A STRUCTURED APPROACH TO CLUSTER LOCATION IN HIGH DIMENSIONAL FEATURE SPACES.

A thesis

submitted to the

University of London

for the degree of Doctor of Philosophy

by

PHILIP JAMES NAYLOR

Department of Physics

King's College London

September 1992

Resubmitted – June 1994

Abstract

After a survey of existing cluster analysis algorithms, a new technique is introduced which seeks to break the problem down into a series of, better defined, sub-goals.

Possible methods for attaining these sub-goals are described, and discussed, and then the technique is applied to the segmentation of multi-spectral imagery, via the location of clusters in cooccurrence matrices.

Finally, the performance of the new technique is compared with that of some existing algorithms, and conclusions are drawn as to its usefulness, both in terms of speed and the quality of its results.

Contents

Figures

Tables

Previous Study & Current Work

The author obtained an Honours degree in Physics with Astrophysics from King's College London in 1987.

He then studied for an M.Sc. in Radio Astronomy at the Nuffield Radio Astronomy Laboratories, University of Manchester. This he obtained in 1988.

In October, of the same year, he returned to King's College and registered as a Ph.D. student in the Physics department.

The work for that Ph.D. culminated in the original publication of this thesis in 1992.

After leaving King's, the author eventually found employment as an assistant computer officer in the Faculty of Computer Studies and Mathematics at the University of the West of England, in Bristol. He has found it hard enough to make time to revise this thesis for resubmission, let alone do any new work in the fields of image processing or pattern recognition.

Declaration

All of the work described in this thesis was performed by the author.

With the exception of the general idea of an adaptive implementation of the CLEAN deconvolution algorithm, no portion of this thesis has been submitted in support of an application for another degree, or qualification, of this, or any other, university, or other institute of learning.

Philip J. Naylor,

29th September 1992.

Revised & resubmitted :

27th June 1994.

Department of Physics,

King's College,

London.

Acknowledgements

I wish to thank Professor R.E. Burge and Dr. M. McCabe for making available the research facilities of the Physics department at King's College and the IT unit at BP Research, respectively. And, I gratefully acknowledge receipt of an SERC 'Case' studentship, co-sponsored by British Petroleum.

I wish to thank my supervisor, Dr. J.F. Boyce, for all his help and guidance over the years.

The airborne multi-spectral imagery, of Blewbury and Churn Farm, was acquired by the Natural Environment Research Council (NERC). The reference data planes were compiled at the NERC Unit for Thematic Information Systems. My thanks to Dr. J.J. Settle, of NUTIS, for making these available.

I also wish to thank Jimmy, for many an amusing anecdote and tolerating my working methods; Margaret, and the rest of her group, for their hospitality at Sunbury; and Professor Burge for his patience whilst I was writing up.

Thanks relating to the revision and resubmission of this thesis go to Professor Ken Jukes, Graeme Matthews, and Julia Dawson, of the Faculty of Computer Studies & Mathematics at the University of the West of England (Bristol) for allowing me the time, and facilities, to get the work done.

Thanks go, as well, (in no particular order) to :

- the image analysis group - Simon, Simon, Simon, Darren, & Chris - for keeping me amused,
- the ladies of Commonwealth Hall - Caitriona, Suraya, Magda, & Sarah - for keeping me sane,
- and anyone I've ever shared an office with (131, 126, or Q135) - in particular Shah for the use of 'phsun' after my departure.

Finally, thanks to Ruth – for cups of tea, and walks in the country.

When work is done as sacred work, unselfishly, with a peaceful mind, without lust or hate, with no desire for reward, then the work is pure.

But when work is done with selfish desire, or feeling it is an effort, or thinking it is a sacrifice, then the work is impure.

And that work which is done with a confused mind, without considering what may follow, or one's own powers, or the harm done to others, or one's own loss, is the work of darkness.

Bhagavad Gita 18, 23–25.

Scope of this Work

The original aim of this work was to produce an algorithm which would take a multi-spectral satellite image of an arid area and, not only differentiate between but also, identify the different rock types within the scene (limestone, granite, etc.). This would be done on the basis of the different colours and textures in the image. Once identified, relationships between the various rock structures, plus any other relevant information, would be used to locate areas which might contain petrochemical deposits.

The basis of the work was to be a cooccurrence matrix based segmentation algorithm which had already been implemented for use with black & white (mono-spectral) images. However, the author's belief that any pattern recognition algorithm worth the name should not require any more information than that which is contained in the data meant that the extension from one spectral band to many was far from simple. In fact it required a solution to the 'cluster validation problem' (i.e. the removal of the need to tell the algorithm how many distinct clusters of data points there are in a data set).

The solution of this problem took up most of the two years for which funding was available. Although, towards the end, some work was done on texture analysis, this never reached a stage where it might be satisfactorily included in this volume. Thus the work, herein described, centres on the solution of the cluster validation problem, and the application of the resulting algorithm to the segmentation of images, on the basis of colour only.

CHAPTER 1
Introduction to Cluster Analysis

An introduction is given to the concept of feature spaces, and cluster analysis. A review is made of existing cluster analysis techniques, and how they may be improved on. A new approach is proposed, which involves breaking the problem down into more manageable pieces.

1.1 Constructing a feature space :

Consider a set of objects, $\Omega = \{\omega\}$, each of which has a set of N quantifiable properties (features), $x(\omega) = (x_1(\omega), x_2(\omega), \ldots, x_N(\omega))_{\omega\in\Omega}$ associated with it. The $x(\omega)$ can be treated as N–dimensional vectors, i.e. :

$$\mathbf{x}(\omega) = x_1(\omega)\hat{\imath}_1 + x_2(\omega)\hat{\imath}_2 + \ldots + x_N(\omega)\hat{\imath}_N \tag{1.1}$$

where $\{\hat{\imath}_1, \hat{\imath}_2, \ldots, \hat{\imath}_N\}$ are the, ortho-normal, basis vectors of the coordinate system.

The $\mathbf{x}(\omega)$ are then referred to as 'feature vectors', which point to locations within a 'feature space'.

If Ω is actually a collection of sub-sets $\{\Omega_A, \Omega_B, \ldots\}$ each of which contains objects which are similar, in their properties, to other objects within the same sub-set, but different from the objects in all the other sub-sets, then the end points of the feature vectors (the data points) will form clusters in the feature space.

The probability of a feature vector lying in the range $\mathbf{x}$ to $\mathbf{x} + d\mathbf{x}$ is

$p(\mathbf{x})d\mathbf{x}$. By Bayes' theorem :

$$p(\mathbf{x}) = \sum_{\Omega_a \in \Omega} p(\Omega_a)\, p(\mathbf{x}|\Omega_a) \tag{1.2}$$

where $p(\Omega_a)$ is the probability of a data point belonging to cluster Ω_a; and $p(\mathbf{x}|\Omega_a)$ is the probability of a data point being at position $\mathbf{x}$ in the feature space, given that it belongs to cluster Ω_a.

