously, the *unconditional probability* is the probability that a single event happens, and the conditional probability is the probability that an event happens *given* that we know that a second event has already happened.

Since probabilities are given as fractions, we must normalize the frequencies of occurrence that are given in Table A4.1. Dividing the values in Table A4.1 by the total number of well samples (10), we find the joint probability, written *P(A,B)*. The joint probabilities are thus given as

P(SS, LP) = 1/10, P(LS, LP) = 3/10, P(SS, HP) = 2/5, and P(LS, HP) = 1/5.

Notice that the sum of the four joint probabilities is equal to one, since one of these occurrences must happen. A summary of the joint probabilities is given in Table A4.2.

Lithology (A)	Porosity (B)		
	LP	*HP*	*Sum =P(A)*
SS	1/10	2/5	1/2
LS	3/10	1/5	1/2
Sum =P(B)	2/5	3/5	1.0

Table A4.2: The joint probabilities *P(A,B)* and unconditional probabilities *P(A)* and *P(B)* of the events in Table A4.1, where *P(A,B) = P(B,A).*

The unconditional probabilities can be found by adding the values in the rows of Table A4.1 for lithology, or the values in the columns in Table A4.1 for porosity, and again dividing by 10. For the lithology, we get *P(SS) = 1/2* and *P(LS) = 1/2*. For the porosity, we get *P(LP) = 2/5* and *P(HP) = 3/5*. Again, the sum of *P(SS)* and *P(LS)* and the sum of *P(LP)* and *P(HP)* are both equal to 1.0. It can also be seen that the unconditional probabilities are the sum of the joint probabilities for a given row or column in Table A4.2.

To compute the conditional probability, written *P(B|A)*, we divide the individual numbers in Table A4.1 by either the row sums (to find *P(A|B)*), or the column sums (to find *P(B|A)*). For example, to compute *P(SS|HP)*, we divide the number of high porosity sandstones by the total number of high porosities, to get 2/3. To compute *P(HP|SS)* we divide the number of high porosity sandstones by the total number of sandstones, to get 4/5. Note that *P(B|A)* is not equal to *P(A|B)*. Table A4.3 shows the conditional probabilities *P(A|B)* and Table A4.4 shows the conditional probabilities *P(B|A)*.

Porosity (B)

Lithology (A)	*LP*	*HP*	*Sum*
SS	3/4	2/3	*N/A*
LS	1/4	1/3	*N/A*
Sum	1.0	1.0	*N/A*

Table A4.3: The conditional probabilities *P(A|B)* of the events in Table A4.1.

Porosity (B)

Lithology (A)	*LP*	*HP*	*Sum*
SS	1/5	4/5	1.0
LS	3/5	2/5	1.0
Sum	*N/A*	*N/A*	*N/A*

Table A4.4: The conditional probabilities *P(B|A)* of the events in Table 4.1.

Let us now look at the relationship between joint probability and conditional probability. The ratio of the joint probability of two events *A* and *B* to the probability of one of those events, let's say *A*, is equal to the conditional probability of *B* given *A*. This can be written

$$P(B \mid A) = \frac{P(A, B)}{P(A)} \tag{A4.2}$$

Alternately, for the conditional probability of *A* given *B,* we can write

$$P(A \mid B) = \frac{P(A, B)}{P(B)} \tag{A4.3}$$

Equations (A4.2) and (A4.3) can be verified by observing the computed values in the previous tables. For example, if we look at the high porosity sandstone cores we see that

$$P(SS \mid HP) = \frac{P(SS, HP)}{P(HP)} = \frac{2/5}{3/5} = \frac{2}{3}, \tag{A4.4}$$

or

$$P(HP \mid SS) = \frac{P(HP, SS)}{P(SS)} = \frac{2/5}{1/2} = \frac{4}{5}\ . \tag{A4.5}$$

By rearranging equation (A4.2) and (A4.3) we arrive at the law of multiplication of probabilities, which can be written as

$$P(A, B) = P(B|A)P(A) \tag{A4.6}$$

or

$$P(A, B) = P(A|B)P(B) \tag{A4.7}$$

Since both of the above equations are equal, we can equate their right hand sides to get:

$$P(B \mid A)P(A) = P(A \mid B)P(B) \tag{A4.8}$$

This leads to Bayes' Theorem of equation (A4.1), by simply dividing through by either *P(A)* or *P(B)*. That is we get either

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}, \tag{A4.9}$$

or

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}, \tag{A4.10}$$

To verify this using our example, note that by re-arranging equations (A4.4) and (A4.5) we get:

$$P(SS,HP) = P(SS|HP)P(HP)=2/3*3/5=2/5$$

and $$P(HP,SS) = P(HP|SS)P(SS)=4/5*1/2=2/5$$

or

$$P(SS|HP)\ P(HP) = P(HP|SS)\ P(SS) \tag{A4.11}$$

Dividing both sides in equation (A4.11) by either *P(HP)* or *P(SS)* leads to Bayes' Theorem. For example, if we divide by *P(HP)*, we get:

$$P(SS \mid HP) = \frac{P(HP \mid SS)P(SS)}{P(HP)} = \frac{2/5}{3/5} = \frac{2}{3}, \tag{A4.12}$$

