Condition Monitoring of Pipelines

FRÉDÉRIC VIOT

Master of Science by Research
in Pattern Analysis and Neural Networks

ASTON UNIVERSITY

December 1999

ASTON UNIVERSITY

Condition Monitoring of Pipelines

FRÉDÉRIC VIOT

Master of Science by Research
in Pattern Analysis and Neural Networks, 1999

Thesis Summary

Structural components are often subjected to life limiting service conditions, such as vibrational fatigue, thermal fatigue, and corrosion. Monitoring the condition of such equipment is critical for its continued safe operation. Monitoring permits operators not only to assess the accumulated damage caused by system operation, but also to identify and avoid or minimise those operating conditions which result in significant damage or reductions in remaining life. The condition of any critical component can be monitored. Thus, people with heart conditions often have electro-cardiograms to monitor a variety of heart abnormalities. An industrially critical component is more likely to be a gearbox, a motor, a transformer or a pipeline.

This report deals with condition monitoring of pipelines for a utility services company. Data from ultrasound scans is analysed in order to detect the presence of pipeline defects. The approach uses data visualisation to aid feature extraction. The main tools employed are PCA, FFTs, AR models and neural network classifiers. The intent of the project is to solve the problem of detection. Indeed, most of the time when the alarm system is raised the region is not actually defective. This report explains a part of the work which has been done before, then the approaches we tried this year. Then, it deals with the two neural network classifiers we used: RBF networks and MLPs.

Keywords: Principal Component Analysis, Pre-processing, Autoregressive Models, Radial Basis Function Networks, Multi-Layer Perceptrons.

Acknowledgements

I am grateful to Dr Ian Nabney for his help, his useful advice and comments he provided me with during this nine month project.

I would like to thank Professor David Lowe for the time spent in introducing me to the work which has been done so far for this project.

I would also like to thank a number of students who pointed out errors in the original thesis and have displayed good humour during long nights spent on the project, particularly Nicholas Hughes, Franck Mertzweiller and Bruno Vincent. Doubtless there are still some mistakes, but the number has been considerably reduced thanks to their efforts.

Eventually, I am also thankful to all PhD students, especially Mehdi Azzouzi, and MSc students at the Neural Computing Research Group as well as the MSc IT students for their help and support throughout this project and for answering all my questions with so much patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

After a presentation of what condition monitoring consists of, this chapter explains why artificial neural networks are helpful in this area. Then, a thesis overview is given.

1.1 Condition monitoring and neural networks

1.1.1 Purpose of condition monitoring

The monitoring of the condition of structural components can significantly reduce the costs of maintenance. Firstly, it can allow the early detection of potentially catastrophic faults which could be extremely expensive to repair. Secondly, it allows the implementation of condition based maintenance rather than periodic or failure based maintenance. In these cases significant savings can be made by delaying scheduled maintenance until convenient or necessary.

To obtain an accurate measure of the condition of components, a wide range of approaches can be employed to extract indicative features. By comparing these features with features for known normal and probable fault conditions, the condition of the component can be estimated. The approaches used vary from measuring vibrations, which yield signals often requiring a significant amount of processing, to oil debris analysis where the metal worn off the machine can be analysed. Other approaches

include monitoring acoustic emission, electrical currents, temperature and system input/output relations [Neale and Associates, 1979].

1.1.2 Are neural networks helpful?

Condition monitoring requires, in most cases, the intervention of skilled personnel who have expert knowledge of the systems under their surveillance. To perform their tasks effectively they bring to bear a wide range of signal processing techniques and associated analytical techniques in order to establish functional linkages between acquired data measurements and operational condition. In this way a rational basis for fault diagnosis can be established.

It is in just this area that neural networks may have much to offer since they are ideally suited to the establishment of such linkages, yet do not require formal knowledge of the precise mechanism that may be involved. This suggests that neural networks could be used to make diagnoses regarding the condition as an alternative to the increasingly scarce resource of skilled personnel.

Neural networks cannot solve every problem, in some cases traditional statistical methods may be better. Nonetheless, neural networks usually are as good as traditional methods and can be much better. Both classical statistical methods and neural networks also assume a model of the data. However, the model space for neural networks is generally rather larger than for classical methods.

1.2 Thesis overview

The thesis consists of six parts. After this introduction to the project, the second chapter presents the first approach we used to visualise and understand the data. Moments of low order and Principal Component Analysis (PCA) are involved.

The third chapter presents the pre-processing step. It is explained in detail as this was one of the most important parts of the work. The pre-processing is based on the selection of the sensor which will 'best' represent the area of suspected corrosion and on dimensionality reduction using autoregressive models. Other steps involved are the quantisation and the 'zero phase' which allows alignment of scans and boxes. The last pre-processing step was the reduction of the number of classes as the data from the 'double results' classes were sparse. Then, neural network classifiers (Radial Basis Function networks (RBF) and Multi-Layer Perceptrons (MLPs)) were trained. The autoregressive coefficients were the input variables. This is discussed in chapter 4 as well as general results obtained.

Finally, a summary of the study and some comments about the methods used are given in chapter 5. Then, conclusions are drawn and some possible ways of implementing the procedure are discussed.

Note that, for obvious reasons, the results are not displayed for the whole data set used. Throughout this report, the files mt2891_50.dat and mt2891_50.sel are used unless otherwise stated in order to follow clearly the different steps. These files have been chosen because they represent the complete data set quite well.

Chapter 2

Data understanding and visualisation

Data understanding and visualisation are crucial steps in general. Plotting a graph in a suitable format distinguishes features such as peaks that characterise the data. It allows planning of the pre-processing required to enhance those features as well. Firstly, this chapter says more precisely what the project deals with and the work which had been done earlier. An explanation of the first steps is given in this section. Moments of low order and Principal Component Analysis are involved. Then, some conclusions from the results are drawn and some explanations about the problems encountered are given.

2.1 Data selection and graphical interface

A circular array of detectors is passed through a pipeline. At regular intervals (scans), the signals are recorded from each detector (channel). Then, regions of metal loss are located by technicians as they visually inspect plots of the resulting 2-dimensional array [R. Rohwer and White, 1996]. This is a time-consuming procedure which will be done automatically soon.

An existing software tool uses an algorithm which locates the potential defects

within the 2-dimensional array. It estimates their lengths in scans and widths in channels. This algorithm is highly effective at locating all defects (D) of interest, but also registers spurious (S) features due to harmless variations in metal thickness such as occur at pipe bends, welds and off-takes, and in the process of manufacturing seamless pipes. Unfortunately, the algorithm detects many times more spurious features than defects.

Four types of files are involved in this project. The first type of files (.era files) contains the 2-dimensional ultrasonic signal array for each weld-to-weld pipeline section. The second type of files (.els files) contains the coordinates for each area (or 'box') of suspected corrosion, along with estimates of the size of the defect (peak depth, length and width...) as well as the type of defect and the box identification. Obviously, there is one file of the second type corresponding to each file of the first type. Throughout this report, the words 'spool' and 'pipeline section' will be used as equivalent. In the same way, any area of suspected corrosion will be a 'box'.

The graphical interface has been done in late 1998 by Dr Ian T. Nabney and a PhD student, Mehdi Azzouzi. Its purpose is to display the data. The zoom and various statistics provided by this application have been very helpful in analysing the data.

In a figure, the 2-dimensional ultrasonic signal array is plotted in green. The box delimiters are also plotted with their class using the following colour code:

- Red : if Class is 0 (S),
- Blue : if Class is 1 (D),

Figure 2.1 displays an example of the data visualiser. It is important to look at the shape of the data in blue boxes as they represent the defects.

One problem with these files is that the use of boxes does not give a standard width (number of scans) and a standard height (number of sensors), whereas neural network

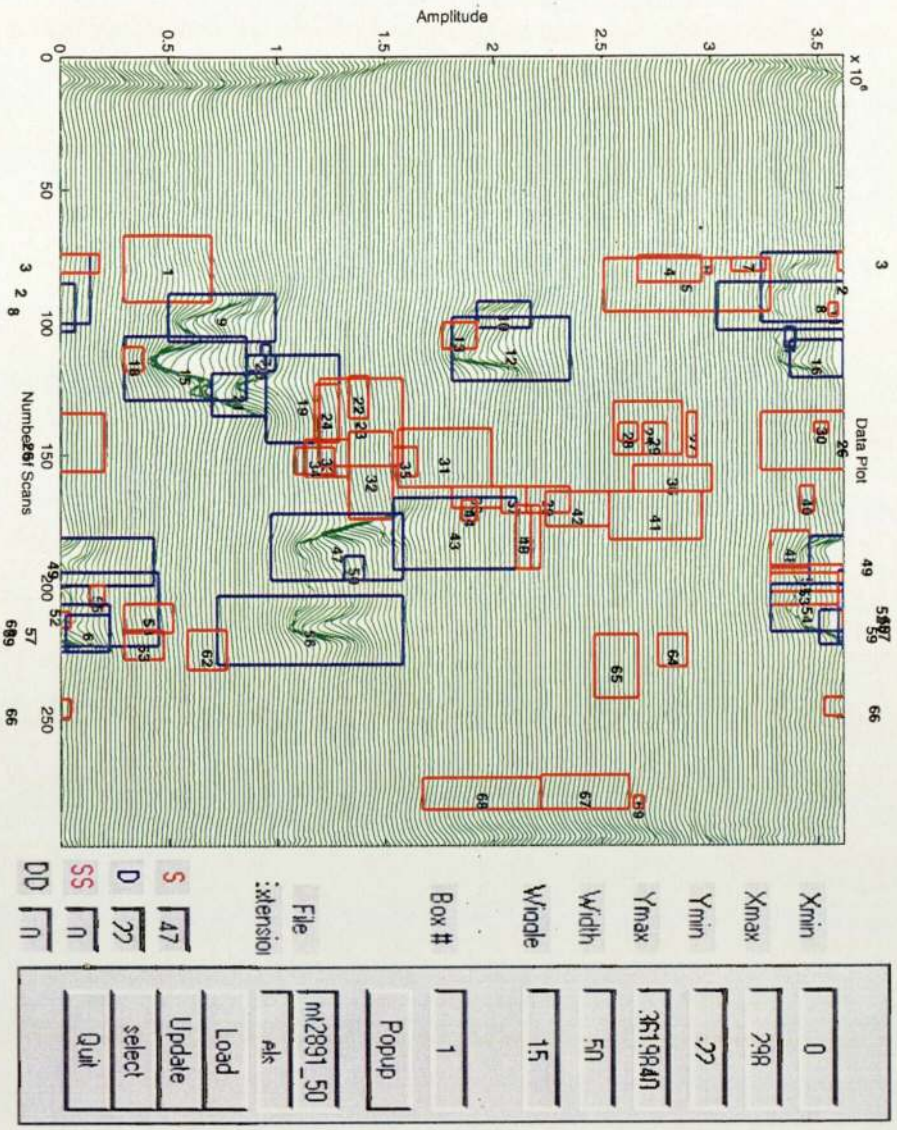

Figure 2.1: Data visualiser displays mt2891_50.era and mt2891_50.els files. The number in the middle of each box is a box identification number. It is chronological and starts at 1 for each spool.

classifiers operate on fixed-dimension vectors. In order to solve this problem a third file type (.sel files) has been created.

The boxes found by the segmentation algorithm are gathered by a hand procedure into new boxes whose length is equal to 51 scans. These new boxes are horizontally centered in the box around the defect and also roughly vertically. The values of the 2-dimensional ultrasonic signal in these new boxes are in the last file type (.dat files). The data selection was done by the project supervisor Dr Ian T. Nabney.

The corresponding .dat and .sel files are displayed in Figure 2.2 where it can be seen that all boxes contain the same number of scans. Two 'double categories' have also been added : DD for boxes containing two features and SS for boxes in welds. Examples are displayed in Figure 2.2 : box 10 is a DD and box 11 is a SS. From now on, to take the new classes into account the following colour code is used:

- Red : if Class is 0 (S),
- Blue : if Class is 1 (D),
- Magenta : if class is 2 (SS),
- Black : if class is 3 (DD).

During the project, most work was done using the last two types of file since they provided the best information.

2.2 Moments

2.2.1 Properties of the data

As a preliminary step, before computing any sophisticated procedures, a basic idea is to look at moments of the data. Interest is usually directed towards the lower moments, especially the first and second, since these serve to describe the most important properties of any data set [Spencer et al., 1977]. Thus, the arithmetic mean ($\bar{x}$) (equation 2.1) of a sample, referred to simply as the mean, is in general the most useful. The variance (s^2) (equation 2.2) is generally the most useful measure of the spread of the data. It is also interesting to see whether the data comes from a normal distribution or not. That can be done by looking at its symmetry since that is one of the principal properties of the normal distribution. To characterise this fact, rather than simply estimating the percentage of data points higher and lower than the mean, as suggested by [Miller and Ruppert, 1986], more useful measures are given by the skewness (γ_1)

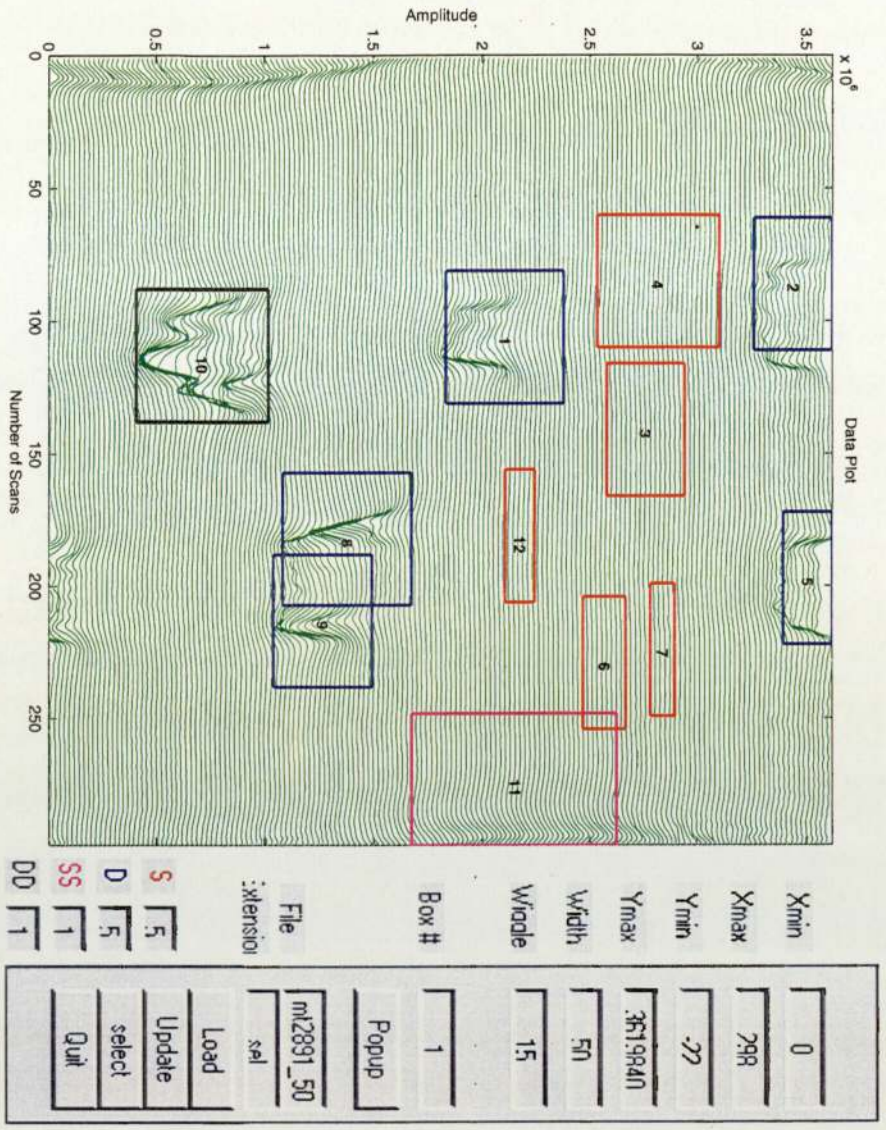

Figure 2.2: Data visualiser displays mt2891_50.dat and mt2891_50.sel files. Example of boxes DD and SS.