Ideally the density distribution of the feature vectors will then be given by:

$$\rho(\mathbf{x}) = M\, p(\mathbf{x}) \tag{1.3}$$

$$\Rightarrow \rho(\mathbf{x}) = M \sum_{\Omega_a \in \Omega} p(\Omega_a)\, p(\mathbf{x}|\Omega_a) \tag{1.4}$$

where M is the total number of feature vectors.

This will only be exactly true in the limit as M tends to infinity. However, it will be approximately true provided that M is large enough for the data to form a reasonably representative sample of the probability distributions (see figure 1.1 for a one dimensional example). As M decreases, the approximation increasingly fails (figure 1.2). The discrepancy between the ideal and actual density distributions can be modeled as a noise term, so that $\rho(\mathbf{x})$ becomes :

$$\rho(\mathbf{x}) = [M \sum_{\Omega_a \in \Omega} p(\Omega_a)\, p(\mathbf{x}|\Omega_a)] \bullet n(\mathbf{x}) \tag{1.5}$$

where $\bullet$ is either multiplication or addition, depending on the noise model being used; and $n(\mathbf{x})$ takes random values, from an appropriate probability distribution.

1.2 The aims of cluster analysis :

There are two, not mutually exclusive, aims which we may have when analysing the clustering of data points in a feature space.

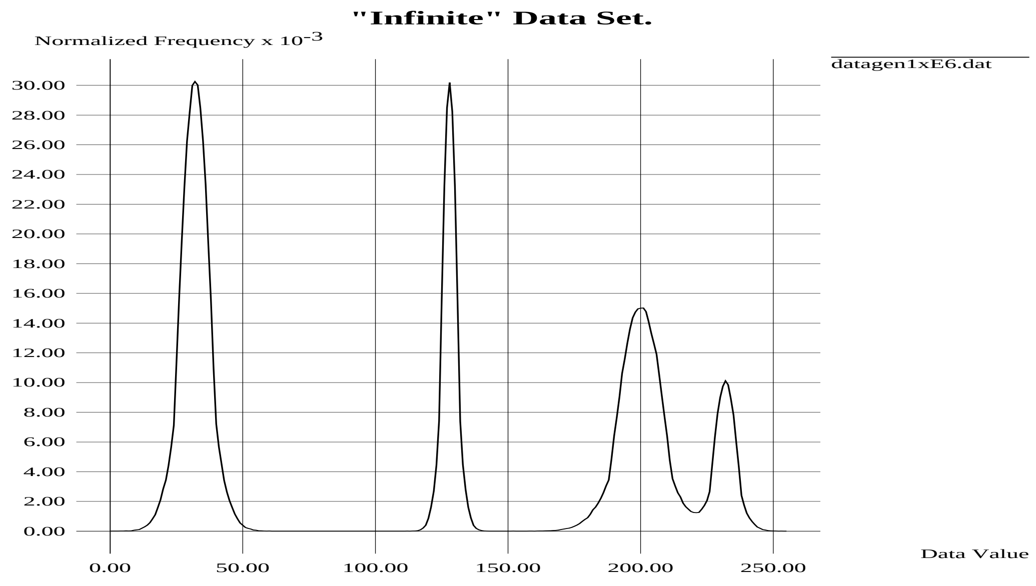


Figure 1.1: Density distribution of 10^6 data points.

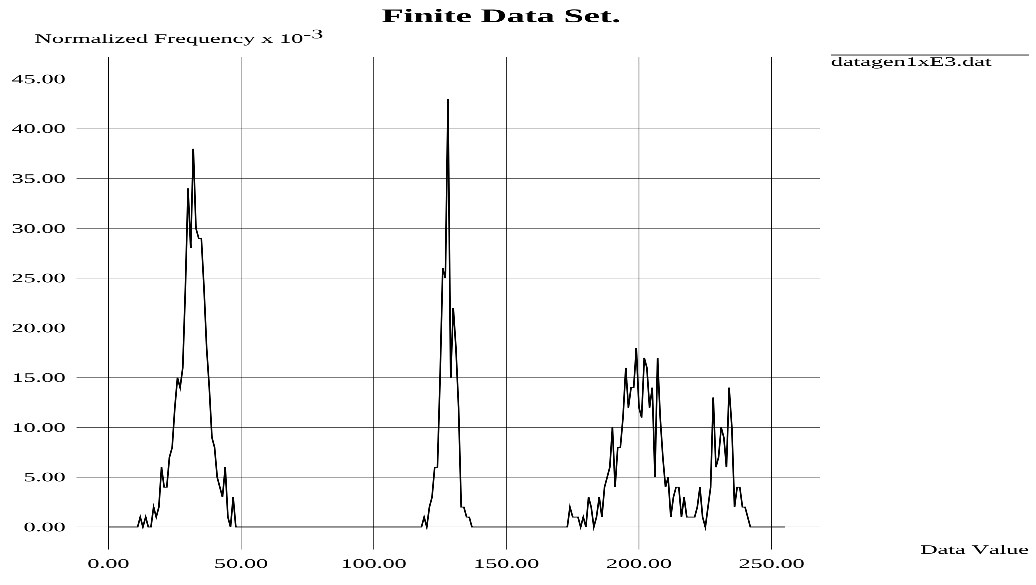


Figure 1.2: Density distribution of 10^3 data points.

The first is the characterisation of the dataset as a whole. This involves the extraction of parameters associated with the clusters. How many are there ? What are their means, variances, etc. ? The determination of parametric descriptions of lines and, or, curves from clusters in Hough space [Hough, 1962] is an example of this first aim.

The second aim is the classification of individual data points as belonging to one of several distinct groups (i.e. approximations to the sub-sets $\{\Omega_A, \Omega_B, \ldots\}$). This involves placing 'partitions', or 'decision boundaries', between each of the clusters. Each of the data points is then assigned a 'label', the value of which depends on which cluster it is deemed to be associated with. The segmentation of digitised images into regions of similar intensity, or colour, is an example of this second aim.

The two aims would be combined, for example, in the case of wanting to use the cluster parameters to characterise, or help identify the underlying nature of, the regions in a segmented image.

1.3 The problems associated with cluster analysis :

The first problem linked with cluster analysis is a perceptual one. The distribution of data points in a two dimensional feature space can be easily represented as a scatter plot, or a contoured, or colour coded, density plot. The dataset can thus be readily visualised by a human analyst. In three dimensions things become a little trickier, but by no means impossible. In four, or more, dimensions we are left at the mercy of a computer's (or, more correctly, a computer program's) 'perception' of the feature space.

Unfortunately, due to the limitations imposed on it by the amount of memory it possesses, a computer is no better equipped to 'visualise' multi–

dimensional feature spaces than a human is. As was stated above, the most natural way of visualising a feature space is as an N–dimensional density plot. However, if, for the sake of argument, each of the measured features could take any of 256 different values (the number of grey-levels in a typical digitised image), then, provided you could get away with only using a single byte to store the density of data points at a given location in the feature space, the amount of memory required to store various dimensions of feature space is :

$$1 \text{ dimension} \Rightarrow 256 \text{ bytes}$$

$$2 \text{ dimensions} \Rightarrow 64 \text{ Kbytes}$$

$$3 \text{ dimensions} \Rightarrow 16 \text{ Mbytes}$$

$$4 \text{ dimensions} \Rightarrow 4 \text{ Gbytes}$$

If we have access to a high performance workstation, then three dimensions is possible. Otherwise we are limited to two dimensions, or less.