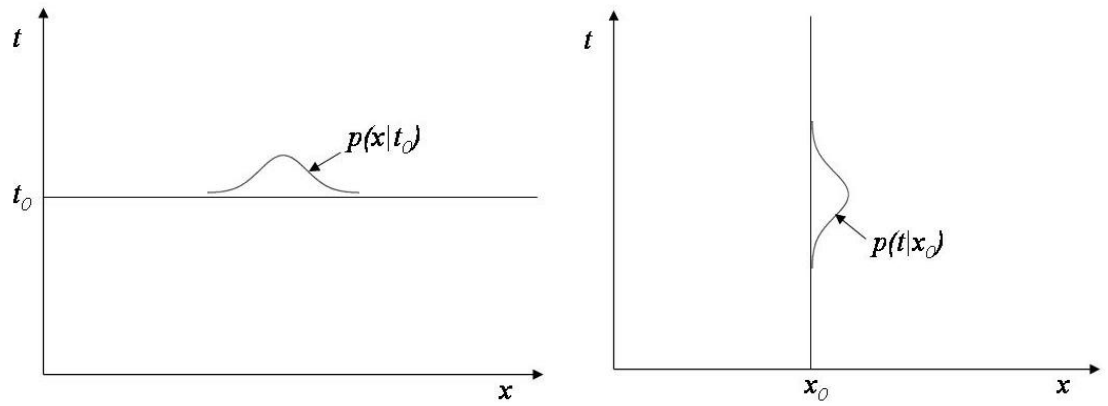
a if we divided by *P(SS)*, we get:

$$P(HP \mid SS) = \frac{P(SS \mid HP)P(HP)}{P(SS)} = \frac{2/5}{1/2} = \frac{4}{5}, \tag{A4.12}$$

A4.3 Bayes' theorem with probability functions

In the previous example, we have assumed that events A and B are discrete events. The restriction that each event only had two outcomes can easily be expanded for any number of events (for example, five different lithologies and a range of porosities from 5% to 25%, in increments of 1%). But the real power of Bayes' Theorem comes into play when we assume continuous probability distributions for the two events. This is

shown graphically in Figure A4.1, where we have assumed conditional distributions of the form $p(x|t)$ and $p(t|x)$ where x represents our input values and t represents our target values.



(a) (b)

Figure A4.1 An interpretation of (a) the conditional probability function $p(x|t)$ at the fixed value t_0, and (b) the conditional probability function $p(t|x)$ at the fixed value x_0.

Notice that the probability function drawn in Figure A4.1 is a Gaussian distribution. Let us assume that the joint probability can be written as

$$p(x,t) = c \cdot exp\left(-\frac{(x-\mu_x)^2}{\sigma_x^2} - \frac{(t-\mu_t)^2}{\sigma_t^2} \right), \tag{A4.9}$$

and the unconditional probabilities written as

$$p(x) = c \cdot exp\left(-\frac{(x-\mu_x)^2}{\sigma_x^2} \right), \tag{A4.10}$$

and

$$p(t) = c \cdot exp\left(-\frac{(t-\mu_t)^2}{\sigma_t^2} \right), \tag{A4.11}$$

The conditional probabilities can then be written as

$$p(x \mid t) = \frac{p(x,t)}{p(t)} = \frac{c \cdot exp\left(-\frac{(x-\mu_x)^2}{\sigma_x^2} - \frac{(t-\mu_t)^2}{\sigma_t^2} \right)}{c \cdot exp\left(-\frac{(t-\mu_t)^2}{\sigma_t^2} \right)}, \tag{A4.12}$$

and

$$p(t \mid x) = \frac{p(x,t)}{p(x)} = \frac{c \cdot exp\left(-\frac{(x-\mu_x)^2}{\sigma_x^2} - \frac{(t-\mu_t)^2}{\sigma_t^2} \right)}{c \cdot exp\left(-\frac{(x-\mu_x)^2}{\sigma_x^2} \right)}, \tag{A4.13}$$

APPENDIX 5: Associative Networks

A5.1 Introduction

Although there are a bewildering array of neural networks, most of these networks are variants on two simple neural network models: the perceptron (McCulloch and Pitts, 1943, Rosenblatt, 1958) and Hebbian learning (Hebb, 1949). The single-layer perceptron lead to the development of the multi-layer perceptron, and Hebbian learning lead to the development of the associative network and the Kohonen self-organizing map. In this appendix, we will discuss the ideas that evolved from Hebbian learning, starting with Hebb's postulate. Donald Hebb was a Canadian psychologist working at Harvard who introduced the following postulate in his 1949 book "The Organization of Behaviour".

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic charge takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

In the preceding quote, cells A and B are biological neurons, and an axon is the transmission line that connects these neurons. Translating Hebb's postulate to mathematical language will lead to autoassociative and heteroassociative learning, as well as the least mean square (LMS) algorithm, singular value decomposition (SVD), and the generalized inverse. These concepts will be illustrated with a straightforward geophysical example.