(equation 2.3) and the kurtosis (β_2) (equation 2.4) which are respectively the moments about the mean of orders 3 and 4 with a normalisation factor.

The skewness provides a measure of the lack of symmetry of the data. Thus, a distribution with a right tail heavier than the left has a positive skewness. Similarly, when the tails of the distribution contain more mass than the normal distribution, the kurtosis is positive, whereas skewness and kurtosis are zero for the normal distribution. Note that the kurtosis, like the other even orders, is associated with spread.

The higher the order of the moment, the more difficult it is to estimate it reliably, since the variance of the estimate gets larger.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.1}$$

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{2.2}$$

$$\gamma_1 = \frac{\sqrt{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{(\sum_{i=1}^{n}(x_i - \bar{x})^2)^{\frac{3}{2}}} = \frac{\sqrt{\frac{n}{(n-1)^3}}\sum_{i=1}^{n}(x_i - \bar{x})^3}{(s^2)^{\frac{3}{2}}} \tag{2.3}$$

$$\beta_2 = \frac{n\sum_{i=1}^{n}(x_i - \bar{x})^4}{(\sum_{i=1}^{n}(x_i - \bar{x})^2)^2} = \frac{n}{(n-1)^2}\frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{(s^2)^2} \tag{2.4}$$

where:

x_i is the value of the 2-dimensional ultrasonic signal at the scan i,

n is the number of scans per box.

2.2.2 Graphical visualisation and results

To illustrate the results of the investigation carried out, some graphical representations are given. Figures B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10, B.11 and B.12 display the results using a colour code (spurious results in red and defects in blue) in order to differentiate between the types of defects. These results were obtained using the representative files : mt2891_50.

The mean and the other moments were worked out separately over each box from the different files. So, for the representative files, 12 graphics corresponding to the 12 boxes for each of the four moments were calculated.

All the sensors from each box are represented. These moments give a good idea of the structure of the data. We can notice several properties on these examples which are true on the whole data set:

- similar values of the defects and spurious results for the four moments,

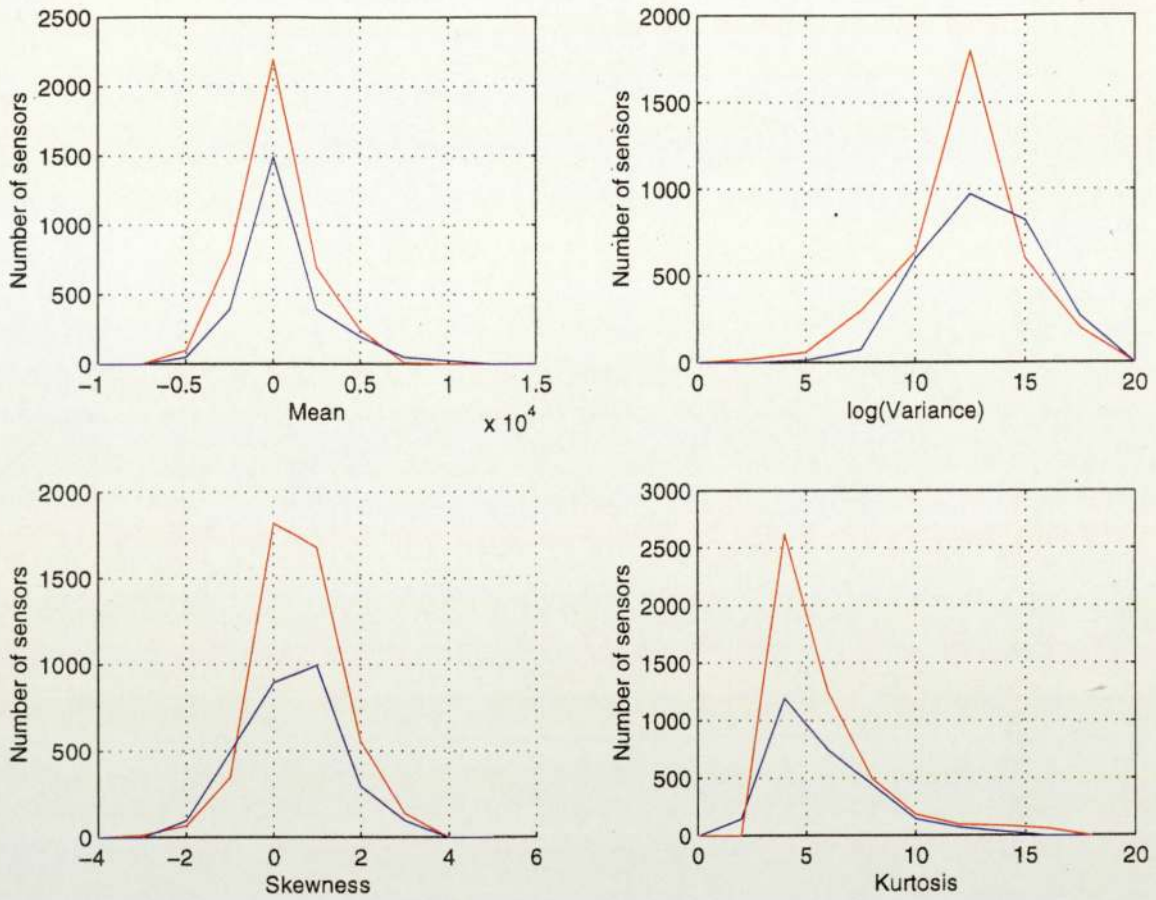

Figure 2.3: Schematic plot of the histograms of the mean, log(variance), skewness and kurtosis for the whole data set. Spurious results are plotted in red and defects in blue. Note that there is a significant overlap between the two histograms in each figure.

- numerous peaks.

Figure 2.3, which displays a schematic plot of the histograms of the moments, shows that there is a poor separation of defects and spurious results. So, if a new pattern is observed it is not known if it is more likely to belong to the class 'spurious results' or to class 'defects'. Therefore, no conclusion can be drawn simply on the basis of the values of these moments.

2.3 Principal component analysis

Lots of problems arise when attempting to perform pattern recognition in high-dimensional spaces. Better generalisation can often be obtained by first mapping the data to a space of lower dimensionality. However, a reduction of the dimensionality of the input space

of eigenvalues just after the point at which the cumulative contribution flattens out is therefore usually an appropriate number. In figure 2.4, the number found, using this 'knee' technique is approximately 6. Therefore, it is interesting to concentrate our attention on the 6 first principal components for this subsample of data.

2.3.3 Graphical visualisation and results

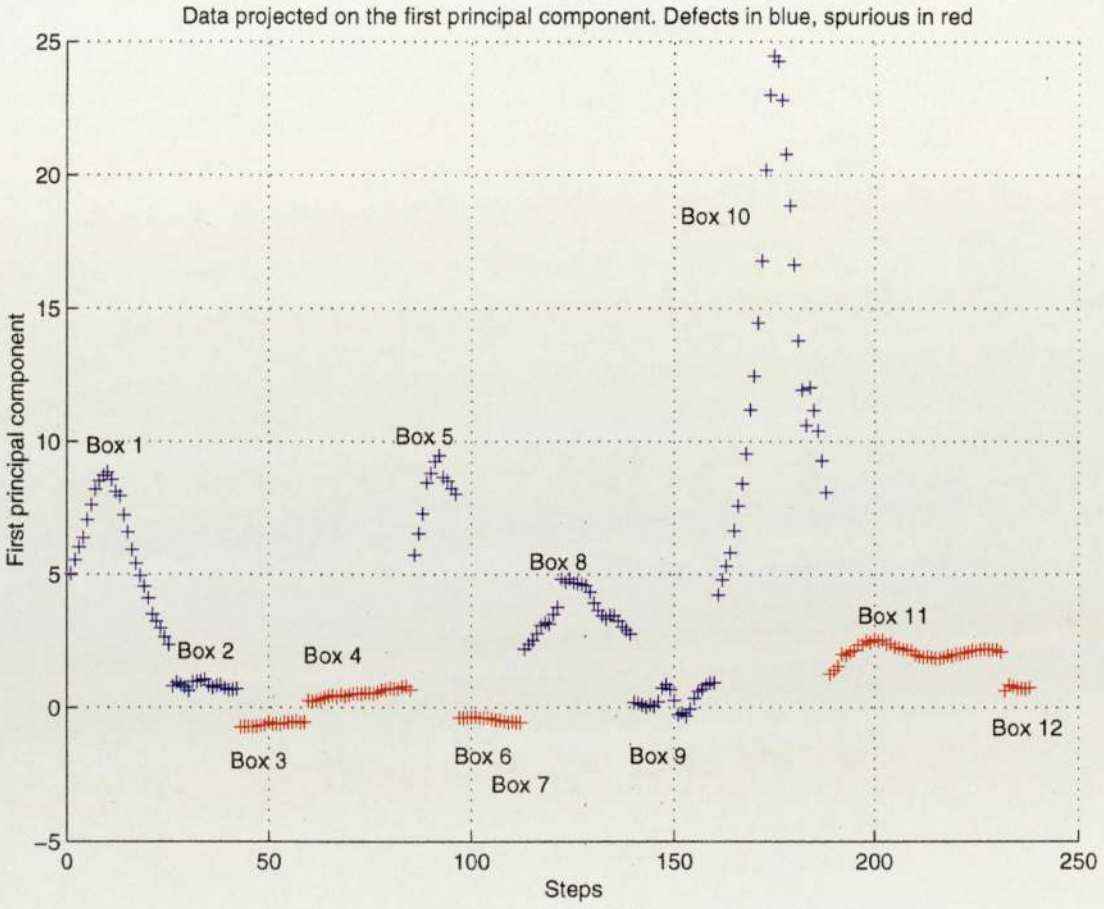

Figure 2.5: Data projected in 1-dimensional feature space (spurious results in red, defects in blue). Note the peaks for the defects. Each group of data points corresponds to one of the 12 boxes that contain the representative files displayed in figure 2.2.

Figures 2.5, 2.7 and 2.8 display the results obtained with the representative files. Figure 2.6 displays the results obtain with the mt2891_33 files. The first two graphics show respectively projections onto the first principal component. The third graphic shows projections onto the first two principal components and the last graphic shows a 3-dimensional representation of the projection onto the first, the second and the third

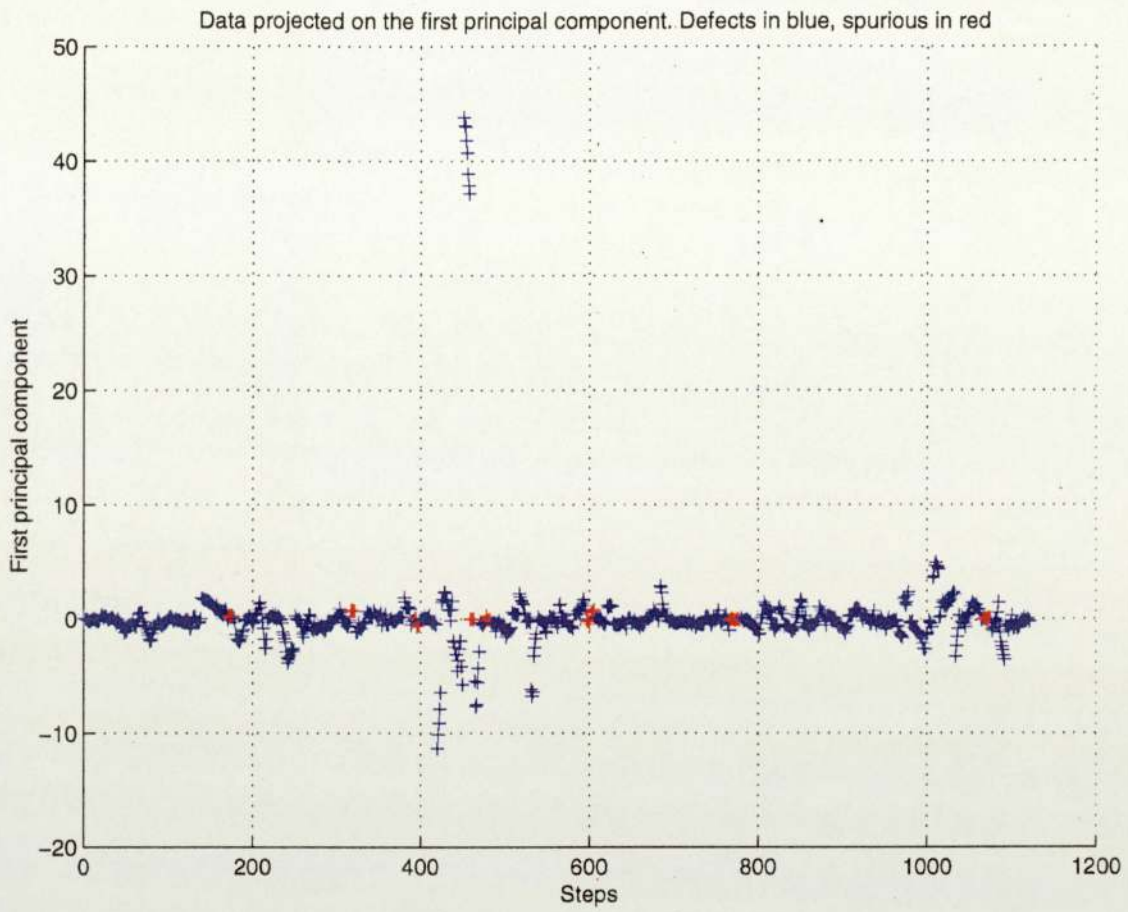

Figure 2.6: Data projected in 1-dimensional feature space (spurious results in red, defects in blue). Files mt2891_33.

principal components.