Of course, much of the feature space will be empty, and so the dataset could be stored as a series of vectors to, non-empty, locations within the feature space, and the density of data points at that location. In the worst case, making the same assumptions as above, this will only require $M(N+1)$ bytes of memory. However, we have now lost the straight forward mapping between feature space coordinates and computer memory locations. Every time a density value is required, the list of non-empty locations must be searched through until the entry for the appropriate location is found. This is not only time consuming, but it also destroys any conception of the dataset as a whole. The dataset becomes a collection of disjoint pieces, and the cluster analysis problem becomes like an attempt to identify the picture on a jig-saw by looking at it one piece at a time (in no particular order).

This treatment of the cluster analysis problem as being the analysis of a set of vectors, rather than as a density distribution (quite often there is not even the attempt to parcel together vectors which are the same), has, in the past, led to a lot of very similar solutions. These are reviewed in the next section. It is the aim of this work to try and break away from this rather blinkered view and, hopefully, provide a new perspective on the problem.

1.4 Existing techniques :

Most existing cluster analysis techniques fall into two main groups : hierarchical algorithms, and partitioning algorithms [Anderberg, 1973; Murtagh, 1990]. Both approaches require measures for three quantities :

The **similarity** between two data points (or a data point and a cluster centre, or even between two cluster centres). This quantity will be positive, take a maximum value when the two objects are identical, and decrease as the objects increasingly differ. This could, for example, be based on the Euclidean distance between the two points in the feature space :

$$S_{\psi\omega} = S(\mathbf{x}_\psi, \mathbf{x}_\omega) = \frac{1}{1 + \{\sum_{i=1}^{N} [x_i(\psi) - x_i(\omega)]^2\}^{\frac{1}{2}}} \tag{1.6}$$

The **quality** of the partitioning (or 'clustering') of the data. This quantity will also be positive, it will take a maximum value when the approximations to the clusters,$\{\Omega_A', \Omega_B', \ldots\}$, are identical to the actual clusters, and it will decrease as more and more data points are assigned to the wrong cluster (or the number of Ω_a' does not match the number of Ω_a). This could be the sum of the squared errors :

$$Q = \frac{1}{1 + \sum_{\Omega_a'} \sum_{\omega \in \Omega_a'} \widehat{(\mathbf{x}(\omega) - \mathbf{m}_a)}(\mathbf{x}(\omega) - \mathbf{m}_a)} \tag{1.7}$$

where the $\mathbf{m}_a$ are the mean vectors of the Ω_a'.

The **location** of a cluster 'centre'. This could be either of the averages (mean or mode) of the feature vectors within a cluster, or it could be the centre of the bounding sphere of the cluster.

Actually, S and Q can only meaningfully take *relatively* high or low values. That is, we can only say that two points are more, or less, similar than two other points, and that one arrangement of the partitions is only better, or worse, than another. We cannot, sensibly, say that two points have a particular similarity, or that an arrangement of partitions results in a particular quality of partitioning.

Hierarchical algorithms are based on the idea of calculating the similarity between every possible pair of clusters, and using this information to decide whether two clusters should be combined, or a single cluster should be split in two [Anderberg, 1973].

Those which combine clusters are called 'agglomerative'. They operate as follows :

1) Assume that all the data points are in clusters of their own.

2) Combine the two clusters which are most similar.

3) Calculate the similarity between the new cluster and each of the others.

4) If more than the required number of clusters (see §1.5) remain, go to (2).

At stage (3) there are four possible methods by which the new similarities may be calculated. The similarity between the new cluster and one of the others can be :

i) the similarity between their mean vectors.

ii) the minimum of the similarity values S_{ik} and S_{jk}, where $\Omega_i^{'}$ and $\Omega_j^{'}$ are the clusters which were combined, and $\Omega_k^{'}$ is any other cluster. This method produces clusters which are single linkage graphs. That is, two clusters are linked when the similarity between their two most similar members (one from each cluster) is greater than the similarity between their most similar members (within each cluster).

iii) the maximum of S_{ij} and S_{jk}. This method produces clusters which are complete linkage graphs. That is, two clusters are linked when the similarity between their two most dissimilar members (one from each cluster) is greater than the similarity between their most dissimilar members (within each cluster).

iv) the average of S_{ij} and S_{jk}. This method produces clusters which are average linkage graphs. That is, somewhere between single linkage, and complete linkage.

The use of agglomerative hierarchical clustering algorithms is only practical for fairly small datasets. Not only because the maximum number of similarities to be stored is $\frac{1}{2}M(M-1)$, but also the number of iterations required is $M-k$ (where k is the number of clusters to be found).

The algorithms which split clusters are called ‘divisive’. They work in, more or less, the opposite way to agglomerative algorithms :

1) Assume that all the data points belong to one big cluster.

2) Take each existing cluster and split it into every possible pair of sub-clusters. The pair of sub-clusters which are, over all, the least similar become two new clusters.

3) Calculate the similarity between each of the new clusters and each of the existing ones.

4) If less than the required number of clusters exist, go to (2).

The maximum number of similarities which have to be stored is now only $\frac{1}{2}k(k-1)$, and the number of iterations is $k-1$. However, step (2) will be so computationally intensive, for any reasonable number of data points, that divisive hierarchical clustering is rarely used.

Partitioning algorithms also fall into two types : iterative, and non-iterative.

The architypal iterative algorithm is Forgy's method [Forgy, 1965]. This works as follows :

1) Arbitrarily (or using a priori information) define a set of k cluster centres.

2) Assign each data point to the cluster whose centre it is most similar to.

3) Recalculate the cluster centres, and calculate the quality of the clustering.

4) If no data points have changed cluster during this iteration, stop. Otherwise go to (2).

The termination rule as stated at (4) is not necessarily the best, since, at some stage, the partitioning may become oscillatory. Termination may also occur when the quality of the clustering ceases to improve, or a maximum number of iterations has been reached, or a combination of all three possibilities.

Various refinements to this method have been proposed. For example, Jancey's variant [Jancey, 1966; Richards, 1986], in which the new cluster centres are the reflection of the old ones through the recalculated ones. This will speed up initial convergence, but could increase the possibility of oscillatory behaviour later on. Also, there is ISODATA [Ball & Hall, 1965] which is basically Forgy's method with the addition of rules for merging clusters which are very similar, and splitting clusters which are very elongated.

One of the problems with iterative algorithms is that, unless a maximum number of iterations is set, there is no knowing how long they will take for a particular dataset. An example of a non-iterative partitioning algorithm is MacQueen's k–means [MacQueen, 1967], which operates thus :

1) Take the first k data points as being clusters of one member each.

2) Assign each of the remaining data points to the cluster whose centre it is most similar to. Each time a cluster gains a member recalculate the position of the cluster centre.

3) Make a final pass through all the data points assigning each one to the cluster whose centre it is most similar to.

A rather more sophisticated non-iterative partitioning algorithm is the single pass algorithm [Borden, et al., 1977] :

1) Assign the first data point to cluster number one.