A5.2 Autoassociative learning

To translate Hebb's postulate to a mathematical model, we will first assume that our cells (A, B, etc) are the *M*-dimensional attribute vectors $\boldsymbol{x}_j$, which can be written (see Appendix 1) as

$$\boldsymbol{x}_j = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{Mj} \end{bmatrix}, \; j = 1,2,\ldots,N .$$

We will next create an operator *W* by summing the outer products of the input vectors. This can be written as

$$W = \sum_{j=1}^{N} \boldsymbol{x}_j \boldsymbol{x}_j^T = XX^T , \tag{A5.1}$$

where *X* is an *M x N* dimensional matrix consisting of the input vectors as its columns (see Appendix 1), and *W* is symmetric and of dimension *M x M*. To understand how *W* is related to Hebb's postulate, consider applying it to our input vectors. This can be written

$$\boldsymbol{y}_j = W \boldsymbol{x}_j , \tag{A5.2}$$

where the vector $\boldsymbol{y}_j$ is the output, of length *M*, the same length as the input $\boldsymbol{x}_j$. (Note that in Appendix 1, equation A5.2 was written using W^T. However, since *W* is symmetrical, $W^T = W$.)

Since *W* was created from the outer products of the inputs, we can think of it as having a "memory" of the inputs, so that the input vectors try to "find" themselves in *W*. Hebb's rule would tell us that the input vectors, or cells, are reinforced by finding a correlated version of themselves within *W*. We can therefore rewrite equation A5.2 as:

$$\hat{\boldsymbol{x}}_j = W \boldsymbol{x}_j , \tag{A5.3}$$

where $\hat{\boldsymbol{x}}_j$ is a reconstructed version of $\boldsymbol{x}_j$. If $\hat{\boldsymbol{x}}_j = \boldsymbol{x}_j$, the memory has perfect recall ($\boldsymbol{x}_j$ has found itself perfectly), and if $\hat{\boldsymbol{x}}_j \neq \boldsymbol{x}_j$, the memory has imperfect recall ($\boldsymbol{x}_j$ has found itself imperfectly). This is referred to as autoassociative learning.

To understand this mathematically, note that by combining equations A5.1 and A5.3, we get

$$\hat{\boldsymbol{x}}_k = W\boldsymbol{x}_k = \sum_j \boldsymbol{x}_j \boldsymbol{x}_j^T \boldsymbol{x}_k = \sum_j \left(\boldsymbol{x}_j^T \boldsymbol{x}_k\right)\boldsymbol{x}_j = \left(\boldsymbol{x}_k^T \boldsymbol{x}_k\right)\boldsymbol{x}_k + \sum_{j \neq k} \left(\boldsymbol{x}_j^T \boldsymbol{x}_k\right)\boldsymbol{x}_j . \tag{A5.4}$$

The result is the sum of two parts. The first part is simply a scaled version of $\boldsymbol{x}_k$. The second part consists of cross-talk between the input vector and the stored vectors (Abdi et al, 1999). If the input vector is orthogonal to all of the training vectors, the result will be zero and we will get perfect recall to within a scale factor. (If the vectors are orthonormal, the scale factor will be equal to one.) If the vectors are not orthogonal, we will get imperfect recall. The amount of error can be quantified by finding the cosine between the input and output vectors, or

$$cos\left(\hat{\boldsymbol{x}}_j, \boldsymbol{x}_j\right) = \frac{\hat{\boldsymbol{x}}_j^T \boldsymbol{x}_j}{\left|\hat{\boldsymbol{x}}_j\right|\left|\boldsymbol{x}_j\right|} . \tag{A5.5}$$

Imperfect recall can have two separate causes. Either the input vector was not used in the computation of *W*, or the vectors used in the training were not orthogonal.

A5.3 Autoassociative Learning Example

Now, let's look at a numerical example of autoassociative learning. Recall that our input consists of a series of vectors containing *M* seismic attribute values at a particular sample. Although the seismic attributes can take on any real value within a range determined by the number of bits in the recording system, in this example we will allow only values *+1* or *-1*. This is equivalent to the sign-bit recording system. If we have two seismic attributes or reservoir parameters, there are thus four possible cases:

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

This can be thought of as the four extreme cases in an AVO A/B crossplot, where $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ are the top and base of a wet sand, and $\boldsymbol{x}_1$ and $\boldsymbol{x}_4$ are the top and base of a gas sand. For the case of $M = 3$, we have $2^3 = 8$ cases, or

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \boldsymbol{x}_5 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \boldsymbol{x}_6 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \boldsymbol{x}_7 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \boldsymbol{x}_8 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}.$$

For $N = 4$, there are $2^4 = 16$ distinct cases, for $N = 5$, there are $2^5 = 32$ distinct cases, and so on. Thus, for N values, the complete input will always consist of 2^N vectors.