Each point corresponds to a 51 dimensional vector which is the set readings for a sensor across the box. All the sensors within the box are used for the PCA. Boxes are displayed from the left to the right. That is to say that the first points (on the left side of the graphic) correspond to the first sensors from the first box of the file and the last points (on the right side) correspond to the last sensors from the last box (box 12 for the mt2891_50.dat).

The general structure of the data can be seen: There are numerous peaks each of which corresponds to a defect. The spurious results are grouped around the same values.

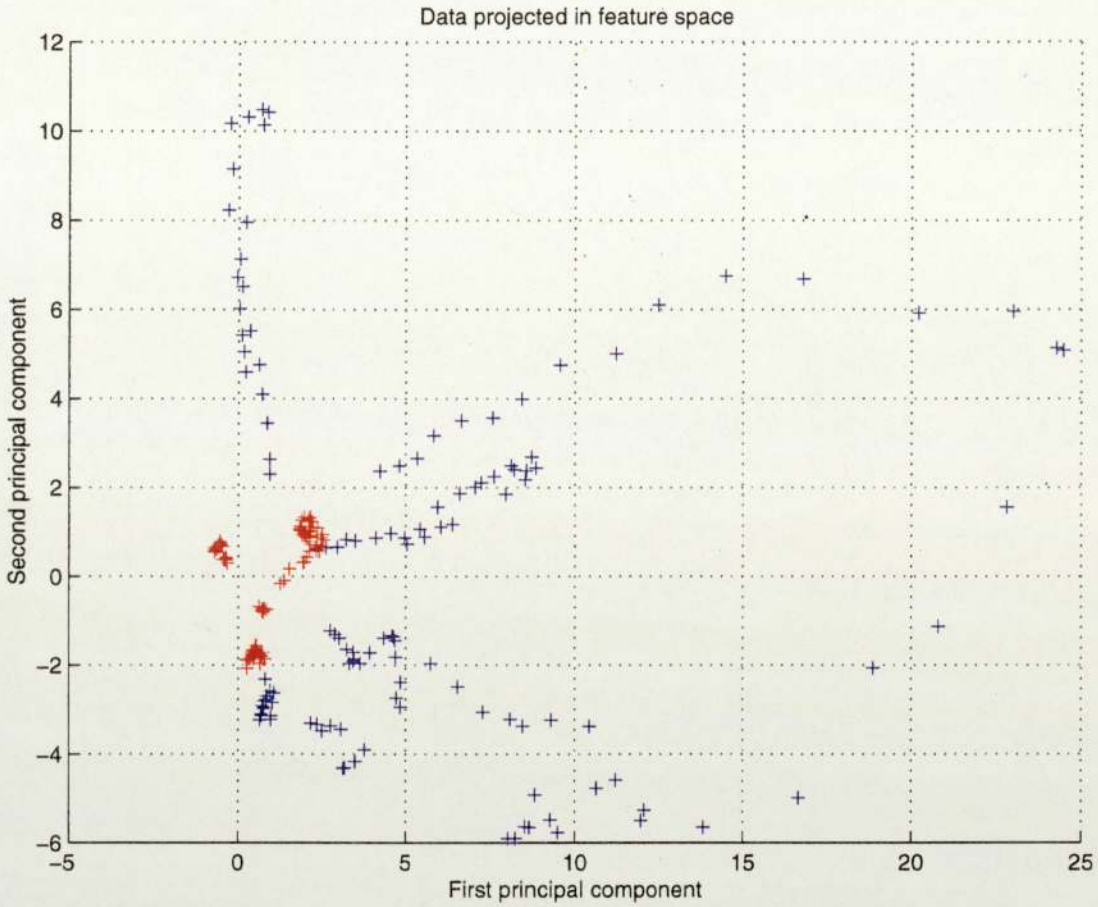

Figure 2.7: Data projected in 2-dimensional feature space (spurious results in red, defects in blue). Note the similar structure of the data points in comparison with Figure 2.5. Files mt2891_50.

From Figure 2.5, we could conclude that PCA is a good option to use to separate the data because all defects have a projection onto the first component with a value greater than about 3.5. There is a clear separation between spurious results and defects. Therefore, we could reliably use this information to help in the classification process. However, from Figure 2.6, we can see that this information would be more difficult to use as the results are more mixed, only a few points could be correctly classified. And it is even worse when we use several files since the results vary from file to file.

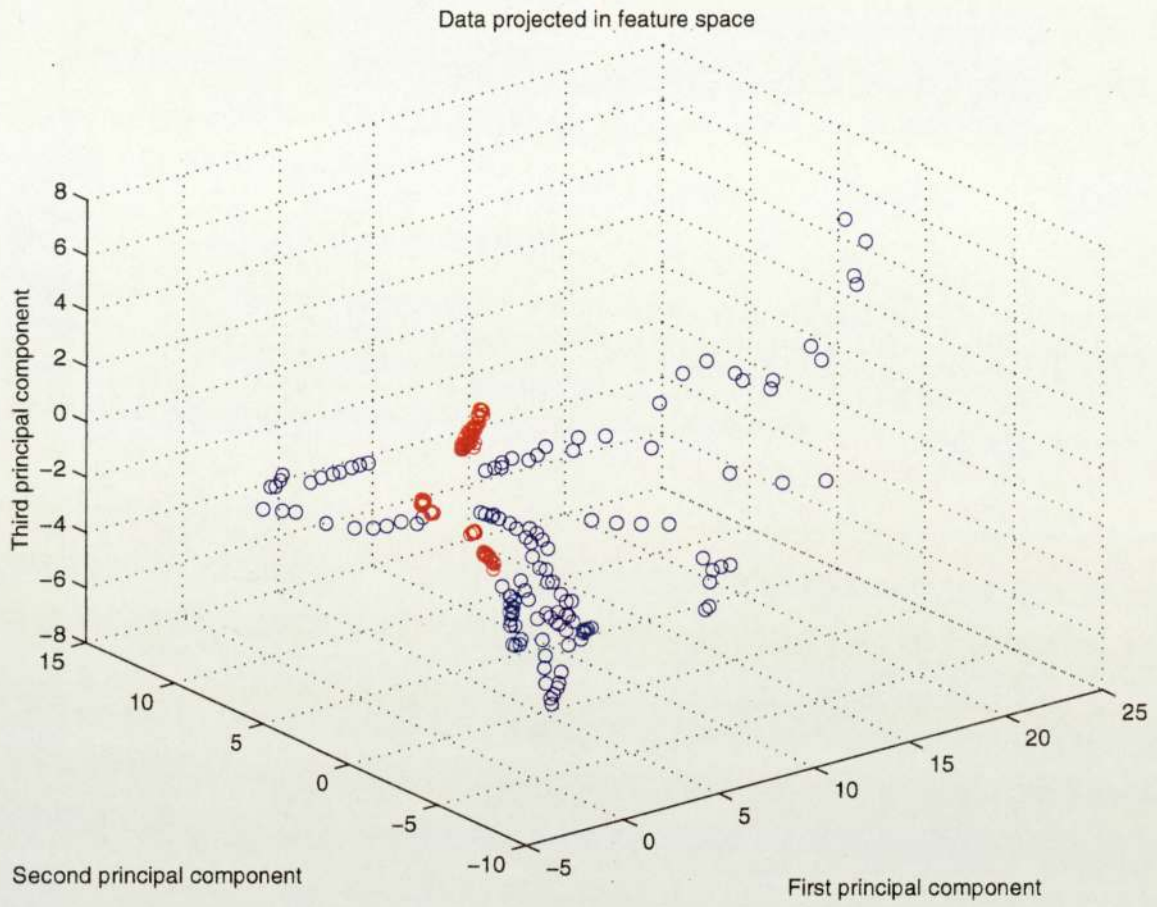

Figure 2.8: Data projected in 3-dimensional feature space (spurious results in red, defects in blue). Files mt2891_50.

2.3.4 Conclusions

This method has to be used carefully. In our example, we discovered that the set of 51-dimensional data can be described with a relatively high accuracy in approximatively 6 dimensions by applying PCA. This is the result of correlations within the data. However, PCA is limited by being a linear technique and may overestimate the true dimensionality of the data. Another problem which may arise is a loss of relevant information for the classification phase as PCA does not take account of the target data.

2.4 Conclusions

At the beginning of an investigation, it is often valuable to understand or, at least, to have some indications of the nature of the data. It avoids making incorrect assumptions which can lead to problems later. We have shown that summary statistics are not useful to separate classes: PCA works better, but there is still a large amount of class overlap. There is quite a lot of variation in the feature values within each box, and even more from section to section, which means that models trained on these features will generalise poorly.

The goal of this approach involving unsupervised learning was to discover any underlying structure in the data. Understanding the data allows the choice of a good pre-processing method.

Chapter 3

Pre-processing procedure

The pre-processing procedure is explained in this chapter and need to be looked at carefully as it is one of the key steps in a classification problem. We started from a method designed by Professor David Lowe and we implemented it. The pre-processing stage is made up of several steps. The tools used in this pre-processing are numerous: Fourier transform, autocorrelation coefficients, autoregressive models... Then, the results will be used as inputs to neural network classifiers. Everything is described in this section except the neural network classifiers. Finally, some conclusions about the pre-processing step are drawn.

3.1 Introduction

First of all, it seems important to introduce the purpose of the pre-processing. It is rare that raw data are presented as inputs to neural networks except maybe in medical prognosis problems. Most of the time, preliminary work has to be done on these data as shown in figure 3.1.

The intent of the pre-processing step is to simplify the data to get the most relevant information. Generally, the main part of a pre-processing procedure is the reduction of the input dimensionality, which can be achieved either directly by principal components analysis (see previous chapter) or by extracting features from the raw data.

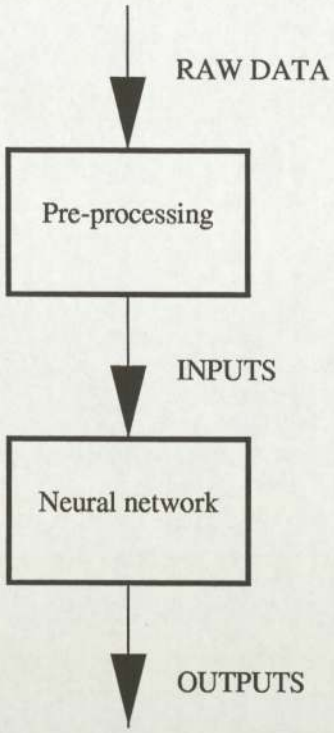

Figure 3.1: Most of the time, neural networks require the original data to be transformed by some form of pre-processing to give a new set of variables which are then utilised as inputs to neural networks. [Bishop, 1995]

The interest is to reduce the number of free parameters in the network as it is strongly linked with the number of inputs. It is important to note that lots of tools are used and great computational demands may be made during this step. Figure 3.2 sums up the different steps of the pre-processing we used in our classification problem.

Then, figure 3.3 displays the original data[1] from a pipeline section. Each curve corresponds to a sensor. A box is composed of several sensors and each sensor contains the measures of 51 ultrasound scans. Note that the colour code is not used yet. It is important to note that we can distinguish two types of sensors in the figure. With the first type of sensor, all the 51 ultrasound scans have approximately the same value. With the second type of sensors, there is a dip, close to the median scan. These types correspond respectively to spurious results and defects.

We had a set of 457 boxes composed of 209 spurious results and 248 defects. This

[1]Data used comes from the files mt2891_50.dat and mt2891_50.sel

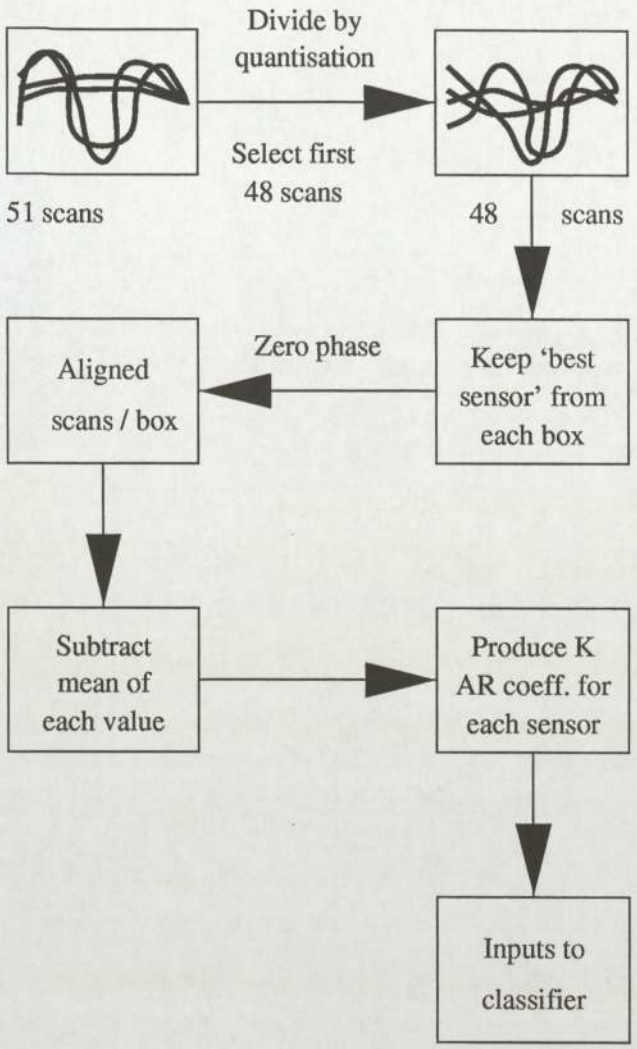

Figure 3.2: Pre-processing stage. Note the different steps of the stage. Each step is explained in this section.

number is highly important as it is one of the key points of the application. Too little data may lead to an inaccurate classifier. On the other hand, too much data leads to very long training times. Thus a sufficient amount, not too low and not too high, of data has to be used and the partitioning between sets has to be done carefully. Another important thing is to have the same number of data coming from the different classes allowing balanced training, validation (if necessary) and test sets. We will return to these points later in this chapter.