2) If data point i is sufficiently similar to the centre of an existing cluster, assign it to that cluster. Otherwise, make it the centre of a new cluster.

3) Repeat (2) for all the data points.

The results produced by both the MacQueen's k–means and single pass algorithms are very dependent on the order in which the data points are presented. In particular, if the data are derived from, say, an image, then the first few data points will, usually, be very similar. This would mean that the MacQueen's k–means algorithm would produce particularly poor results. The problem can, of course, be alleviated by randomising the order of the data points. However, in general, non-iterative algorithms will not produce as good results as iterative ones, because there is no opportunity for early mistakes to be corrected.

There is no reason why the algorithms described in this section need be used in isolation. We could use a hierarchical algorithm, or a non-iterative partitioning algorithm, to get an estimate of the positions of the cluster centres. Then use an iterative partitioning algorithm to refine the results.

1.5 The problem with existing techniques :

As was mentioned earlier, all the techniques described above treat the dataset as just being a collection of vectors. The cluster analysis is achieved by comparing the individual vectors with each other, or with cluster centres. This approach to the problem results in any macroscopic view of the data being lost. In particular, the number of clusters contained in the data, k, does not (in most cases) come out of the analysis in any particularly natural way. This is known as the 'cluster validation' problem.

The one exception is the single pass algorithm which does provide a value for k as part of the analysis. But, since it only takes one look at the data, the rest of the analysis tends to yield poor results. Of course, another algorithm can be applied to improve the results, once k is known.

Also, a value for k can be obtained, retrospectively, from the hierarchical algorithms. In the agglomerative case, the clusters would be continued to be merged until only one cluster remains. At each stage some quantity, like the quality of clustering, or the mean distance between clusters, is evaluated. The way that this value varies with the number of clusters is then used to determine the optimal value for k, and the results from that level of the hierarchy are then extracted.

However, the most commonly used techniques, the iterative partitioning algorithms, all require that either a fixed value of k, or, in some cases (e.g. ISODATA), a modifiable estimate of k, be specified before the analysis begins. If the cluster analysis is being provided as a computerised aid to a human analyst, then the value of k may be supplied by a cursory visual inspection of the data, or by informed guesswork. However, neither of these options is available if we desire a fully automated system. In this case a considerably less ad hoc method must be used.

If the range of possible values for k is known, and is not too large, then the cluster analysis could be performed for all the likely values of k. The quality of clustering measure could then be used to determine the value of k which gives the best description of the dataset. Otherwise, some sort of pre-clustering analysis of the data must be performed. For example, the model fitting approach used by Zhang & Modestino [1990] prior to employing the MacQueen's k–means algorithm.

1.6 A better approach to the problem :

Quite simply, since the clusters are a macroscopic property of the data, we must try and get away from the microscopic view which is the basis of the techniques described previously. We really need to look at the distribution of data points as a whole.

The ideal way of doing this is to extend the idea of fitting a model to the density distribution, to cover not only the evaluation of k but also all the other parameters which are necessary to make an adequate characterisation of the dataset. This is the basis of the 'expectation maximisation' (EM) algorithm [Hartley, 1958]. Something similar to this was used by Goldberg & Shlien in 1978, but the idea does not seem to have been developed. This may be because the datasets that they were working with were much smaller than those which are common today: 64 greylevel, 4 band, LANDSAT Multi-Spectral Scanner (MSS) data; as opposed to (for example) 256 greylevel, 7 band, LANDSAT Thematic Mapper (TM) data.

The problems associated with this approach are the same as for any model fitting, i.e. those of trying to optimise a (potentially) complex cost function in a many dimensional parameter space. These, basically, involve trying to find a compromise between speed and the possibility of locating a sub-optimal solution. It is hard to say, in general, how complex the cost function will be in this case. The number of parameters, though, *will* be large: $(N + 1)k$, if the clusters are characterised by position and amplitude alone; $3Nk$, if widths and rotation angles are included. And, there is the added complication that k **itself** is one of the parameters to be determined by the fitting process.

A much simpler approach is to use the fact that the clusters will always be associated with local maxima in the density distribution. However, in order to perform the analysis needed to locate these maxima directly, it is necessary to construct the entire feature space. As was shown in section 1.3, this is often not practical. One solution, as used by the Dipix Aries II package [Letts, 1978], is to consider only that part of the feature space which lies within the bounding box of the clusters, and to sub-sample the resulting sub-space until the number of data bins is small enough to fit within the available computer memory. Unfortunately, this method suffers not only from the enforced loss of detail, but, as will be seen in section 3.3, the location of local maxima in a feature space of high dimensionality is by no means an easy task (even when you already have some idea of where they are).

The underlying principle of this algorithm is, however, still sound: clusters of data points will, in general, have their highest density at, or near, their centres (depending on the definition of 'centre'). All that is required is a better approach to the location of these points of locally maximal density.

A question which should, perhaps, always be asked before attempting to produce an algorithm for solving a problem is 'How would one go about doing this, using only pen and paper ?'. With the possible exception of the hierarchical methods, this question does not appear to be the basis of any of the existing algorithms. Certainly one cannot picture anyone sitting down and performing Forgy's method on the back of an envelope.

How, then, **would** one go about locating the local maxima using only pen and paper ? Let us assume the luxury of a data plotting package, in order to avoid having to consider things at too basic a level. This would allow us to produce two dimensional plots of the density distribution of the data points, like those in figures 1.3 to 1.5. These happen to have the density variation shown by contour lines, but we could just as well work with plots which show the density by intensity or colour. The actual feature space is three dimensional, and these plots show all the possible two dimensional views of it (the data comes from an image of the BBC testcard, after the removal of lighting effects (see chapter 4)).

The local maxima in the two dimensional plots are fairly easily located by eye (taking into account the noise due to inadequate sampling) – there would appear to be six of them. The three dimensional coordinates of the cluster centres can then be determined by crossreferencing, as follows :

local maxima at		a cluster centre at
$(a, b), (b, c), (c, a)$	$\Rightarrow$	(a, b, c)

In fact, a cluster centre at (a, b, c) can be inferred from just the local maxima at (a, b) and (b, c), but the knowledge that there is a local maximum at (c, a) provides a sanity check – particularly useful if noise causes the maxima locations to shift slightly between different views

The cluster centres determined from the data in figures 1.3 to 1.5 are shown in table 1.1.

Local maxima			Cluster centre
Blue,Green	Green,Red	Red,Blue	
152,119	119,113	113,152	152,119,113
130,121	121,133	133,130	130,121,133
110,123	123,150	150,110	110,123,150
108,108	108,170	170,108	108,108,170
136,150	150,98	98,136	136,150,98
90,136	136,154	154,90	90,136,154

Table 1.1: Cluster centre locations inferred from figures 1.3 to 1.5.