Next, consider the matrix X and its transpose X^T, which contain the full set of input vectors. For $N = 2$, we get:

$$X = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \end{bmatrix} \quad and \quad X^T = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix} = [\boldsymbol{a}_1, \boldsymbol{a}_2],$$

where $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are the seismic attribute traces as a function of time (see Appendix 1), given by

$$\boldsymbol{a}_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \text{ and } \boldsymbol{a}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}.$$

Notice that $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are orthogonal, since they each contain two 1's and two -1's. Thus, from equation A5.1, we find

$$W = XX^T = \begin{bmatrix} \boldsymbol{a}_1^T\boldsymbol{a}_1 & \boldsymbol{a}_1^T\boldsymbol{a}_2 \\ \boldsymbol{a}_2^T\boldsymbol{a}_1 & \boldsymbol{a}_2^T\boldsymbol{a}_2 \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} = 4\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

For $N = 3$, we find that

$$XX^T = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 8 \end{bmatrix}.$$

In both cases, W is equal to 2^N times the N x N identity matrix. We can produce similar results for N of any size. The scaling factor of N can be removed by normalizing the vectors, which involves dividing by $\sqrt{N}$.

For the above case in which a complete orthogonal set of vectors is used to create X, the resulting weight matrix will always produce perfect recall. If we reduce the number of input vectors in the training, the result will depend on whether the remaining vectors are orthogonal or not. For example, if we return to the $N = 2$ case, and use vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in the training, which are orthogonal, we get:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$

and there is perfect recall. However, if we use vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_4$ in the training, which are not orthogonal, the result is less than perfect. That is:

$$W = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

$$\Rightarrow \hat{X} = WX = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -4 \\ 4 & 0 & 0 & -4 \end{bmatrix}.$$

In this case, although the vectors used in the training have been recalled perfectly to within a scale factor, the others have been set to zero.

For a more realistic example, consider the case of $N = 3$. We will design the weight matrix using only the first two vectors, $\boldsymbol{x}_1^T = [1, 1, 1]$ and $\boldsymbol{x}_2^T = [1, 1, -1]$, and then apply the weights to the complete set of inputs. This gives us:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix},$$

$$\text{and} \quad \hat{X} = WX = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 4 & 0 & 0 & 0 & 0 & -4 & -4 \\ 4 & 4 & 0 & 0 & 0 & 0 & -4 & -4 \\ 2 & -2 & 2 & 2 & -2 & -2 & 2 & -2 \end{bmatrix}.$$

This result tells us that we are able to recall vectors 1 and 2, the training vectors, but with an error given by

$$cos(\hat{\boldsymbol{x}}_1, \boldsymbol{x}_1) = cos(\hat{\boldsymbol{x}}_2, \boldsymbol{x}_2) = \frac{40}{48} = 0.833 .$$

It also tells us that we did a very poor job of recalling vectors 3 through 6. Their error was:

$$cos(\hat{\boldsymbol{x}}_j, \boldsymbol{x}_j) = \frac{8}{48} = 0.167, \quad j = 3,4,5,6 .$$

Notice that vectors 7 and 8 were recovered with the same error as vectors 1 and 2. This is because these vectors are reverse polarity versions of vectors 1 and 2, and filter operations are insensitive to a polarity reversal. To understand why vectors 1 and 2 were imperfectly reconstructed in the above case, we can use equation A5.5 to find that

$$\hat{\boldsymbol{x}}_1 = (\boldsymbol{x}_1^T \boldsymbol{x}_1)\boldsymbol{x}_1 + (\boldsymbol{x}_2^T \boldsymbol{x}_1)\boldsymbol{x}_2$$

$$\Rightarrow \hat{\boldsymbol{x}}_1 = 3\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + 1\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 2 \end{bmatrix}.$$

Note the cross-talk error due to the fact that $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are not orthogonal.

A5.4 The LMS algorithm

As we have just seen, the Hebb rule does not work perfectly if the learned vectors are not orthogonal. However, the method can be improved by iteratively adjusting the weights based on the observed error. This method goes by a number of names: the delta rule, the perceptron learning rule, the least mean square (LMS) algorithm, or the Widrow-

Hoff learning rule (Widrow and Hoff, 1960), named after the inventers of the LMS algorithm. This algorithm is also identical to the gradient descent method and, in the limit, to the Moore-Penrose generalized inverse. The LMS algorithm can be derived by modifying equation (A5.1). Note that this equation can be rewritten iteratively as

$$W(j) = W(j-1) + \alpha \boldsymbol{x}_j \boldsymbol{x}_j^T, \quad j = 0, 1, \ldots, N, \tag{A5.6}$$

where we have added the term α, which is a constant (positive and less than 1) called the learning rate, and assumed that *W(0)* is the zero matrix:

$$W(0) = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}.$$

Next, we replace $\boldsymbol{x}_j$ in the second term on the right hand side of that equation by the error between the input and the output. The error can be written as $\boldsymbol{x}_j - \hat{\boldsymbol{x}}_j$, or $\boldsymbol{x}_j - W(n-1)\boldsymbol{x}_j$ giving

$$W(n) = W(n-1) + \alpha(\boldsymbol{x}_j - W(n-1)\boldsymbol{x}_j)\boldsymbol{x}_j^T. \tag{A5.7}$$

Thus, we iteratively update the weight matrix by using the error between the correct output and the output of the previous iteration. Our criterion for stopping will be when the error approaches a small enough value. We can also remove the dependency on the *j* term by using all of the input vectors at the same time, or

$$W(n) = W(n-1) + \alpha(X - W(n-1)X)X^T \tag{A5.8}$$

Equation A5.8 can also be written as

$$W(n) = W(n-1) + \alpha E X^T, \tag{A5.9}$$

where *E* is the error term, or

$$W(n) = W(n-1) + \Delta W(n-1), \tag{A5.10}$$

where $\Delta W(n-1)$ is the correction matrix for the *(n-1)st* iteration.