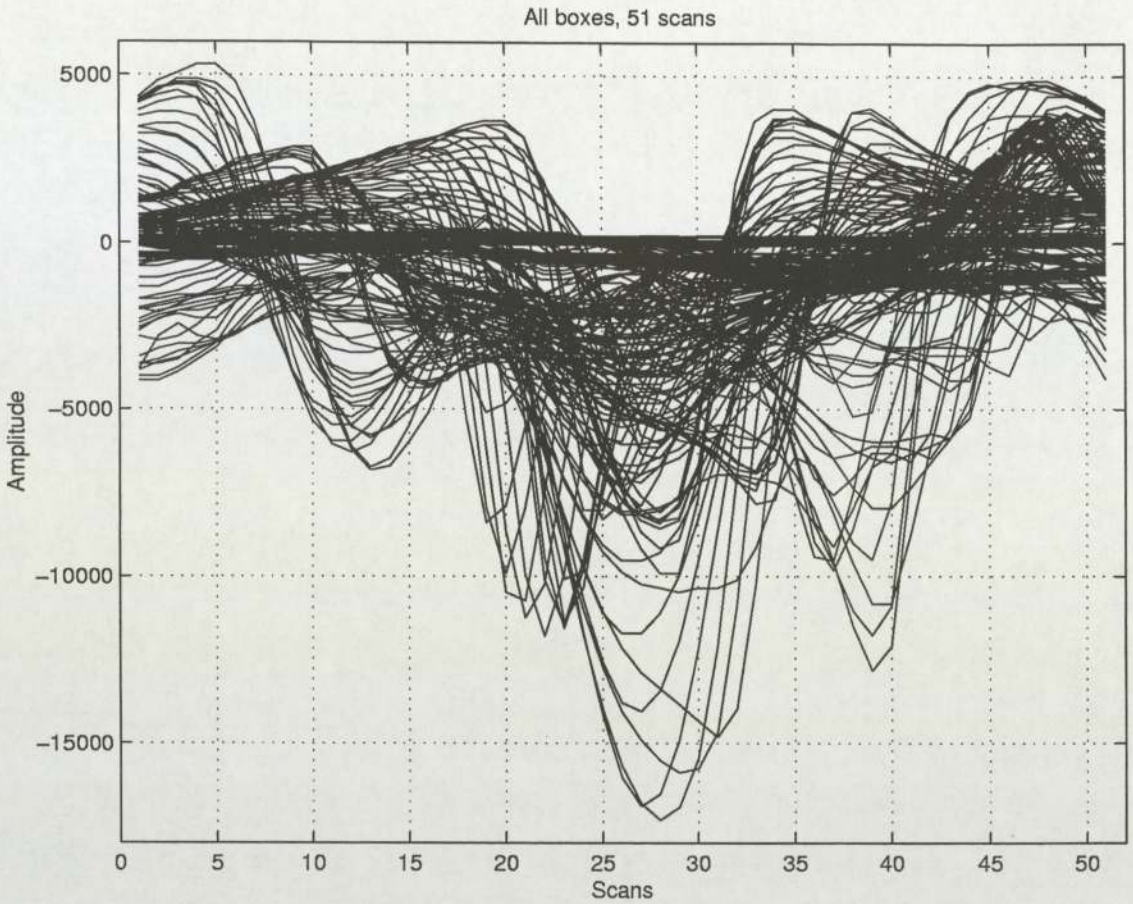

Figure 3.3: Original data from a pipeline section. All sensors from all boxes are displayed. We can notice that there are 51 ultrasound scans for each sensor and that some sensors have approximately the same values through the scans.

3.2 Quantisation

All sensor values are multiples of 16, so the first thing to do is to divide each scan by 16 in order to simplify the computations. It is obvious that it does not affect the general results.

Then, only the first 48 scans from each box are used. This number was chosen because the reshape simplifies and accelerates the data processing especially during the use of the Fourier transformation (FFT) introduced later. In addition, the FFT is most efficient when the window width is a power of 2. Note that these parameters are really easy to modify by the user as they are declared and defined at the top of the main file of the application I designed (see code in appendix).

3.3 Selection of the most representative sensor from each box

This is certainly the most important step in the pre-processing work. At this moment the choice of a single representative sensor from each box is made. This is done to reduce the computational demands of the pre-processing stage. Also the boxes chosen algorithmically (.sel files) often extend vertically a long way from the defect, so a defective box will contain many 'normal sensors' as well. In the next subsections, the two possibilities which have been coded and the results on the representative files are explained.

3.3.1 First method: the middle sensor

The first method we use is to select the middle sensor from each box. This method was the one used by Professor David Lowe. This criterion was chosen because the middle sensor is supposed to be the most representative sensor of each box when the boxes were hand selected.

Now, assume that the representative data files have M sensors distributed in N boxes before the pre-processing step. As we keep only the middle sensor of each box there are only N sensors kept. Results on the file are shown in Figure 3.4. In this case, N is equal to 12 (see Figure 2.2). We can notice more clearly the two different shapes of sensors (with and without dip) discussed earlier in the chapter.

3.3.2 Second method: the sensor with the biggest dip

The second method we use is the selection of the sensor with the biggest dip. Once again, if we assume that the files have M sensors distributed in N boxes before this step, we will keep only N sensors.

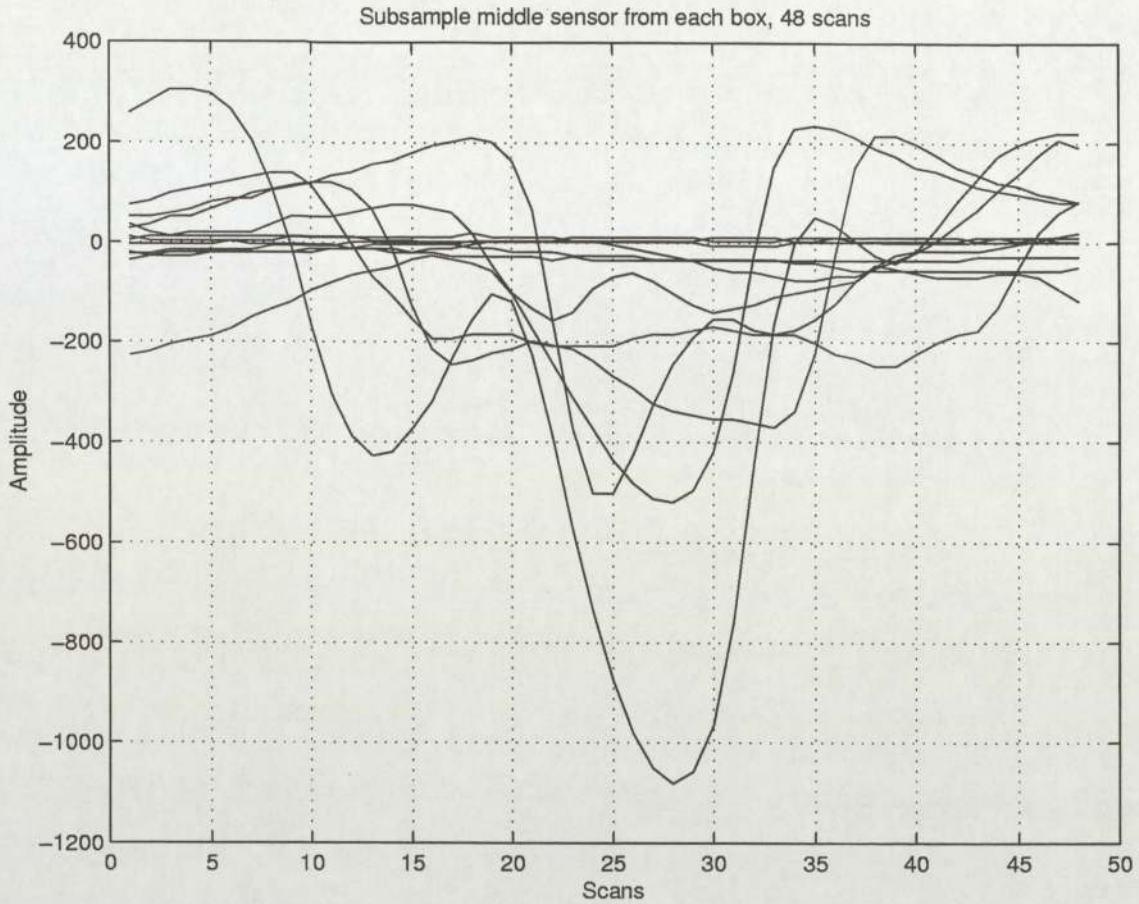

Figure 3.4: Method 1 : Selection of the middle sensor from each box of a pipeline section. As the file contains 12 boxes, only 12 sensors are kept, one per box. Note that every sensor now contains only the first 48 ultrasound scans.

The dip[2] is defined to be the difference between the mean of this sensor and its minimum. Thus, for a sensor:

$$Mean(sensor) = \frac{1}{n}\sum_{i=1}^{n} s_i \tag{3.1}$$

$$Dip(sensor) = Mean(sensor) - \min_{1\leq i\leq n}(sensor) \tag{3.2}$$

where:

s_i is the value of the sensor at the scan i,

n is the number of scans per box.

[2]Note that this quantity is always positive (or zero).

And obviously, the biggest dip from a box is defined by:

$$BiggestDip = \max_{1 \leq i \leq m} (Dip(i)) \tag{3.3}$$

where:

m is the number of sensors in the box.

An option is useful with this type of selection. As the boxes from the .sel files are hand selected, scans at the edges of these boxes can be far from the defect itself. Therefore, these scans may introduce uncertainty in the computation of the biggest dip of a sensor. So this option allows the user to exclude the scans at the edges of a box. In other words, it means that the computations of the mean and the minimum of a sensor do not use a number, specified in the option, of scans at both sides of the box. For example, with k ignored scans, the previous formula become:

$$(Equation\ 3.1) \Longrightarrow Mean(sensor) = \frac{1}{n-2k} \sum_{i=1+k}^{n-k} s_i \tag{3.4}$$

$$(Equation\ 3.2) \Longrightarrow Dip(sensor) = Mean(sensor) - \min_{1+k \leq i \leq n-k} (sensor) \tag{3.5}$$

where:

n is the number of scans per box,

k is the number of ignored scans.

3.3.3 Conclusions

Due to computational demands, a choice has been made: to keep only one sensor per box. There are other ways to select the most representative sensor from a box. These ones are the most intuitive. A fruitful direction could be to select several sensors per box in order to resolve some of the difficulties in misclassification. We could classify each sensor and use a voting system to classify the box. Therefore, it would be interesting in the near future to work further on this step.

3.4 Zero Phase

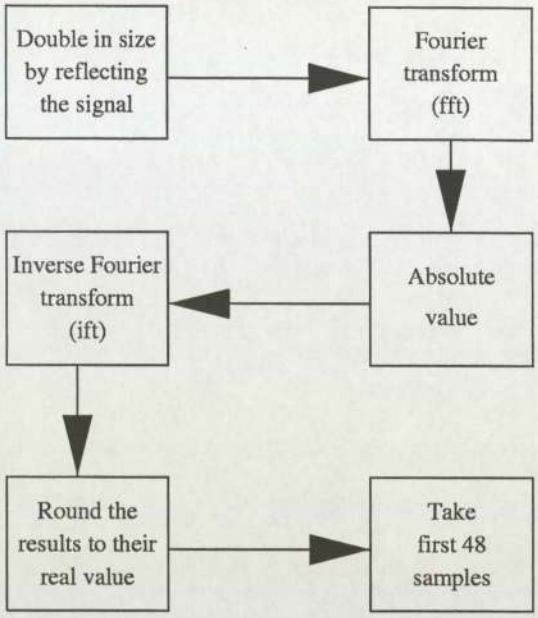

Figure 3.5: Zero phase: steps involved

The intention of this procedure is to align scans inside boxes in a consistent way. It also aims to make the representation translation invariant. As shown in figure 3.5 the first step is to double the sensors in size by reflecting the ultrasonic signal in order to have data of the same length after the discrete Fourier transform (fft) and inverse discrete Fourier transform (ift). We also tried to double the sensors in size by padding each sensor with 48 zeros but the results were not as good. Then, we apply the discrete Fourier transform to each sensor and we take the absolute value of these coefficients, and apply the inverse discrete Fourier transform. We round the results to their real value because we know that after the ift we want a real signal. However, because floating point arithmetic is used rounding errors are possible after the fft and ift. Therefore, we have to round off to zero the small imaginary values after the ift. Finally, we take the first 48 samples from each sensor to obtain vectors of the same length as the original ones.

Then the data was centered vertically by subtracting the mean of each sensor. Figure 3.6 shows the results on the data after this step.

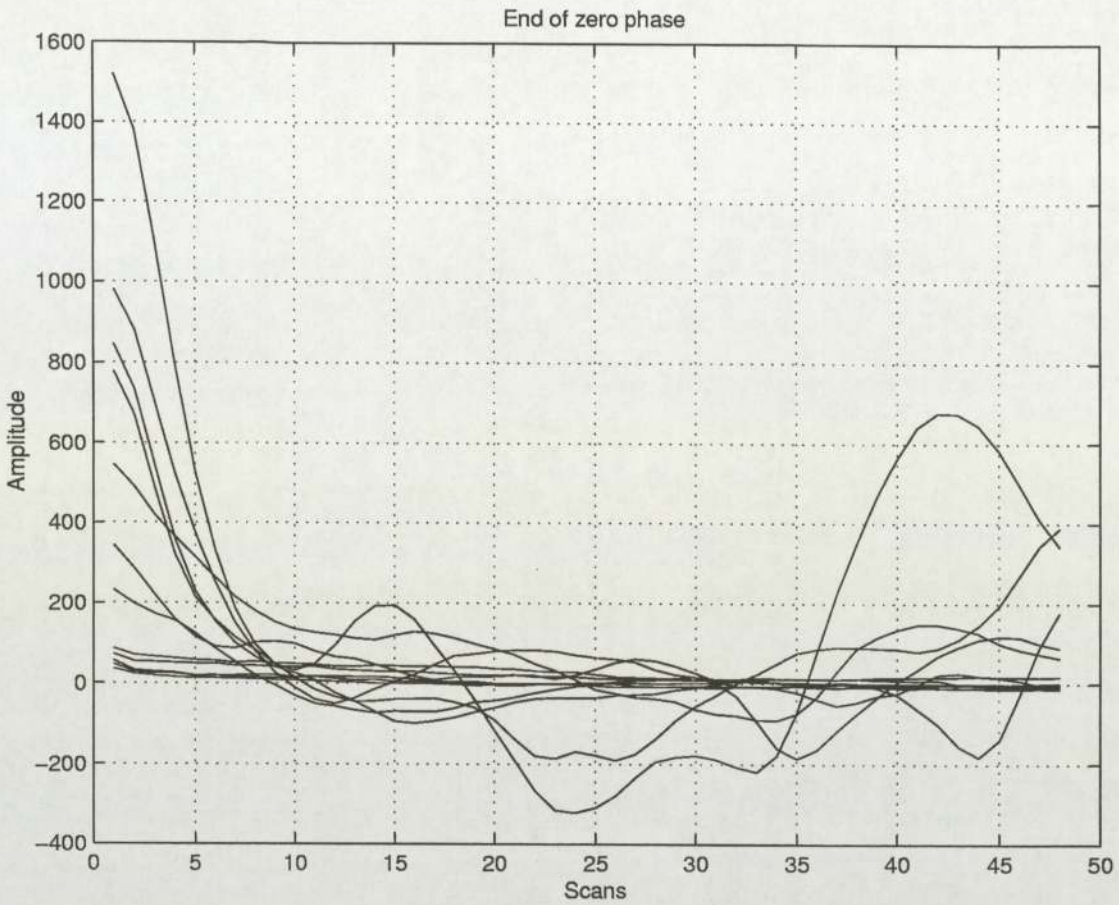

Figure 3.6: Results on the representative data files after the 'zero phase'. Note that there are still 12 sensors (one per box) and each one still has 48 ultrasound scans. The method used for the selection of the 'best sensor' was the middle sensor method.