The most obvious problem associated with this approach is that of clusters being superimposed when the feature space is projected down to two dimensions. Then, the number of maxima located in one, or more, of the two dimensional views will be less than the number of clusters. This will not be a problem if the clusters are in exact alignment (see figure 1.6), since the crossreferencing will still work – it will just mean that several cluster centres

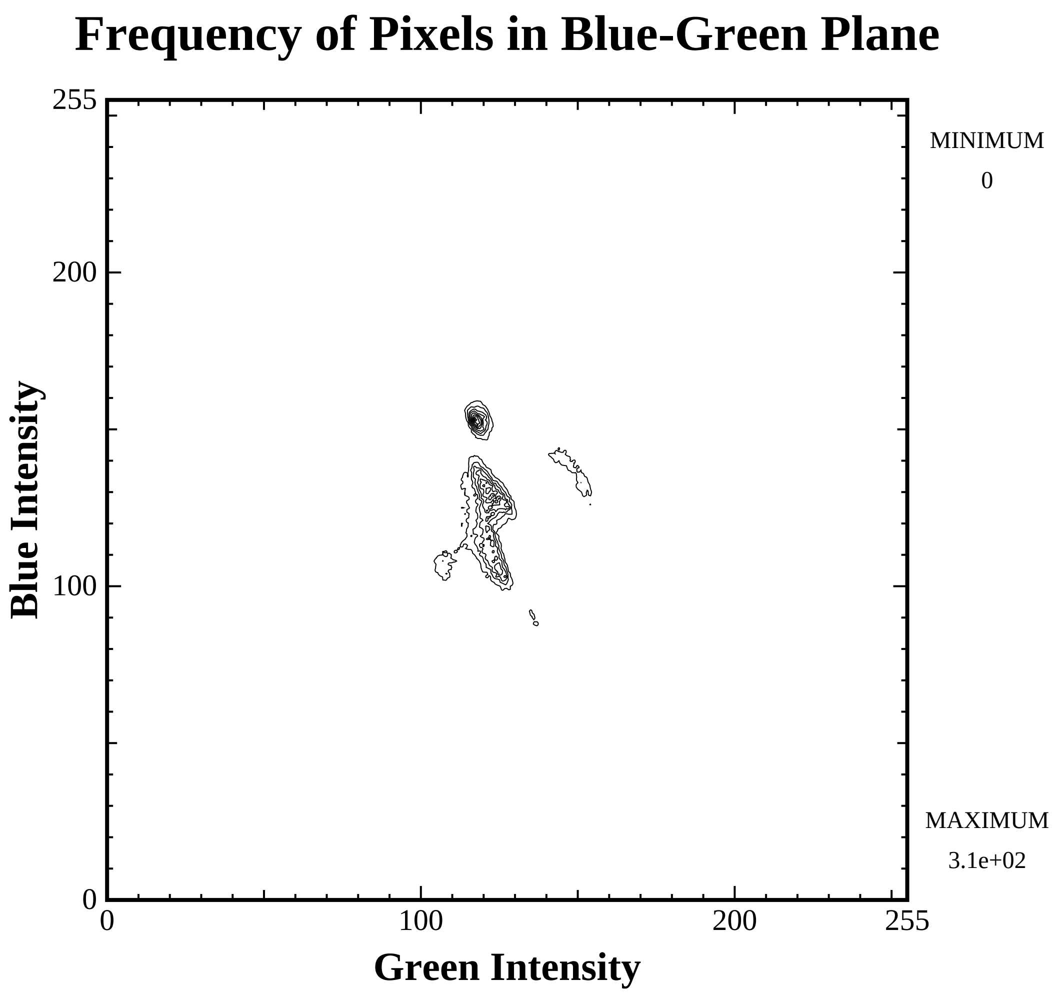


Figure 1.3: Density of data points in BBC testcard image, viewed down the Red axis.

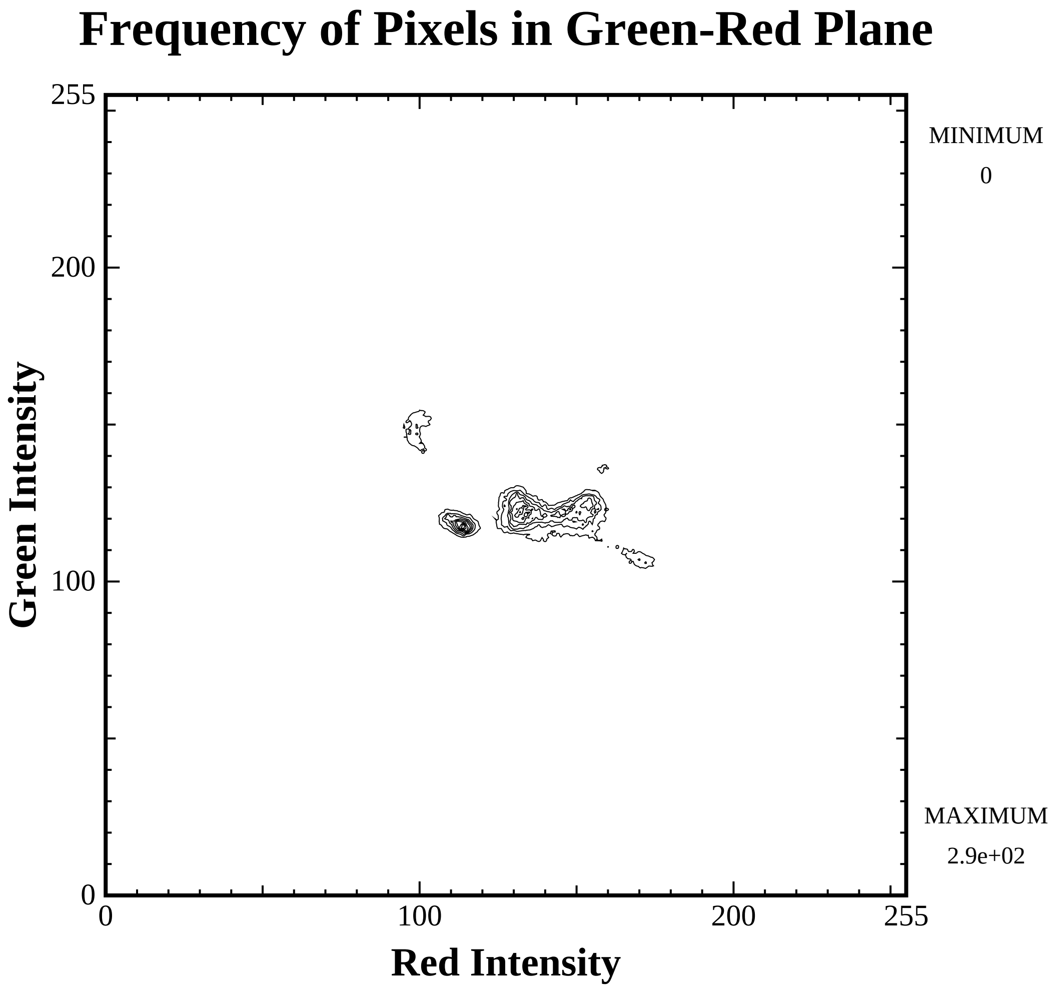


Figure 1.4: Density of data points in BBC testcard image, viewed down the Blue axis.