Let us now go back to our previous example and apply the LMS algorithm. We will let $\alpha = 0.25$ (choosing an optimum value of α will be discussed later), and we will initialize *W* to the 3 x 3 zero matrix.

Since $X = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}$, we find that $\hat{X}(0) = W(0)X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$, and $E(0) = X - X(0) = X$.

Thus we, after the first iteration, get:

$$W(1) = W(0) + 0.25E(0)X^T = 0.25XX^T = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}.$$

The weight matrix after the first iteration is simply the matrix given in section A5.3 times the α factor. For the second iteration, the new output is

$$X(1) = W(1)X = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0.5 & -0.5 \end{bmatrix}.$$

Note that this is again simply our original result scaled by α. Now compute the error:

$$E(1) = X - \hat{X}(1) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.5 & -0.5 \end{bmatrix}.$$

Notice that the error is largest where the misfit in the output was greatest, in the third row. Next, we compute the delta matrix:

$$\Delta W(1) = 0.25E(1)X^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}.$$

The correction is only non-zero in one element in the bottom row of the delta matrix. Thus for the second iteration of the weight matrix we get

$$W(2) = W(1) + \Delta W(1) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 0.75 \end{bmatrix}.$$

In the updated matrix, the value in the bottom row is now larger. This gives us the new result:

$$\hat{X}(2) = W(2)X = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0.75 & -0.75 \end{bmatrix}.$$

Although the result is still not perfect, we are going in the right direction. All we need to do now is iterate through the process. After 20 iterations we find that

$$W(20) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \hat{X}(20) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

A5.5 Eigendecomposition and singular value decomposition (SVD)

An alternative approach to the implementation of the LMS algorithm can be found using singular value decomposition (SVD). To understand this approach, let us briefly review eigenvalues and eigenvectors. If $\boldsymbol{v}$ is an eigenvector of the square matrix W, and

$$W\boldsymbol{v} = \lambda \boldsymbol{v} \tag{A5.11}$$

then λ is the eigenvalue associated with eigenvector $\boldsymbol{v}$. To illustrate this, let us take the inner product matrix of our previous example, or

$$A = X^T X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}.$$

Then:

$$A\boldsymbol{v} = \lambda \boldsymbol{v} \Rightarrow |A - \lambda \boldsymbol{v}| = 0 \Rightarrow \begin{vmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{vmatrix} = 0$$

$$\Rightarrow (3-\lambda)^2 - 1 = 0 \Rightarrow \lambda^2 - 6\lambda + 8 = 0 \Rightarrow (\lambda - 4)(\lambda - 2) = 0.$$

Thus, matrix A has eigenvalues of $\lambda_1 = 4$ and $\lambda_2 = 2$, and is said to be of full rank equal to 2 (Strang, 1988). The eigenvectors are

$$A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 \Rightarrow \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \text{ and}$$

$$A\boldsymbol{v}_2 = \lambda_2 \boldsymbol{v}_2 \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ have been normalized.

Let us now compute its outer product of X and find the eigenvalues and eigenvectors. The outer product has already been computed and is given by

$$B = XX^T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

The eigenvalues for B are computed by

$$\begin{vmatrix} 2-\lambda & 2 & 0 \\ 2 & 2-\lambda & 0 \\ 0 & 0 & 2-\lambda \end{vmatrix} = 0 \Rightarrow (2-\lambda)^3 - 4(2-\lambda) = 0$$

$$\Rightarrow \lambda^3 - 6\lambda^2 + 8\lambda = 0 \Rightarrow \lambda(\lambda^2 - 6\lambda^2 + 8\lambda) = 0.$$

which gives $\lambda_1 = 4$, $\lambda_2 = 2$, and $\lambda_3 = 0$. Since the third eigenvalues is equal to zero, matrix B is not of full rank (which would be 3) and has a rank of 2, the same as matrix A. The eigenvectors are computed as

$$(B - \lambda_1 I)\boldsymbol{u}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} -2 & 2 & 0 \\ 2 & -2 & 0 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

$$(B - \lambda_2 I)\boldsymbol{u}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{21} \\ u_{22} \\ u_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} u_{21} \\ u_{22} \\ u_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \text{ and}$$

$$(B - \lambda_3 I)\boldsymbol{u}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}.$$

For the zero eigenvalue, notice that the matrix can be written in the form:

$$M\boldsymbol{u} = 0,$$

and the vector ***u*** is said to be in the null space of *M*.