The next step in this pre-processing method was to reduce the dimensionality of the input space. It was performed using autoregressive models and it is described in the next section.

3.5 Autoregressive models

In this section, some explanation of the autoregressive (AR) models is given. Firstly, it provides a reminder of what an AR model is. It then explains how the AR coefficients were calculated using the Yule-Walker equations and how many coefficients were used later as inputs to the neural network classifiers. Finally, results and graphical visualisations are shown.

3.5.1 Overview

In an AR model there is a memory or feedback and therefore the system can generate its own internal dynamics. The definition that will be used here is as follows:

$$x_t = \sum_{i=1}^{N} a_i x_{t-i} + \epsilon_t \tag{3.6}$$

or $x_t \approx$ AR(N).

where:

a_i are the autoregressive coefficients,

x_t is the series under investigation,

N is the order of the AR model,

ϵ_t is the noise term.

Note that, the noise term or residue is almost always assumed to be Gaussian noise and that N is generally very much less than the length of the time series.

Equation 3.6 means that the current term of the series is estimated by a linear weighted sum of previous terms in the series. The weights are the autoregressive coefficients. The problem in AR analysis is to derive the best values for a_i given a series x_t. A number of techniques exist for computing AR coefficients. The main two categories are least squares and Burg method. Within each of these there are a few variants and the most common least squares method is based upon the Yule-Walker equations.

The most common method for deriving the coefficients involves multiplying the definition above by x_{t-d}, taking the expected values and normalising giving a set of linear equations called the Yule-Walker equations (see [Kendall and Ord, 1990]) that can be written in matrix form as in (equation 3.7):

$$\begin{pmatrix} \rho(1) \\ \rho(2) \\ \vdots \\ \rho(N) \end{pmatrix} = \begin{pmatrix} 1 & \rho(1) & \cdots & \rho(N-1) \\ \rho(1) & 1 & \cdots & \rho(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ \rho(N-1) & \rho(N-2) & \cdots & 1 \end{pmatrix} \begin{pmatrix} a(1) \\ a(2) \\ \vdots \\ a(N) \end{pmatrix} \tag{3.7}$$

Yule-Walker equations

where:

$\rho(i)$ are the auto-correlation coefficients[3],

$a(i)$ are the autoregressive coefficients,

N is the order of the autoregressive model.

The auto-correlation between y_t and y_{t-k} is defined[4] as follows:

$$corr(y_t, y_{t-k}) = \frac{cov(y_t, y_{t-k})}{\sqrt{var(y_t)var(y_{t-k})}} \tag{3.8}$$

where:

$var(y_t) = \varepsilon[y_t - \mu_t]^2$ is the variance[5],

$cov(y_t, y_{t-k}) = \varepsilon[(y_t - \mu_t)(y_{t-k} - \mu_{t-k})]$ is the auto-covariance.

The auto-correlation coefficients vary between -1 and $+1$ and values near ± 1 indicate a strong correlation.

It is intuitive that the higher the AR model order the better the results are. Nevertheless, we have to consider the noise in the data then the results may get worse with a higher order model. Therefore, we have two constraints: the first one is that N is generally much smaller than the length of the series (48 in our case) and the estimates generally improve with the order.

[3]The diagonal is $\rho(0) = 1$.

[4]$corr(y_t, y_{t-k})$ is more commonly denoted by $\rho(k)$. Note that $\rho(k) = \rho(-k)$

[5]$\varepsilon[X]$ represents the expectation of X.

In our case, the autoregressive coefficients will be used as a pattern to describe the box. Hence, we need a separately calculated set of coefficients for each box. Table 3.1 gives the coefficients for a number of model orders for a subsample of data[6] we used.

	Coefficients							
Order	1	2	3	4	5	6	7	8
1	0.8180	-	-	-	-	-	-	-
2	0.9213	-0.1262	-	-	-	-	-	-
3	0.9214	-0.1269	0.0008	-	-	-	-	-
4	0.9213	-0.1231	-0.0270	0.0301	-	-	-	-
5	0.9212	-0.1230	-0.0266	0.0271	0.0033	-	-	-
8	0.9208	-0.1224	-0.0263	0.0254	0.0157	-0.0063	0.0084	0.0239

Table 3.1: Table gives the coefficients for a number of model orders for the first box of the file mt2891_50.sel.

To visualise the results, it is useful to plot the root mean square error (RMS error) between the series estimated by the AR coefficients and the actual series. The RMS error can be computed using a discrete error formula (equation 3.9):

$$E^{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \| p_i - \hat{p}_i \|^2} \tag{3.9}$$

where:

p_i is the correct solution,

$\hat{p}_i$ is the proposed solution.

There is no general rule to determine a good autoregressive model order. Generally, the RMS error decreases quickly for the first orders and then more slowly. An order just after the point where the trend decreases is usually chosen even though more formal techniques exist, the most common used of which is the Akaike Information Criterion. We used the method called 'knee' explained earlier for determining the optimal autoregressive model order as we did to interpret the results of the cumulative contribution of the eigenvalues in PCA in the previous chapter. Results over the

[6]Data comes from the first box of the file mt2891_50.dat and mt2891_50.sel.

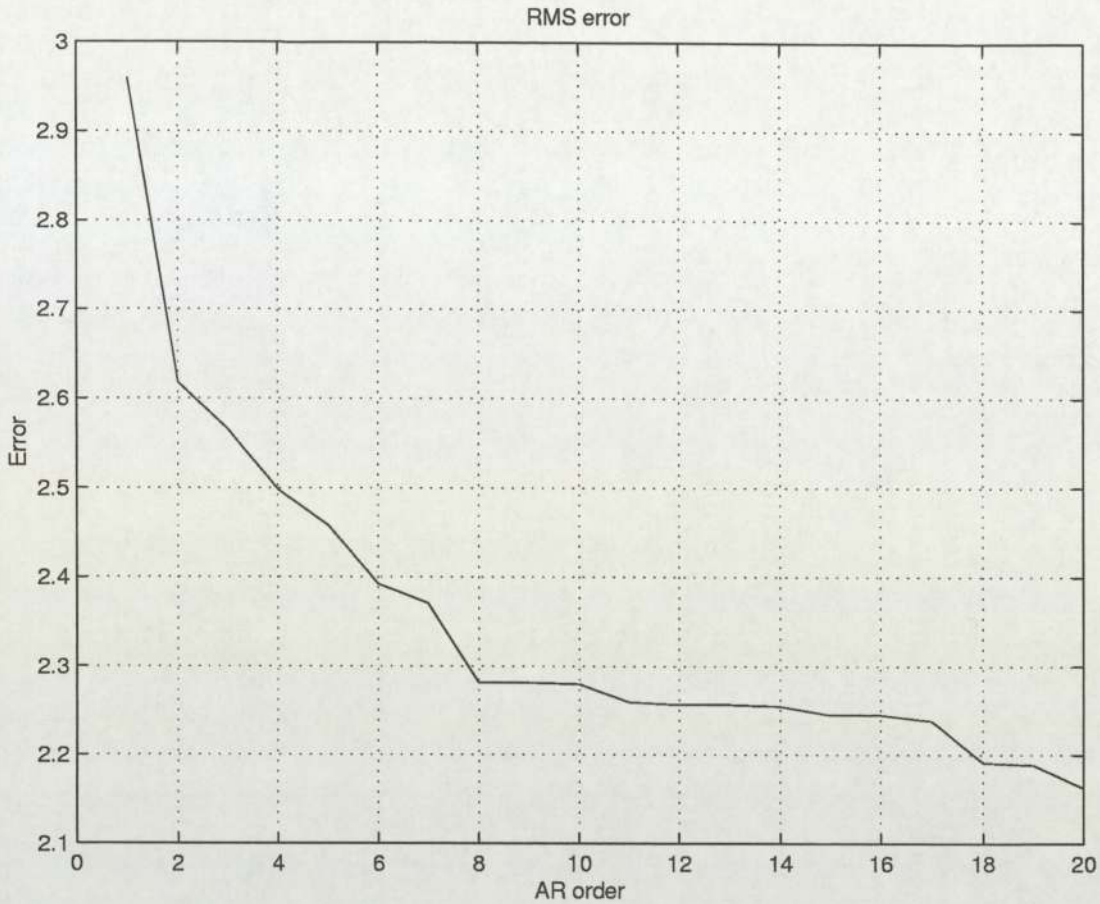

Figure 3.7: RMS error vs AR model order. The 'knee' of the curve is located at the eighth AR coefficient.

representative files are shown in figure 3.7.

3.5.2 Autoregressive model used

We chose to work with an AR model order equal to 8. We used the Yule-Walker equations to compute these coefficients. Therefore, 8 auto-correlation coefficients were required in order to solve these equations. Note that each box is treated separately so it means that different auto-correlation coefficients and AR coefficients had to be computed for each box. Each of these boxes was represented by its 'best sensor'.

So, there were A (where A denotes the number of boxes in the current file) sets of 8 AR coefficients for each file. As a sensor contains 48 values, there is a maximum of 95 auto-correlation coefficients which go from $\rho(-47)$ to $\rho(47)$, of which we just needed

$\rho(1)$ up to $\rho(8)$.

Then, once we knew the auto-correlation coefficients, by multiplying both the right hand side and left hand side by the inverse of the central matrix, we could work out the autoregressive coefficients.

3.5.3 Graphical visualisation and results

By projecting the first three autoregressive coefficients (figure 3.8) from the whole data set (see appendices for its composition) and using the colour code defined previously, we noticed that even when we superimposed several pipeline sections, the different types of defects were well-separated. We did not have these results with the first approach where the different types of defects were well-separated for a pipeline section but were all mixed when we superimposed several sections. Therefore, with this approach we have enough good results to use neural network classifiers where the inputs will be the 8 autoregressive coefficients.

A colour code[7] is used, so it is easy to see the separation between the defects and the spurious results even though the 'double results' are mixed. This particular issue only concerns a low percentage of data and is solved in the next section.

3.6 Reduction of the number of classes

The pre-processing step is now almost over. We saw before that it was best if the number of examples coming from all the classes in the different sets was approximately the same. This phenomenon is called balancing. The classes and their associated frequencies are detailed in table 3.2.

We notice that the 'double results' (ie SS and DD) are under-represented with 37 patterns out of 457. This can lead to a problem in the classification using neural

[7]Red (S), Blue (D), Magenta (SS), Black (DD).

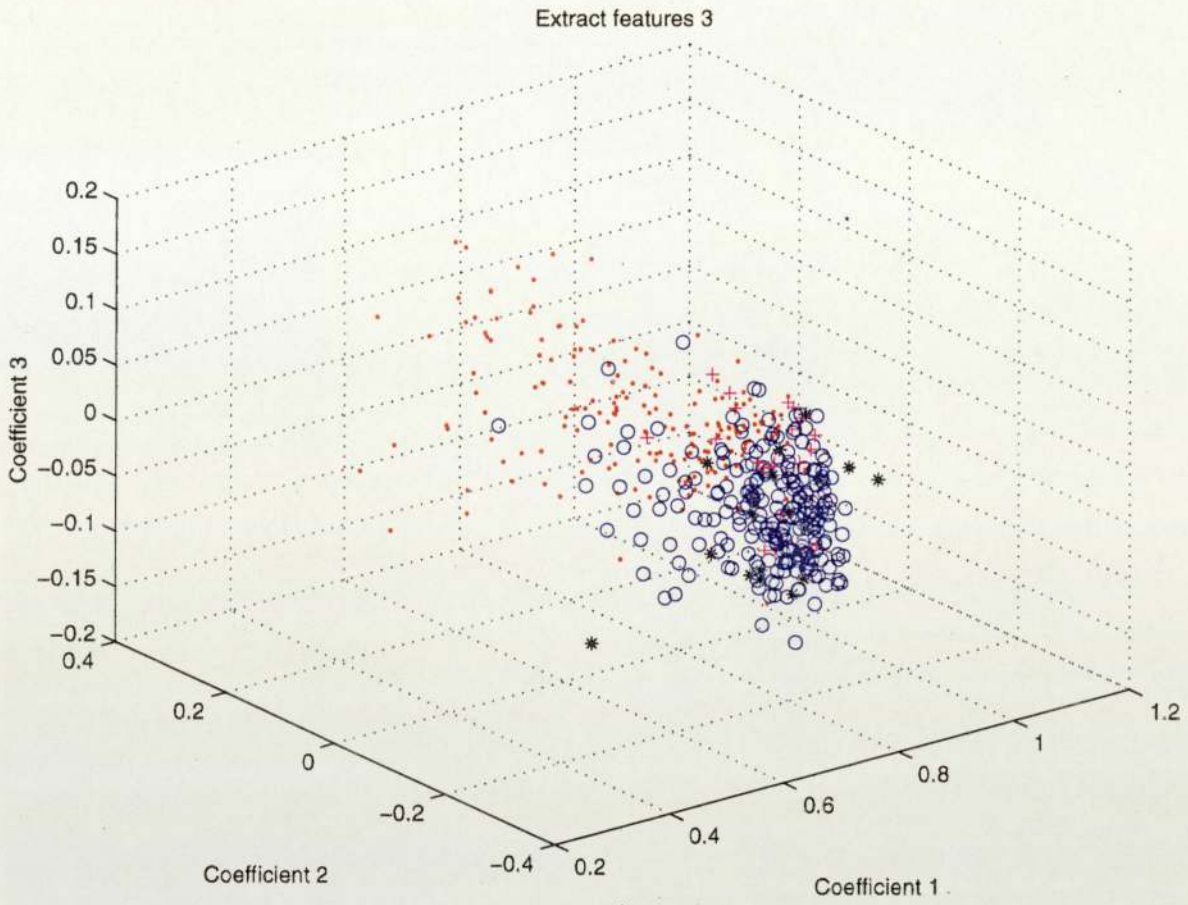

Figure 3.8: Projection of the first 3 AR coefficients. Note the separation between defects and spurious results. The method used for the selection of the 'best sensor' was the middle sensor method.

networks. With few examples from these two classes the network may not be able to determine the generating function correctly. To avoid this problem we decided to merge the four classes into two classes.

After closely looking at the shapes and properties of each kind of results, we mixed the classes D and DD and on the other hand S and SS. The results are displayed in the second line of table 3.2. In conclusion, only two classes will be presented to the inputs of the networks. By simplification, from now on we will call these classes the spurious results (S) and the defects (D).

S	SS	D	DD	Total
190	19	230	18	457
209		248		457

Table 3.2: Classes and their associated frequencies.