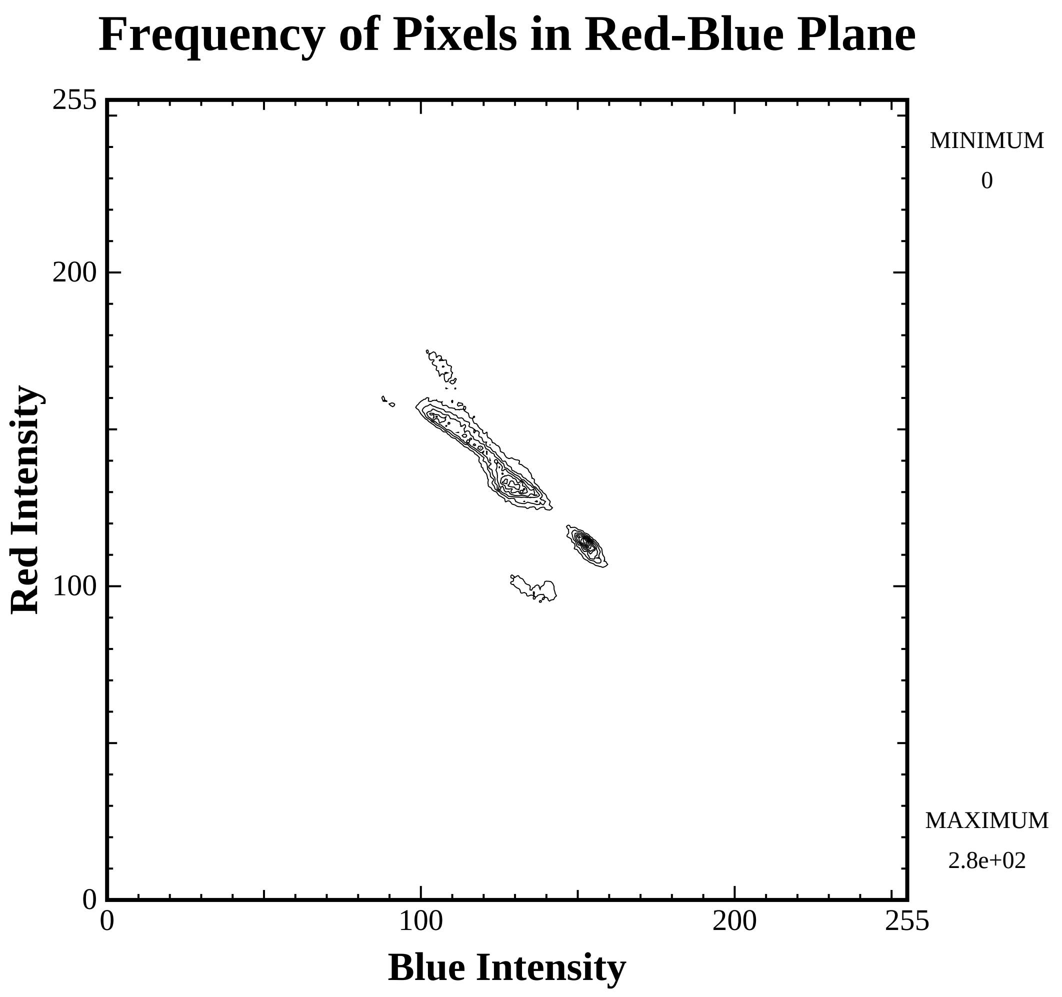


Figure 1.5: Density of data points in BBC testcard image, viewed down the Green axis.

make use of the same local maximum. If, however, the clusters are just close together (see figure 1.7) the crossreferencing will fail.

If a particular pair of clusters are close together, with respect to a particular pair of variables, (x, y), then they will appear superimposed in only the two dimensional view which has both x and y as coordinates. Now, there are $\frac{N(N-1)}{2}$ possible views of the data, of which only $N-1$ would be sufficient for the calculation of the cluster centre location. So, provided $N \geq 3$, losing one of the views does not mean that the cluster centre location cannot be calculated. Unfortunately, although the views do contain redundant information, once we start considering more than one pair of clusters being close together with respect to more than one pair of variables, the crossreferencing becomes too complicated to contemplate. What is really needed is for the analysis of the views to be able to locate the projections of the cluster centres even when the clusters are superimposed.

Producing a computer algorithm to mimic the human analyst's location of local maxima in the two dimensional views would be fairly simple. One possible approach is to apply a suitable amount of smoothing (to reduce the noise) and then apply a gradually decreasing threshold. Of the points which appear above the threshold, those which are not connected to any found with the previous threshold must be local maxima (see figure 1.8). However, the complication of trying to deal with superimposed clusters is far from easily dealt with. The approach to the problem of untangling superimposed clusters, if not its actual solution, might be made much simpler by only having to work in one dimension. Certainly, the space that we would be working in would be much smaller, and simpler. The superpositioning problem would not be made any worse than it already is, and the crossreferencing stage should only be marginally more complicated.