Once we have computed the eigenvalues and eigenvectors of a matrix, we can express it as

$$W = U\Lambda U^{-1}, \tag{A5.12}$$

where U is a matrix whose columns are the eigenvectors and Λ is a diagonal matrix whose main diagonal consists of the eigenvalues of W. In the symmetrical case we have been considering, note that $U^{-1} = U^T$. For our two cases:

$$A = V\Lambda_A V^T = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}4 & 2\\ 4 & -2\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = \begin{bmatrix}3 & 1\\ 1 & 3\end{bmatrix},$$

and

$$B = U\Lambda_B U^T = \begin{bmatrix}\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}}\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}4 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0\\ 0 & 0 & 1\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0\end{bmatrix}$$

$$= \begin{bmatrix}\frac{4}{\sqrt{2}} & 0 & 0\\ \frac{4}{\sqrt{2}} & 0 & 0\\ 0 & 2 & 0\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0\\ 0 & 0 & 1\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0\end{bmatrix} = \begin{bmatrix}2 & 2 & 0\\ 2 & 2 & 0\\ 0 & 0 & 2\end{bmatrix}.$$

Eigenvalues and eigenvectors can only be computed from square matrices. In the more general case, we wish to eigendecompose a rectangular matrix. This is especially important in seismic data analysis since we are almost always dealing with rectangular matrices (i.e. more equations than unknowns). This is referred to as singular value decomposition (SVD) and will not be derived here (see Strang, 1988, Appendix A). The result is given by:

$$X = U\Delta V^T, \tag{A5.13}$$

where X is any rectangular matrix, U is the eigenvector matrix of the XX^T, Δ is a diagonal matrix with the square roots of the eigenvalues of XX^T on its diagonal (these are called singular values) and V is the eigenvector matrix of X^TX. Since U and V are of a different size, we can either append zeros to V or drop the zero eigenvalues and its associated

eigenvector from Δ and U. Obviously, it is simpler and more efficient to do the latter. Using our example from the last section, we get:

$$X = U\Delta V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 \\ \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

A5.6 The LMS algorithm and SVD

As shown by Widrow and Stearns (1985) the SVD leads to a very fast way of solving the LMS algorithm. Recall that the LMS algorithm is written

$$W(n) = W(n-1) + \alpha(X - W(n-1)X)X^T. \tag{A5.14}$$

This can be written as the eigendecomposition:

$$W(n) = U\Phi(n)U^T, \tag{A5.15}$$

where $\Phi(n) = I - (I - \alpha\Lambda)^n$.

From equation (A5.21), we note two important points. First, since $\Phi(n)$ is only stable if (Hagan et al., 1995)

$$|1 - \alpha\lambda_i| < 1,$$

then the maximum stable learning rate parameter must satisfy:

$$1 - \alpha\lambda_{max} < -1$$

or $\alpha < \frac{2}{\lambda_{max}}$.

(In the case we have been considering, $\alpha < \frac{2}{4} = \frac{1}{2}$, thus 0.25 was a reasonable choice for α). Second, we can write

$$(I - \alpha\Lambda)^n = \begin{bmatrix} f_l^n & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & f_m^n \end{bmatrix},$$

where $f_i < 1$.

Then, for $n = \infty$, this matrix becomes the zero matrix, and we find that

$$W(\infty) = UU^T.$$

The effect of the LMS algorithm is thus to whiten the weight matrix by setting each of its eigenvalues to 1. To illustrate this using our previous example, note that:

$$W = UU^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow W = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

This was the same answer we got using the recursive LMS algorithm, but took a lot less work.

A5.6 Heteroassociative learning

In autoassociative learning, we try to detect patterns without supervision. However, the type of problem we are most interested in this thesis is the supervised problem, in which we are presented with at least some examples of what our output should look like. Specifically, this may be the measured porosity in a well that is intersected by a seismic line. This is called heteroassociative learning, and is closely related to the autoassociative case that we considered in the previous section. The basic equations for heteroassociative learning are almost identical to equations (A5.1) and (A5.2), except that the output vector $\boldsymbol{y}$ is replaced with the training vector $\boldsymbol{t}$. That is:

$$\boldsymbol{t}_j = \begin{bmatrix} t_{1j} \\ t_{2j} \\ \vdots \\ t_{Mj} \end{bmatrix} = W \boldsymbol{x}_j. \qquad \text{(A5.16)}$$

Equation (A5.15) is for the single input (or stimulus) case. As in autoassociative learning, we can also consider the batch learning case, in which we process N input vectors simultaneously. In this case, we write:

$$T = W X, \tag{A5.17}$$

where $T = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_N] = \begin{bmatrix} t_{11} & \cdots & t_{1N} \\ \vdots & \ddots & \vdots \\ t_{K1} & \cdots & t_{KN} \end{bmatrix}$, and $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N] = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{MN} \end{bmatrix}$.