3.7 Conclusions

Throughout this chapter, we used different pre-processing methods in order to extract the most relevant information from the data. The data dimension was reduced from 51 to 8, which is the autoregressive model order. The autoregressive coefficients will be the inputs to the neural network classifiers introduced in the next chapter. We also merged the double classes as the data from them was sparse and it would not have allowed the network to learn enough relations between input patterns and their associated targets.

Chapter 4

Neural network classifiers

4.1 Introduction

In a classification problem the goal is to assign previously unseen patterns to the correct class based on previous examples from each class. Multi-Layer Perceptrons (MLPs) and Radial Basis Function (RBF) networks are the two most commonly used types of feed-forward networks. The fundamental difference between them is the way in which hidden units combine values coming from preceding layers in the network. While the MLP uses inner products, the RBF uses Euclidean distance. There are also differences in the customary methods for training MLPs and RBF networks, although most methods for training MLPs can also be applied to RBF networks. A more detailed comparison of the two types of network is given in [Bishop, 1995].

4.2 Multi-Layer Perceptron

Firstly, this section gives an overview of one of the two neural network classifiers used: MLPs. Then, it deals with its architecture and its training procedure, which involves three sets of data.

4.2.1 Presentation

The Multi-Layer Perceptron (MLP) network is one of the most popular and successful neural network architectures in practical applications. Its principal applications are pattern discrimination and classification, prediction and forecasting, regression and control. The strength of MLPs lies in their ability to recognise complex patterns in real-world data which is noisy or uncertain.

An MLP is a feed-forward network so that it contains no feedback loops. It has one or more hidden layers for which the combination function is the inner product of the inputs and weights, plus a bias. The activation function is usually a logistic or 'tanh' function. Each layer uses a linear combination function. The inputs are fully connected to the first hidden layer, each hidden layer is fully connected to the next, and the last hidden layer is fully connected to the outputs.

Thus in mathematical terms, it means that the output of the jth hidden unit is obtained first by forming a weighted linear combination of the d input values:

$$a_j = \sum_{i=0}^{d} w_{ij}^{(1)} x_i \tag{4.1}$$

where[1]:

$w_{ij}^{(1)}$ denotes a weight in the first layer going from input i to hidden unit j,

x_i denotes the ith input value.

The activation of hidden unit j is then obtained by transforming the linear sum in equation 4.1 using an activation function g to give:

$$z_j = g(a_j) \tag{4.2}$$

In this project, we dealt with a two-class problem, so we just needed a single output unit.

[1]Note that the bias is included in the sum as x_0.

4.2.2 Architecture

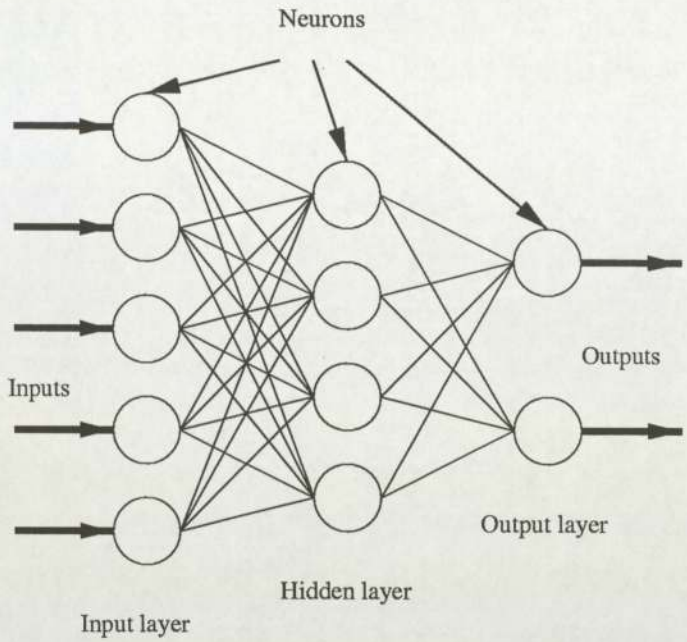

Figure 4.1: An example of an MLP having 5 inputs, 4 hidden units, 2 output units and 2 layers of adaptive weights. Note that the bias parameters are not shown.

The first step when using an MLP is to choose its architecture (see figure 4.1). It should be an architecture which gives good generalisation performance for a given set of input data. A two-layer MLP with sigmoid activation function can approximate any continuous function to arbitrary accuracy if enough hidden units are used. It therefore reduces the problem so that we only need to choose the number of hidden units. This has to be chosen carefully to avoid over-training and to ensure that the model is flexible enough to capture the underlying relationships in the data.

The number of inputs I is determined in our problem by the number of autoregressive coefficients. The number of outputs K is equal to 1 as we have a 2-class problem. The procedure to determine the number of hidden units J is explained in the next subsection.

4.2.3 Multi-layer Perceptron training

We want to find the network with the best generalization performance. The simplest approach to reach that goal is to compare different networks by evaluating their error functions using data which is separate from that used for training. Various networks are trained by minimisation of an appropriate error function defined with respect to a training set. The performance of the networks is then compared by evaluating the error function using an independent validation set. This method is called the hold out approach. However, such a method can lead to some over-fitting to the training set. Therefore the performance of the selected network should be confirmed by measuring its performance on a third independent set of data called the training data set. The entropy error measure was monitored so as to determine when to stop the training. This allows us to interpret the network output as $P(C|x)$.

To find a local minimum, the optimisation algorithm used was scaled conjugate gradients (SCG). Several tries were made with the conjugate gradient and the quasi-Newton method but results were not as good as with SCG.

4.3 Radial Basis Function networks

Firstly, a brief presentation of the Radial Basis Function network (RBF) is given in this section. Then, it deals with its architecture and its training procedure.

4.3.1 Presentation

RBF networks usually have only one hidden layer for which the combination function is based on the Euclidean distance between the input vector and the weight vector. RBF networks do not have anything that is exactly the same as the bias term in an MLP, but somes types of RBFs have a 'width' associated with each hidden unit or with the entire hidden layer. Instead of adding it in the combination function like a bias, you divide the Euclidean distance by the width.

4.3.2 Architecture

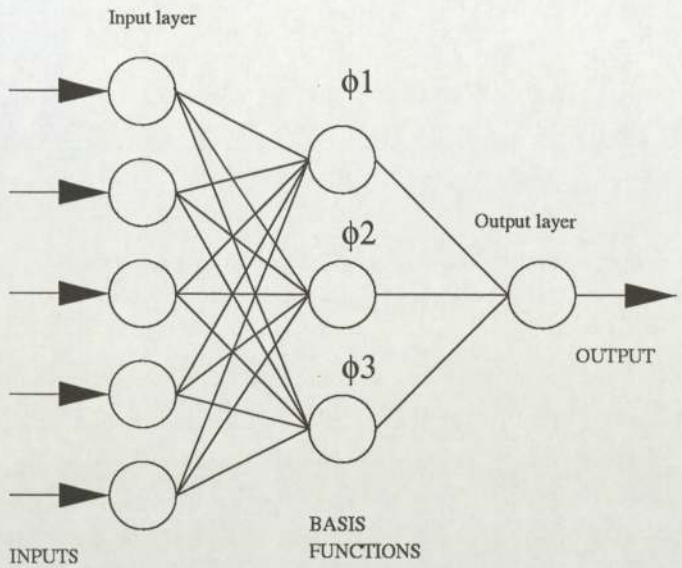

Figure 4.2: An example of an RBF network having 5 inputs, 3 basis functions (ϕ) which act like hidden units, 1 output unit and 2 layers of adaptive weights.

Figure 4.2 shows the architecture of a RBF. Any RBF network with one hidden layer can approximate any continuous function to arbitrary accuracy provided there are enough hidden units [Bishop, 1995]. The size of an RBF network is determined by the number of centres in the hidden layer which is explained in the next subsection.

4.3.3 RBF network training

Several experiments were carried out to determine the optimal number of centres. The data was partitioned into two data sets:

- training set,
- test set.

The sizes of the sets were determined using a similar procedure that the one we used for MLPs. The misclassification error rate was monitored in order to determine the optimal network.

4.4 General results

In this part, we give the results we obtained with MLPs and RBFs. Each time a comparison between the different options used during the pre-processing step has been done.

4.4.1 Results using MLPs

The best results have been obtained with MLPs using the logistic sigmoid activation function:

$$g(a) = \frac{1}{1+e^{-a}} \tag{4.3}$$

As we wanted balanced data sets and we had a set of 457 data with 209 spurious results and 248 defects (see table 3.2), the size of the three sets was defined by the equation 4.4:

$$Size = 3 * E[(min(S, D))/3] \tag{4.4}$$

where:

S is the number of spurious results in the original data set,

D is the number of defects in the original data set,

$min(A, B)$ is the smallest element between A and B,

$E[A]$ rounds the element A to the nearest integer towards minus infinity,

$Size$ is the size of the three sets (training, validation and test).

The numerical computation gave:

$$Size = 3 * E[(min(209, 248))/3] = 207 \tag{4.5}$$

Therefore, 207 spurious results and 207 defects were used for these experiments and were separated between the three data sets. Thus, each data set contained 138 data (69 defects and 69 spurious results) and only 414 selected data out of 457 were used.

150 MLPs were trained in order to find the optimal network. All had 8 input units, which was the number of autoregressive coefficients and one output for the two classes. J, the number of hidden units was varied between 1 and 15. Ten training runs for each architecture were made. For each MLP the optimisation algorithm required between 87 and 275 passes through the training set to reach the lowest value of the error function. Table 4.1 shows the misclassification error rate for the two classes using either the middle sensor method or the biggest dip method for the selection of the most representative sensor of a box.

	Misclassification error rate (%)	
Number of hidden units	Biggest Dip	Middle Sensor
1	10.98	11.68
2	10.66	11.57
3	11.06	11.42
4	10.18	11.40
5	10.54	11.43
6	10.48	11.37
7	10.32	10.97
8	10.62	11.66
9	10.69	11.54
10	10.40	11.49
11	10.69	11.25
12	10.69	11.55
13	10.69	11.60
14	10.69	11.69
15	10.88	11.85

Table 4.1: Misclassification error rate given by MLPs classifying defects and using the hold-out method averaged over 10 different random sets.

Table 4.1 shows that the number of hidden neurons has very little influence on the results. Thus we do not obtain a classical minimum for the error function. It takes values around 11% with a minimum of 10.18% reached for 4 hidden neurons using the biggest dip method.

4.4.2 Results using RBF networks

Here the best results have been obtained using the thin plate spline activation function which is defined by:

$$tps(r) = r^2 log(r) \tag{4.6}$$

where r is the squared distance to the centers of the RBF.

190 RBF networks were trained in order to find the optimal network. All had 8 input units and one input unit for the two classes. We chose an upper limit of 20 hidden units. For each of these configurations 10 partitions with an equal number of randomly selected patterns (208) from both classes were used. The misclassification error rate was assessed for each partition and the average of these minimum was worked out.

	Misclassification error rate (%)	
Number of hidden units	Biggest Dip	Middle Sensor
2	14.66	14.97
3	11.80	12.42
4	11.65	12.40
5	11.62	12.43
6	11.57	12.37
7	11.45	11.97
8	11.33	11.66
9	11.25	11.54
10	11.05	11.49
11	10.99	11.25
12	10.97	11.40
13	11.12	11.50
14	11.20	11.59
15	11.24	11.65
16	11.25	11.69
17	11.30	11.75
18	11.32	11.85
19	11.50	11.90
20	11.51	11.99

Table 4.2: Misclassification error rate given by RBF networks classifying defects using the thin plate activation function in the hidden units layer.

Table 4.2 shows the misclassification error rate for the two classes using either the middle sensor method or the biggest dip method for the selection of the most representative sensor of a box. It shows that once again the number of hidden neurons has very little influence on the results. It takes values around 11% with a minimum of 10.97% for 12 centres using the biggest dip method.

4.4.3 Reject Option

We previously saw that the output of the RBF and MLP is the probability that the input pattern is a defect. It is a value between 0 and 1. The intent of the reject option is not to classify the results close to 0.5. Basically, we know that an output close to 0.5 means that the corresponding input pattern is close to the decision boundary. This leads to uncertainty for its classification and therefore it is much better to treat separately this kind of pattern. For example, if the reject threshold value is 0.1, then the outputs:

- are classified as 0 : spurious results (S) if lower than 0.4,
- are classified as 1 : defects (D) if greater than 0.6,
- are rejected if between 0.4 and 0.6.

This option makes it possible to have a better classification error rate with both neural network classifiers. On the other hand, it means more work for the user as it would be necessary for them to look at 'rejected' patterns. It is obvious that the larger the reject threshold is, the higher the number of non-classified input patterns. Therefore, it is important to choose a reject threshold as small as possible in order to avoid a large number of non-classified input patterns. However, this option allows an improvement in the classification error rate so it does have benefits.

Here is an example of the effectiveness of the reject option. It was assessed with an MLP with 8 input units, 4 hidden units and 1 output unit. Different thresholds ranged

from 0 (no reject option) to 0.15 were tested. The method used for the selection of the 'best sensor' was the biggest dip method and 4 scans were ignored at both sides of boxes.

The misclassification error rate without using the reject option was 10.87% (15 input patterns). First of all, table 4.3 shows the set of non-classified results with a 0.15 reject threshold. Table 4.4 shows the results for different reject thresholds.

File	Box ID	Net result	True type	Misclassified
23	1	0.464232	1	Y
33	22	0.522766	1	N
33	120	0.561292	1	N
41	1	0.367248	1	Y
41	2	0.593867	1	N
44	11	0.478263	0	N
44	7	0.386722	1	Y

Table 4.3: 7 results were non-classified. Among them, 3 input patterns would have been misclassified and 4 classified correctly.

Thresholds	Nb of classified results	Nb of non-classified results	Misclassification error rate
0	138	0	10.87
0.01	138	0	10.87
0.02	138	0	10.87
0.03	136	2	11.03
0.04	135	3	10.37
0.05	135	3	10.37
0.06	135	3	10.37
0.07	134	4	10.45
0.08	134	4	10.45
0.09	134	4	10.45
0.10	133	5	10.53
0.15	131	7	9.17

Table 4.4: Misclassification error rate vs reject thresholds.

The best value of the reject threshold for this example was 0.04 (it has the best trade-off), and the improvement on the misclassification error rate was approximately

0.5%. Only a few examples (7 out of 138) were close to the decision boundary.

4.5 Conclusions

Table 4.5 shows that RBF networks and MLPs produce similar levels of generalisation, although MLPs tend to be a little better. The main advantage of the RBF networks is a shorter training time in comparison with the MLPs, so we managed to run more experiments quickly. Finally, we dealt with the reject option which leads to another improvement of the error rate.

Network	Pre-processing method	Minimum error rate	Number of hidden units
MLP	Biggest dip	10.18	4
MLP	Middle sensor	10.97	7
RBF	Biggest dip	10.97	12
RBF	Middle sensor	11.25	11

Table 4.5: Minimum error rates given by networks

Chapter 5

Conclusions

The first part of this project was concerned with data analysis and data visualisation. We initially saw from histograms of the moments of low order and PCA, that there was not a clear separation between the spurious results and the defects.