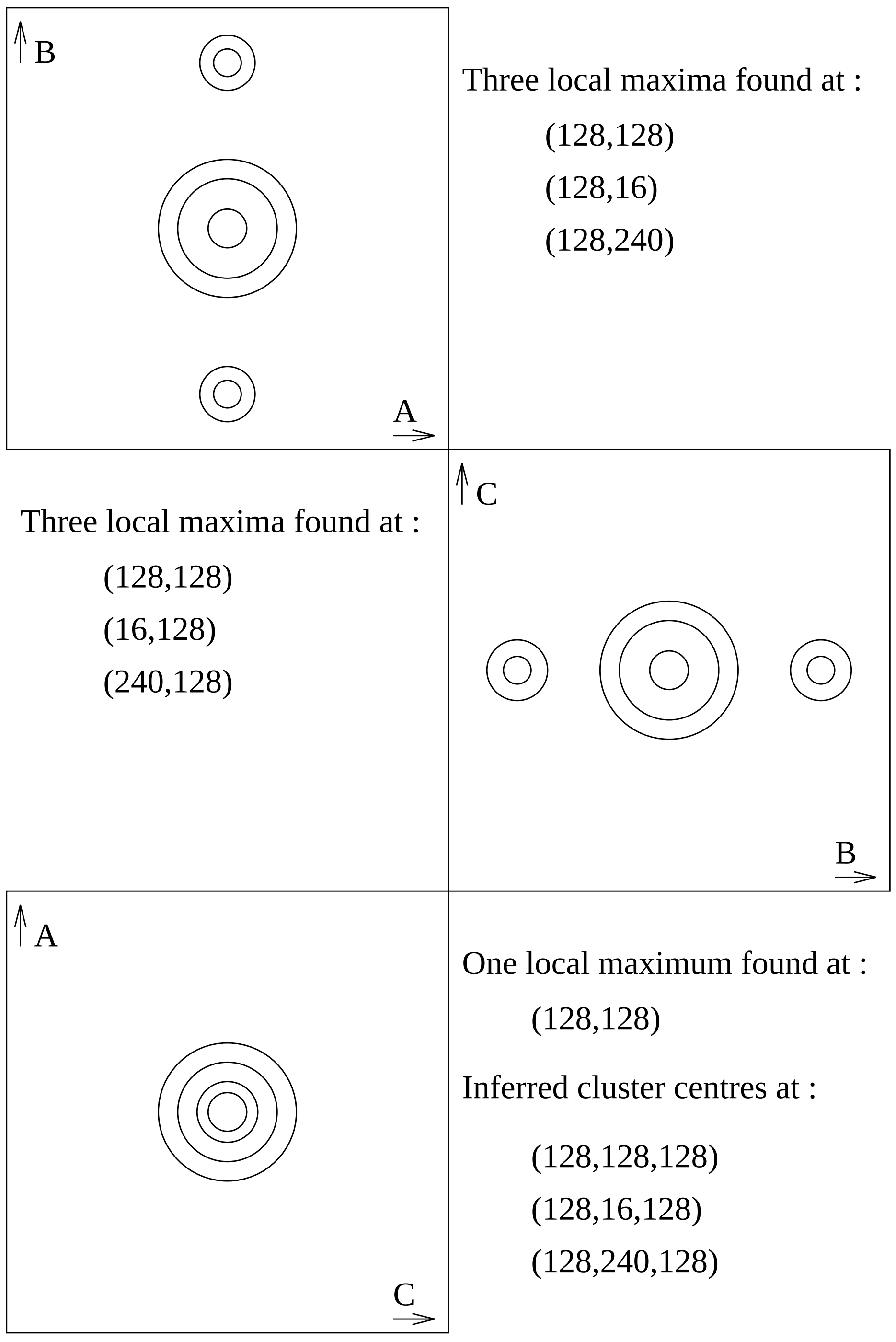

B
A
Three local maxima found at :
(128,128)
(128,16)
(128,240)
C
Three local maxima found at :
(128,128)
(16,128)
(240,128)
B
A
One local maximum found at :
(128,128)
Inferred cluster centres at :
(128,128,128)
(128,16,128)
(128,240,128)
C

Figure 1.6: Superimposition of clusters, leading to the multiple use of a local maximum during crossreferencing.

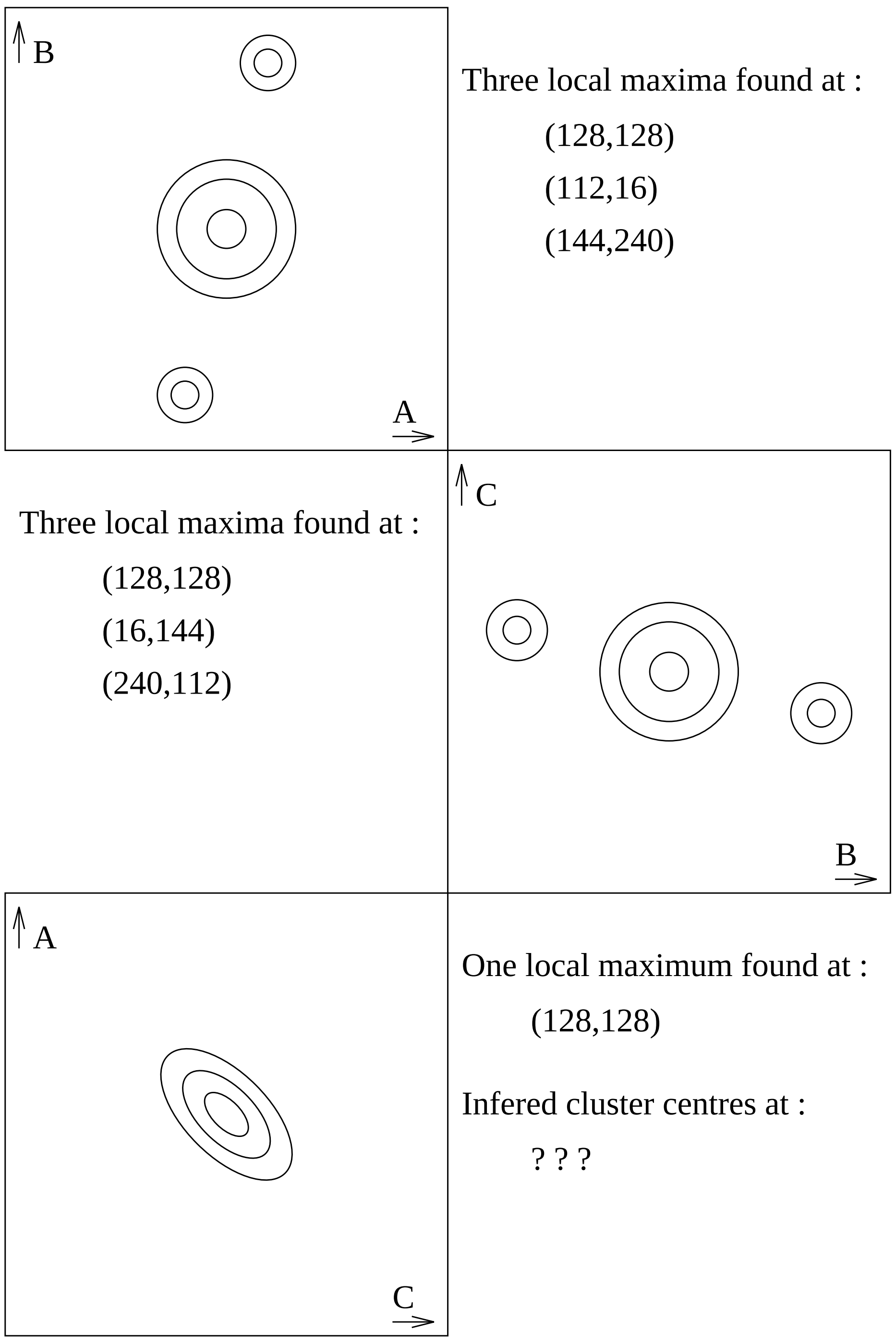

B
A
Three local maxima found at :
(128,128)
(112,16)
(144,240)
C
Three local maxima found at :
(128,128)
(16,144)
(240,112)
B
A
C
One local maximum found at :
(128,128)
Infered cluster centres at :
? ? ?

Figure 1.7: Superimposition of clusters, leading to the failure of the crossreferencing of the local maxima locations.