The key question is again how to determine the weight matrix W. Heteroassociative learning does this using by summing all possible outer products between the $\boldsymbol{x}_j$ and $\boldsymbol{t}_j$ vectors, as in the autoassociative case. That is:

$$W = \sum_{j=1}^{N} \boldsymbol{x}_j \boldsymbol{t}_j^T = X T^T . \tag{A5.18}$$

The accuracy of the prediction of the training vectors can be computed by comparing the estimated values:

$$\hat{T} = W^T X, \tag{A5.19}$$

with the true values, using the cosine rule given earlier in equation (A5.6). For the training vector case, we get:

$$cos(\hat{\boldsymbol{t}}, \boldsymbol{t}) = \frac{\hat{\boldsymbol{t}} \boldsymbol{t}^T}{|\hat{\boldsymbol{t}}||\boldsymbol{t}|} . \tag{A5.20}$$

We will again find that the accuracy of the prediction will depend on the orthogonality of the input vectors. That is:

$$\hat{\boldsymbol{t}}_k = W^T \boldsymbol{x}_k = \sum_j \boldsymbol{t}_j \boldsymbol{x}_j^T \boldsymbol{x}_k = \sum_j (\boldsymbol{x}_j^T \boldsymbol{x}_k) \boldsymbol{t}_j = (\boldsymbol{x}_k^T \boldsymbol{x}_k) \boldsymbol{t}_k + \sum_{j \neq k} (\boldsymbol{x}_j^T \boldsymbol{x}_k) \boldsymbol{t}_j . \tag{A5.21}$$

If the $\boldsymbol{x}_j$ vectors are orthogonal, the first term will equal a scaled version of $\boldsymbol{t}_k$ (equal to $\boldsymbol{t}_k$ for orthonormal inputs) and the second term will be zero. For non-orthogonal input vectors, the second term will introduce cross-talk.

A5.7 Heterassociative example

In this example we will use the same input vectors as in our previous example, or:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix},$$

but will now try to predict the two-dimensional output

$$T = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Thus, we find:

$$W = XT^T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 2 & 0 \\ 0 & 2 \end{bmatrix},$$

$$\text{and} \quad \hat{T} = W^T X = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 2 & -2 \end{bmatrix}.$$

The error in computing both $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ is therefore:

$$\cos(\hat{\boldsymbol{t}}_1, \boldsymbol{t}_1) = \cos(\hat{\boldsymbol{t}}_2, \boldsymbol{t}_2) = \frac{6}{\sqrt{20} \cdot \sqrt{2}}$$

If we compute the estimate for the first vector using equation (A5.21), we get:

$$\hat{\boldsymbol{t}}_1 = (\boldsymbol{x}_1^T \boldsymbol{x}_1)\boldsymbol{t}_1 + (\boldsymbol{x}_2^T \boldsymbol{x}_1)\boldsymbol{t}_1 = \left\{ \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \left\{ \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 3 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}.$$

As in our earlier autoassociative example, note that the error is due to the fact that $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are not orthogonal.

A5.8 LMS and heteroassociative learning

The LMS algorithm applied to heteroassociative learning is simply an extension of the autoassociative equation given in equation (8). Since we want to improve our estimate of *T*, we simply replace the term $X - \hat{X}$ with $T - \hat{T}$. That is:

$$\begin{aligned} W^T(n) &= W^T(n) + \alpha(T - W(n)X)X^T \\ &= W^T(n) + \alpha\left(T - \hat{T}(n)\right)X^T \\ &= W^T_{(n)} + \alpha E(n)X^T \\ &= W^T_{(n)} + \Delta W(n), \end{aligned} \tag{A5.22}$$

where *E(n)* is the error at the nth iteration, and *ΔW(n)* is the matrix of corrections at the nth iteration. Let us apply equation (A5.22) to our previous example.

A5.9 Example

Recall that $X = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}$ and $T = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. Let us assume that $W^T(0) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\alpha = 0.25$. (Recall that $\alpha < \frac{2}{\lambda_{\max}} = \frac{2}{4} = \frac{1}{2}$, so this is an acceptable value.) Thus:

$$W^T(1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 0.25\left\{\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\right\}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}.$$

As expected, $W^T(1)$ is α times our original estimate of *W*. For iteration 2, we get:

$$W^T(2) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + 0.25\left\{\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} - \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}\right\}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + 0.25\left\{ \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 0.5 & 0.5 \end{bmatrix} \right\} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + 0.25 \begin{bmatrix} 0 & 0 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0.25 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.75 \end{bmatrix}.$$

When we apply this updated weight matrix to *X*, we get:

$$\hat{T}(2) = W^T(2)X = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0.75 & -0.75 \end{bmatrix}.$$

Thus, our solution has improved. Using a computer we find that after 15 iterations we converge to:

$$W^T = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 1.0 \end{bmatrix} \Rightarrow \hat{T}(15) = W^T(15)X = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

A5.10 SVD and LMS for heteroassociative learning

Since we now have the matrix *T* in our LMS calculation, the SVD solution is slightly more complex, and can be written (Abdi et al., 1999) as:

$$W^T(n) = TV\Delta^{-1}\Phi(n)U^T, \tag{A5.21}$$

where $\Phi(n) = I - (I - \alpha\Lambda)^n$ and *V*, *U* and Δ are as defined earlier. If α is chosen to be less than $\frac{2}{\lambda_{\max}}$, then:

$$W^T(\infty) = TV\Delta^{-1}U^T. \tag{A5.22}$$

Using our previous example:

$$W^T(\infty)=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}\frac{1}{2} & 0\\ 0 & \frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0.25 & 0.25 & 0.5\\ 0.25 & 0.25 & -0.5\end{bmatrix}=\begin{bmatrix}0.5 & 0.5 & 0\\ 0 & 0 & 1\end{bmatrix}.$$