After we analysed and visualised the data, we were able to decide on the form of the pre-processing procedure. Thus, we applied several tools to the raw data whose composition before the pre-processing is shown in appendix A. One of them was the selection of the most representative sensor from each box. The goal of this was to reduce the computational demands. Two methods were investigated. The first was the selection of the middle sensor and the second was the selection of the sensor with the biggest dip. A comparison between these two methods showed that the best results were obtained with the selection of the sensor with the biggest dip. However, during this stage a lot of information is discarded about the shape of the defect since a single sensor is taken. As this is one of the key points of the success of this application, further work on this area could be very useful.

Next, two neural networks were used to classify the data. The optimal architecture was assessed as an MLP with 4 hidden units using the biggest dip method. The level of generalisation performance for such an MLP was assessed as about 10.18%.

A comparison between both classifiers showed that the optimal MLP tends to outperform the optimal RBF network by a small amount. The difference between them is approximately 1% in the misclassification error rate. This is representative of the fact that in many applications MLPs give a slightly improved classification error in comparison with RBF networks.

Up to now, there is no evidence that using any of the power of MLPs or RBF networks actually increases performance above that which could be obtained with classical statistical analysis (for example, logistic regression). The feature extraction chosen was much more successful than the simple moments and PCA would have been and in particular, generalised across sections of pipeline much better.

Two directions may be interesting for future work. The first one is the selection of the boxes from the .era and .els files to the .dat and .sel files. They are hand selected so far. Finding an algorithm could be very efficient since the pre-processing is based on data from the .dat files and target data comes from the .sel files. The other one is the selection of the most representative sensor from each box as mentioned earlier.

Appendix A

Data used

The data set used for this project (.sel and .dat files) was composed of 7249 sensors. Each of theses contains the measure of 51 ultrasound scans. The sensors were distributed into 457 boxes which were either spurious results (209 boxes) or real defects (248 boxes).

The data were distributed into 50 files:

mt2891_15.dat and mt2891_15.sel,

mt2891_20.dat and mt2891_20.sel,

mt2891_22.dat and mt2891_22.sel,

mt2891_23.dat and mt2891_23.sel,

mt2891_26.dat and mt2891_26.sel,

mt2891_27.dat and mt2891_27.sel,

mt2891_28.dat and mt2891_28.sel,

mt2891_29.dat and mt2891_29.sel,

mt2891_30.dat and mt2891_30.sel,

mt2891_31.dat and mt2891_31.sel,

mt2891_33.dat and mt2891_33.sel,

mt2891_39.dat and mt2891_39.sel,

mt2891_40.dat and mt2891_40.sel,

mt2891_41.dat and mt2891_41.sel,

mt2891_42.dat and mt2891_42.sel,

mt2891_44.dat and mt2891_44.sel,

mt2891_45.dat and mt2891_45.sel,

mt2891_46.dat and mt2891_46.sel,

mt2891_47.dat and mt2891_47.sel,

mt2891_48.dat and mt2891_48.sel,

mt2891_49.dat and mt2891_49.sel,

mt2891_50.dat and mt2891_50.sel,

mt2891_51.dat and mt2891_51.sel,

mt2891_52.dat and mt2891_52.sel,

mt2891_53.dat and mt2891_53.sel.

Appendix B

Moments

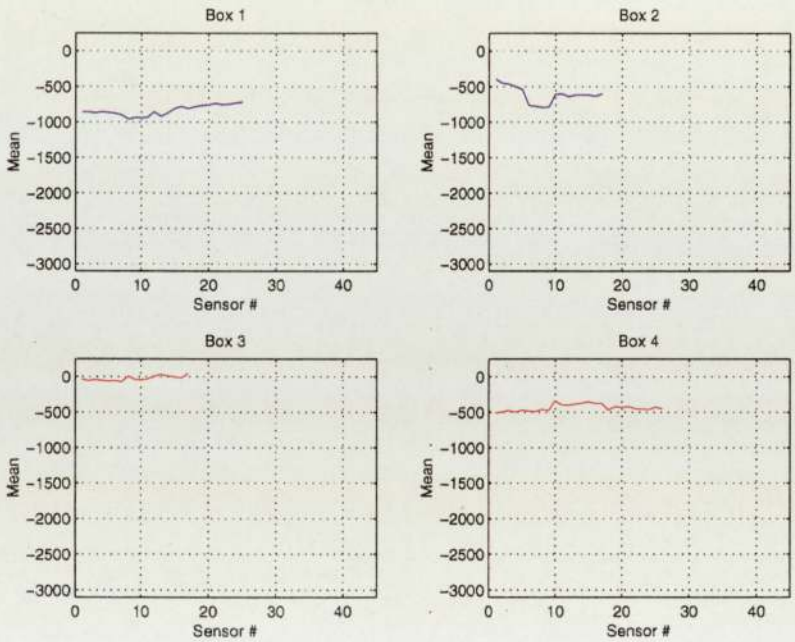

Figure B.1: Mean using the four first boxes on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

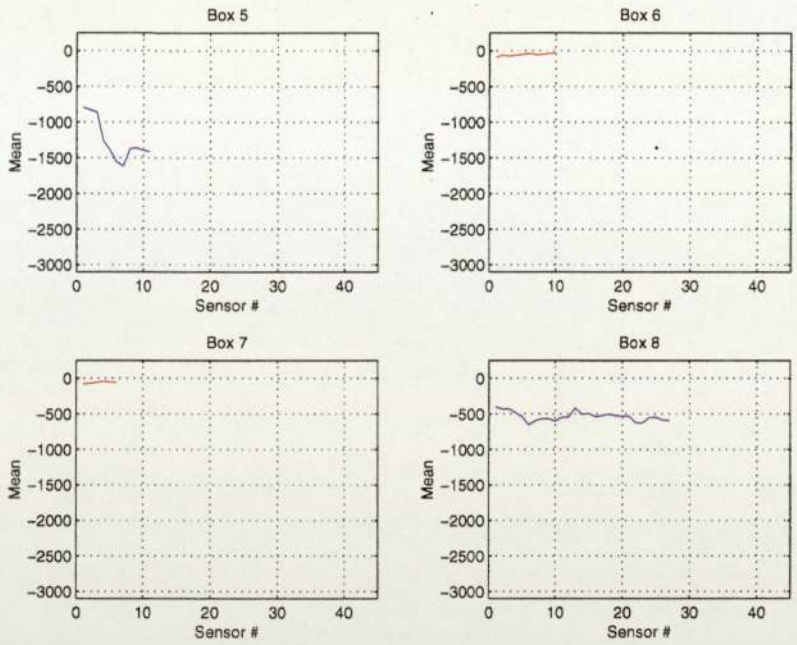

Figure B.2: Mean using boxes 5 to 8 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

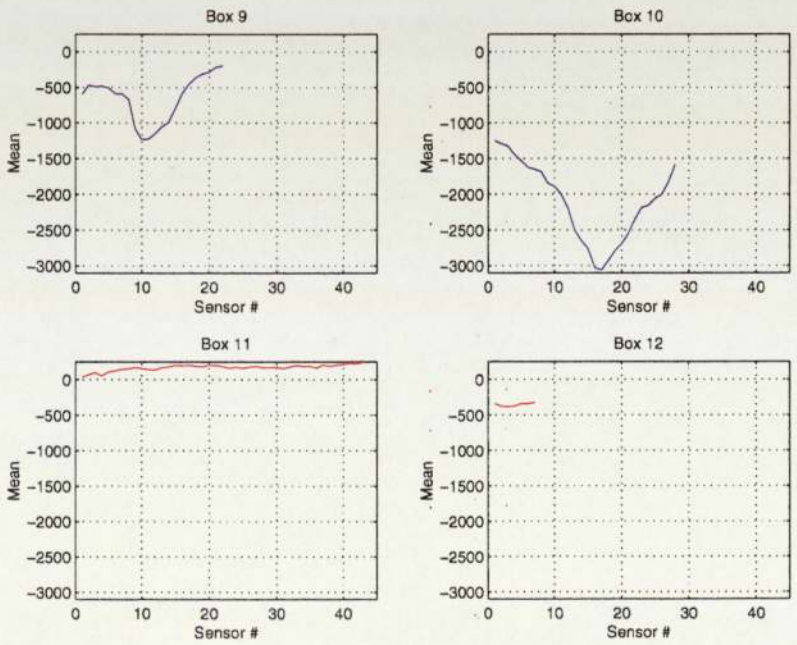

Figure B.3: Mean using boxes 9 to 12 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

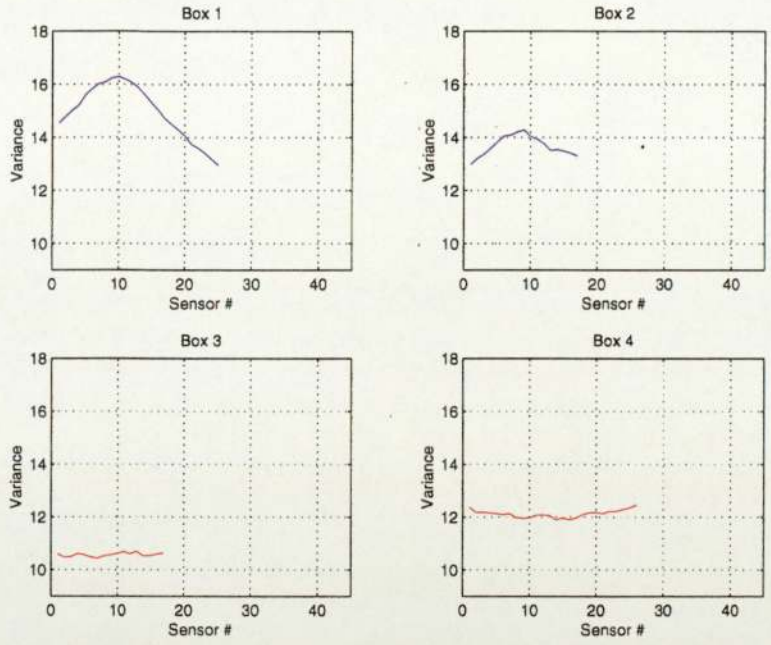

Figure B.4: Log(Var) using the four first boxes on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

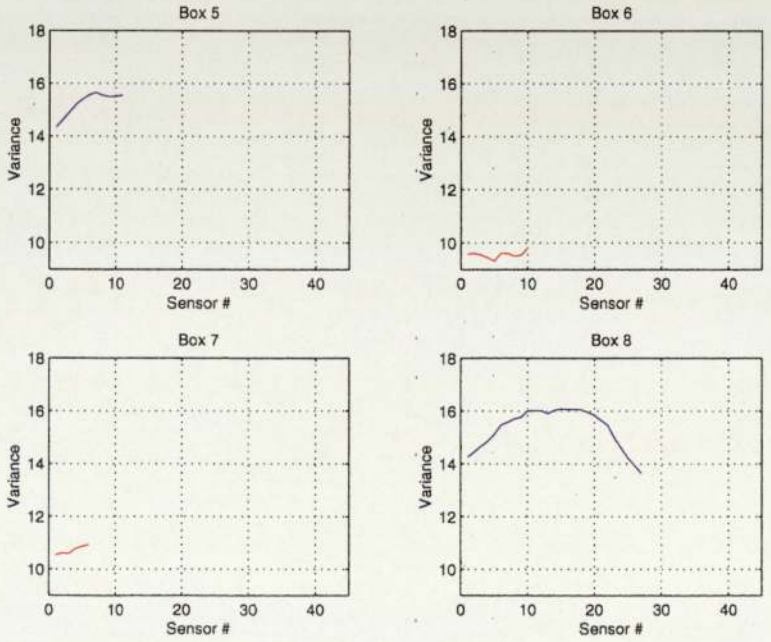

Figure B.5: Log(Var) using boxes 5 to 8 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

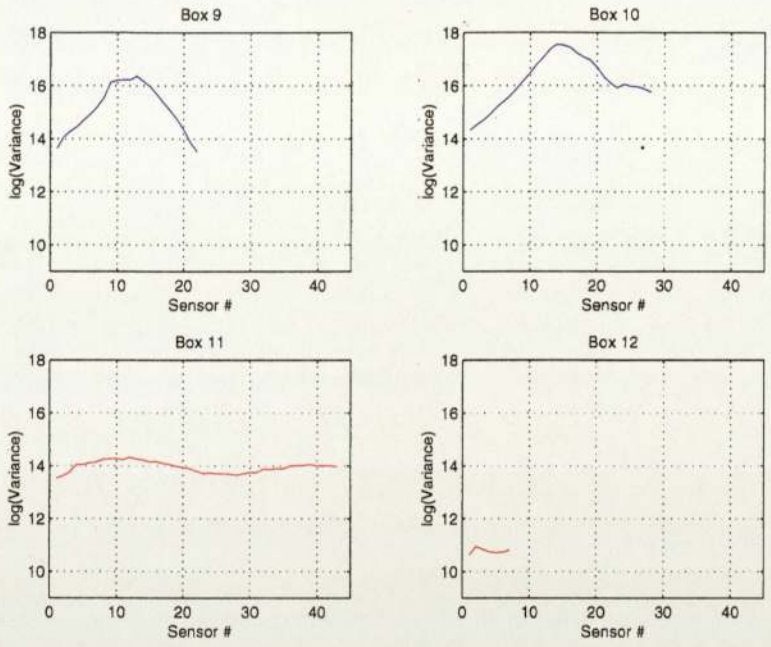

Figure B.6: Log(Var) using boxes 9 to 12 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

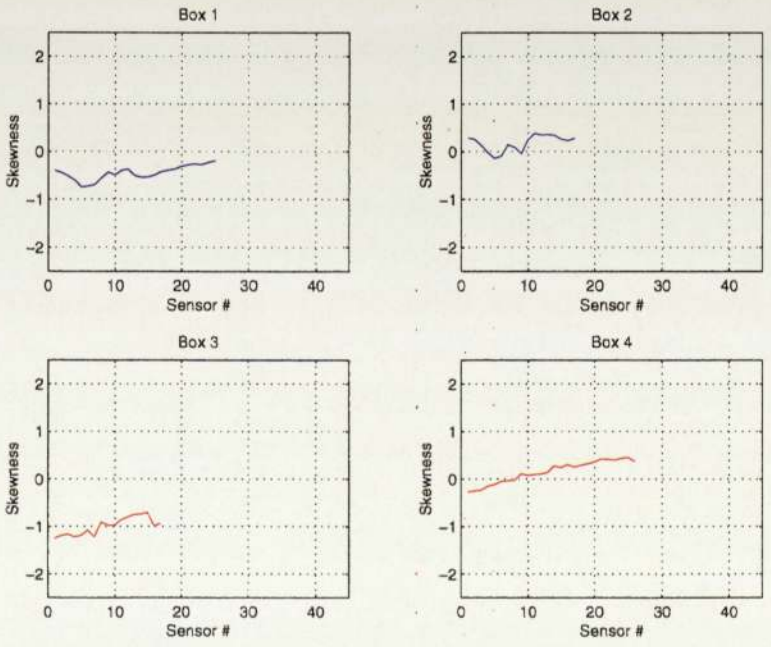

Figure B.7: Skewness using the four first boxes on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

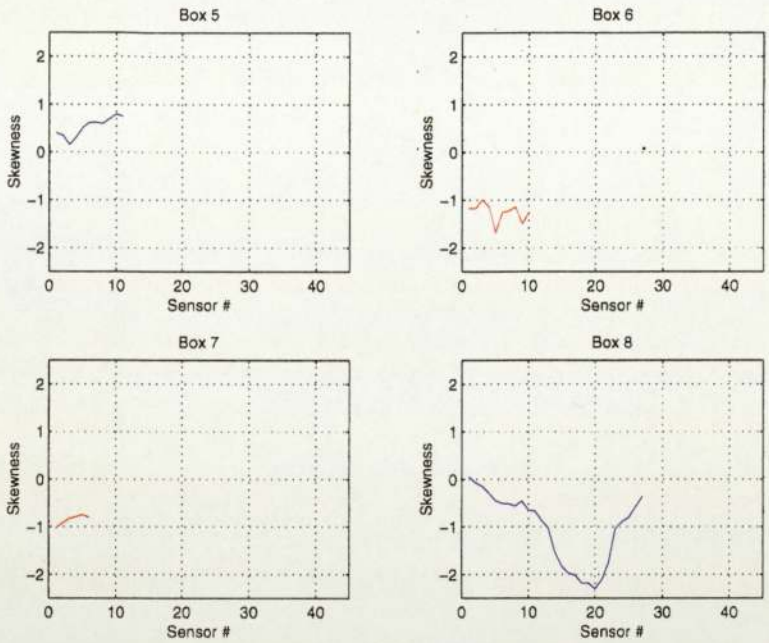

Figure B.8: Skewness using boxes 5 to 8 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

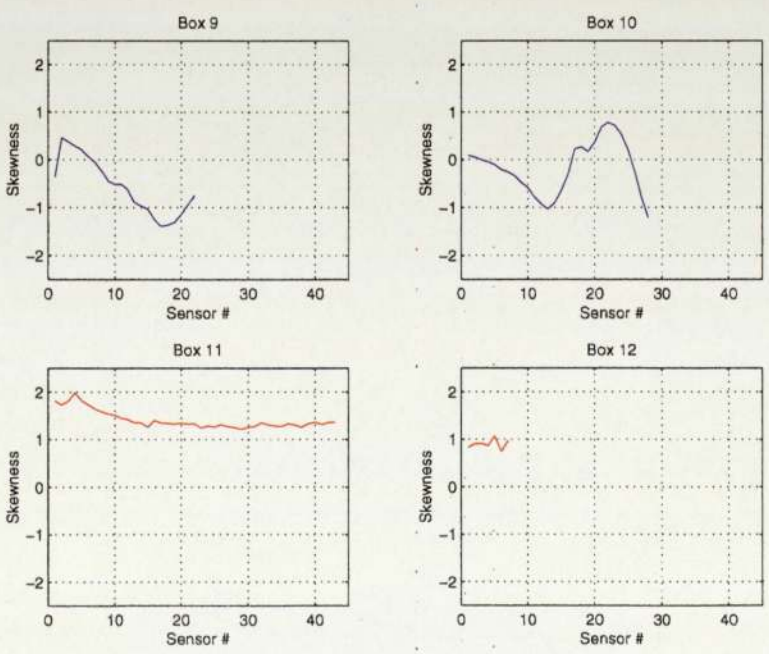

Figure B.9: Skewness using boxes 9 to 12 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

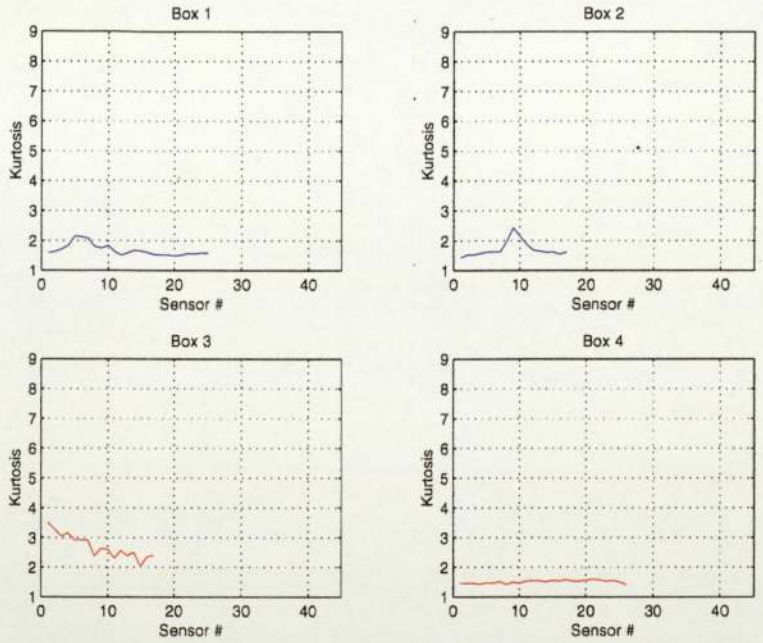

Figure B.10: Kurtosis using the four first boxes on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

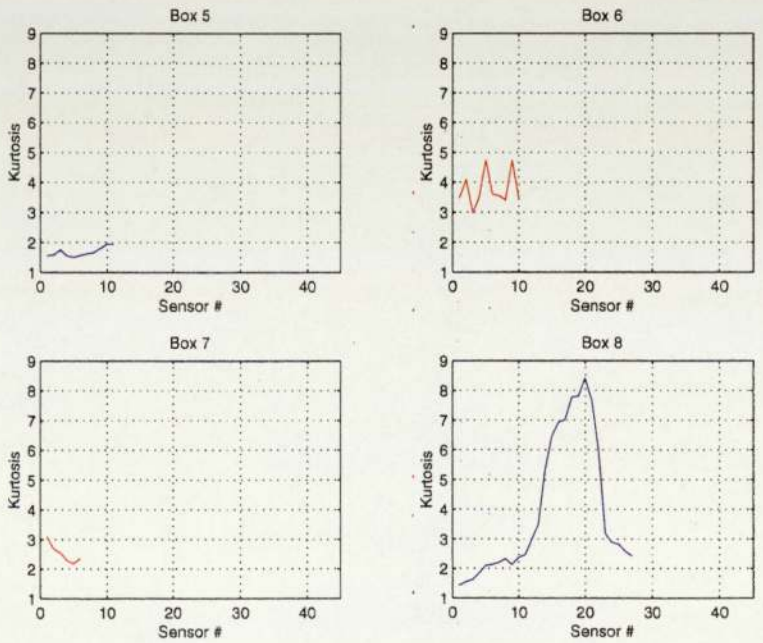

Figure B.11: Kurtosis using boxes 5 to 8 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

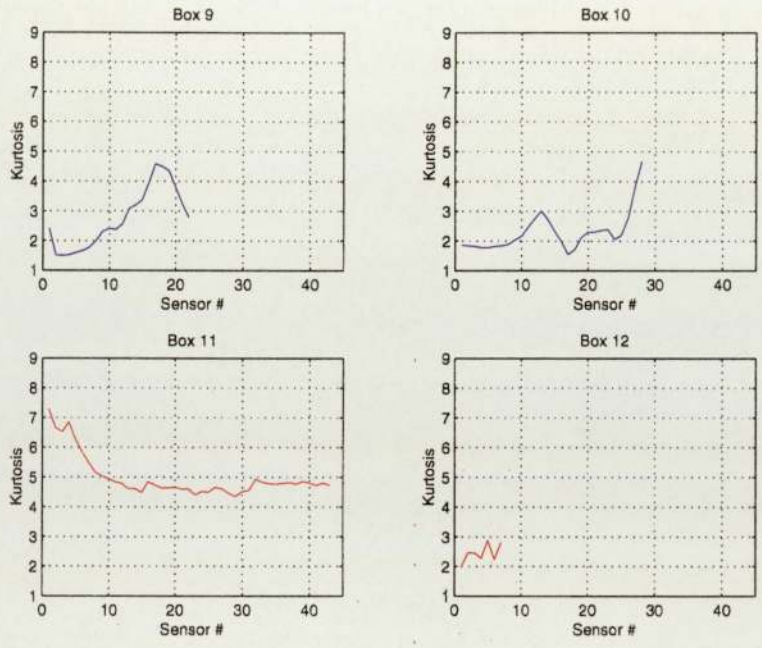

Figure B.12: Kurtosis using boxes 9 to 12 on the mt2891_50.sel and mt2891_50.sel files. Spurious results are plotted in red and defects in blue.

Appendix C

Code

Note that the files Nb_of_type2.m, Round.m, Indix.m, PrintNetResults2.m, Round-off.m, SelectBiggestDip.m, SelectMidsensor.m, Local2.m, ReadDatFile.m, center.m Merge-Double.m, ReadSelFile.m and compare2.m are not included in this appendix as they are not algorithmically important.

C.1 ZeroPhase.m

```
function AlignedScanBox = ZeroPhase(BestSensors);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ZEROPHASE aligns scan / box.                                                    %
%                                                                                 %
% Description :                                                                   %
%      - double size by reflecting the signal                                     %
%      - Fourier transform                                                        %
%      - take the absolute value                                                  %
%      - inverse Fourier transform                                                %
%      - round off                                                                %
%      - take first 48 samples / scan                                             %
%                                                                                 %
% If PLOTTRANSFO == 1 then plot the result                                        %
%                                                                                 %
% SEE ALSO Roundoff.                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% declare global variables
```

```
global PLOTTRANSFO NBCOLUMNSP2


% double in size by padding with NBCOLUMNSP2 zeros/scan
%for i=1:size(BestSensors)
%  BestSensors(i, NBCOLUMNSP2+1:2*NBCOLUMNSP2) = 0;
%end


%  double in size by reflecting the signal
for i=1:size(BestSensors)
  for j=1:NBCOLUMNSP2
    BestSensors(i, 2*NBCOLUMNSP2+1-j) = BestSensors(i,j);
  end % for
end % for


% Fourier transform
FourierTransform = fft(BestSensors');

% take the absolute value and
% do the Inverse Fourier transform
InvFourier = ifft(abs(FourierTransform));


% do a round off
% Indeed, after the ift we want a *real* signal. And
% depending on the computer rounding is possible after the fft and
% ift, which gives a small imaginary component to the ift.
% the Roundoff() function just rounds off small values to zero.
RealInvFourier = Roundoff(InvFourier);


% Take first 48 samples/scan
AlignedScanBox(1:NBCOLUMNSP2, :) = RealInvFourier(1:NBCOLUMNSP2, :);

% transpose the matrix
AlignedScanBox = AlignedScanBox';
```

C.2 FindCoeffARmodel.m

```
function CoeffARmodel = FindCoeffARmodel(AlignedCenteredScanBox, ARorder, ..
    indrho0);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FINDCOEFFARMODEL finds the coefficients of an AR model whose order %
%                  is ARorder using the Yule-Walker equations        %
%                  and only one sensor per box.                      %
%                                                                    %
% Description :                                                      %
%     - alignedCenteredScanBox is the original data.                 %
%     - the output is a matrix whose size is nbboxes * ARorder.      %
%       Each row corresponds to a box and contains the Arorder AR    %
%       coefficients.                                                %
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% calculate nb boxes * ARorder autocorrelation coeff
for i=1:size(AlignedCenteredScanBox, 1)
  % calculate the auto-correlation coefficients for one sensor
  ACcoeff = xcorr(AlignedCenteredScanBox(i, :), 'coeff');
  % keep only rho(1) up to rho(ARorder) for the current sensor
  rho(i,:) = ACcoeff(1, indrho0 + 1:indrho0 + ARorder);
end % for


% Now, calculate the AR coeff with the Yule-Walker equations
% using the auto-correlation coefficients (Example of oreder 8)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  Yule-Walker equations                           %
%                                                                  %
% (rho(1))     (  1     rho(1)  rho(2)   ...  rho(7)) (a1 )        %
% (  .    )    (rho(1)    1     rho(1)   ...  rho(6)) (a2 )        %
% (  .    )    (rho(2)  rho(1)    1      ...     .  ) ( . )        %
% (  .    )    (  .       .       .      ...     .  ) ( . )        %
% (  .    )  = (  .       .       .      ...     .  ) ( . )        %
% (  .    )    (  .       .       .      ...     .  ) ( . )        %
% (  .    )    (  .       .       .      ...     .  ) ( . )        %
% (  .    )    (rho(6)  rho(5)  rho(4)   ...  rho(1)) ( . )        %
% (rho(8))     (rho(7)  rho(6)  rho(5)   ...     1  ) (a8)         %
%                                                                  %
% where ai are the AR coefficients                                 %
%   and rho(i) the autocorrelation coefficients                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% for each 'best sensor' from a box
for k=1:size(AlignedCenteredScanBox, 1)
```

```
  % init
  % A is the squared matrix used in the Yule-Walker equations
  A = diag(ones(ARorder,1));

  % fill the squared matrix diagonal by diagonal using symmetry
  for i=1:ARorder-1
    rhotemp = repmat(rho(k,i), ARorder-i, 1);
    A = A + diag(rhotemp,i) + diag(rhotemp, -i);
  end % for

  % Calculate the 16 AR coefficients with Yule-Walker equations
  % for each middle sensor
  a(:, k) = inv(A) * rho(k, :)';

end % for

% return the results
CoeffARmodel = a';
```

Bibliography

[Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford Univ. Press.

[Kendall and Ord, 1990] Kendall, S. M. and Ord, J. (1990). *Time Series, third edition.* Arnold.

[Miller and Ruppert, 1986] Miller, R. and Ruppert, D. (1986). *Beyond ANOVA : the Basics of Applied Statistics.* Wiley, New-York.

[Neale and Associates, 1979] Neale, M. and Associates (1979). *A Guide to the Condition Monitoring of Machinery.* Oyez Press Limited.

[R. Rohwer and White, 1996] R. Rohwer, A. W. and White, K. (1996). *Classification of Metal Loss regions in Pipelines using Neural Networks.*

[Saporta, 1990] Saporta, G. (1990). *Pobabilits Analyse des Donnes et Statistique.* Technip, Paris, France.

[Spencer et al., 1977] Spencer, A., Parker, D., Berry, D., England, A., Faulkner, T., Green, W., Holden, J., Middleton, D., and Rogers, T. (1977). *Engineering Mathematics, Volume 1.* Van Nostrand Reinhold Company Ltd.