A5.11 The pseudo-inverse

We have seen that the Hebb rule can give the exact solution for our weight matrix in the equation

$$T = W^T X$$

only if the vectors in X are orthogonal. We also discussed an iterative approach to solving for W, called the LMS algorithm, and its implementation using singular value decomposition. However, there is also a direct matrix inversion approach that can be used. If the matrix X is square and has a non-zero determinant, we can write:

$$W^T = TX^{-1}.$$

However, in general X is not square. In this case, we use the generalized inverse, or

$$W^T = TX^+,$$

where $X^+ = \left(X^T X\right)^{-1} X^T$. Returning to our example, note that:

$$X^+ = \left(X^T X\right)^{-1} X^T = \left\{\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & -1\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & 1\\ 1 & -1\end{bmatrix}\right\}^{-1}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & -1\end{bmatrix}$$

$$=\begin{bmatrix}3 & 1\\ 1 & 3\end{bmatrix}^{-1}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & -1\end{bmatrix}=\frac{1}{8}\begin{bmatrix}3 & -1\\ -1 & 3\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & -1\end{bmatrix}=\begin{bmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{2}\\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{2}\end{bmatrix}$$

$$\Rightarrow W^T=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{2}\\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{2}\end{bmatrix}=\begin{bmatrix}\frac{1}{2} & \frac{1}{2} & 0\\ 0 & 0 & 1\end{bmatrix}.$$

APPENDIX 6 Derivation of the radial basis function approach

It was shown by Poggio and Girosi (1990) that the theoretical foundation of the basis function method could be derived using regularization theory (Tikhonov and Arsenin, 1977). Regularization theory involves controlling the smoothness of a mapping function by adding a second term that penalizes values that are not smooth. This is done by minimizing an error function given by

$$E = \sum_{j=1}^{N} \left[y(\boldsymbol{x}_j) - t_j \right]^2 + \lambda \left\| P y(\boldsymbol{x}_j) \right\|^2 , \tag{A6.1}$$

where λ is the regularization parameter, P is a differential operator; and y, x_j, and t_j are the final computed reservoir parameter value, attribute vectors and observed training values, respectively, as discussed throughout this dissertation. Equation (A6.1) can be solved using the calculus of variations (Bishop, 1995) by setting the functional derivative with respect to $y(x_j)$ equal to zero. This gives

$$\sum_{j=1}^{N} \left[y(\boldsymbol{x}_j) - t_j \right] \delta(\boldsymbol{x} - \boldsymbol{x}_j) + \lambda \hat{P} P y(\boldsymbol{x}_j) = 0 , \tag{A6.2}$$

where $\hat{P}$ is the adjoint differential operator to P, δ is the delta function and $\boldsymbol{x}$ is an arbitrary input sample vector. The equations in (A6.2) are better known as the Euler-Lagrange equations and have the solution given by

$$\hat{P} P G(\boldsymbol{x}, \boldsymbol{x}_j) = \delta(\boldsymbol{x} - \boldsymbol{x}_j) , \tag{A6.3}$$

where $G(\boldsymbol{x}, \boldsymbol{x}_j)$ are the Green's functions of the operator $\hat{P}P$. In the case of an operator which is rotationally and translationally invariant, as in the case of a radial basis function, the solution to (A6.3) is given by

$$y(\boldsymbol{x}) = \sum_{j=1}^{N} \boldsymbol{w}_j G\left(\left\| \boldsymbol{x} - \boldsymbol{x}_j \right\| \right). \tag{A6.4}$$

Poggio and Girosi (1990) consider several forms of the Green's function given in equation (A6.4). The most straightforward form is given by the differential equation

$$(-1)^m \nabla^{2m} G(\boldsymbol{x}) = \delta(\boldsymbol{x}) , \tag{A6.5}$$

where ∇^{2m} is the *m*-iterated Laplacian in *n* dimensions. This leads to the multi-dimensional spline function given by

$$G(\boldsymbol{x}) = |\boldsymbol{x}|^{2m-n} \ln|\boldsymbol{x}|. \tag{A6.6}$$

By generalizing equation (A6.5) to an infinite number of terms, we derive the partial-differential equation for the Green's function given by

$$\sum_{n=0}^{\infty} (-1)^n \frac{\sigma^{2m}}{m!2^m} \nabla^{2n} G(\boldsymbol{x}, \boldsymbol{x}_j) = \delta(\boldsymbol{x} - \boldsymbol{x}_j). \tag{A6.7}$$

By applying Fourier techniques to equation (A6.7) we arrive at the radial basis function solution given by

$$y(\boldsymbol{x}) = \sum_{j=1}^{N} \boldsymbol{w}_j \exp\left[-\frac{1}{2\sigma^2} |\boldsymbol{x} - \boldsymbol{x}_j|^2 \right]. \tag{A6.8}$$