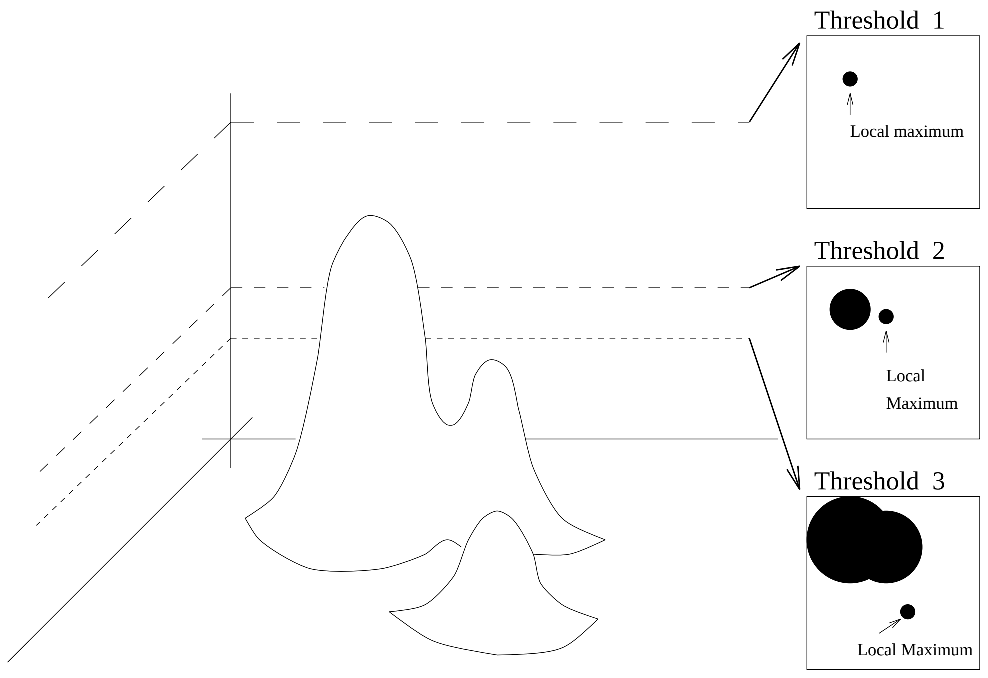


Figure 1.8: Local maxima location in a two dimensional feature space, by gradual thresholding.

Figure 1.9 shows the one dimensional projections of the dataset used for figures 1.3 to 1.5, and the projected locations of the cluster centres determined in table 1.1. There is some superpositioning, but it does look as if it should be possible to use the locations of the peaks, in these sort of plots, as the basis of a method for locating the cluster centres.

We are now in the position of being able to reformulate the rather vague question ‘Where are the cluster centres in an N–dimensional feature space ?’ as the more specific questions ‘Where are the (possibly superimposed) peaks in the N one dimensional projections of the density distribution in an N–dimensional feature space ?’, and ‘How should the positions of these peaks be combined, in N–dimensions, in order to give the locations of the cluster centres ?’.

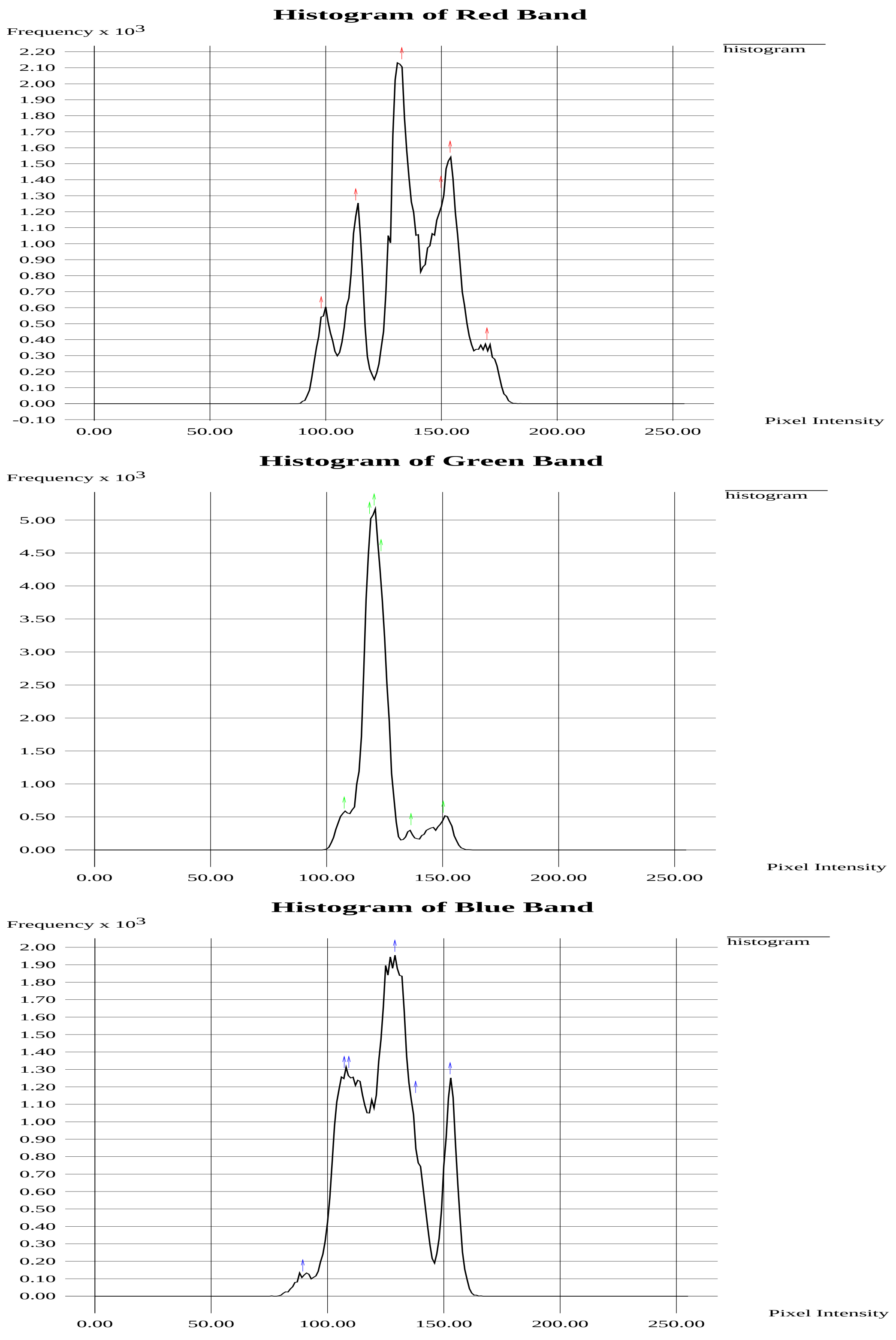


Figure 1.9: One dimensional views of the density distribution of the BBC testcard dataset.

This structured, or modular, approach to the problem should, hopefully, mean that, not only is the problem more easily solved, through being better defined, but also that, if the solution should fail at all, the reasons for its failure can be more easily traced, and dealt with. It is this modular approach, and the algorithms which go to make up the modules, which comprise the major novel aspect of the work which follows. Chapters two and three deal with the answering of our two questions. Then chapters four to six look at the application of this approach to cluster location in the field of image processing. Finally, chapters seven and eight compare the approach with some of those discussed in this chapter, and consider its usefulness and possible future development.

CHAPTER 2
Location of Peaks in One Dimension

The production of one dimensional feature sub-spaces from an N–dimensional feature space is examined. Then an answer to the question 'Where are the peaks in the one dimensional projections ?' is sought. Existing techniques for locating peaks in one dimension are reviewed, and a new, deconvolution based, technique is introduced. This is shown to work in cases where previous techniques must fail.

2.1 Reducing the number of dimensions :

The first task, then, is to convert the N–dimensional feature space in to N one dimensional feature spaces. This is done by 'projecting' the data set onto a set of orthogonal vectors :

$$\rho_i(x) = \int \delta(x - \mathbf{v_i} \cdot \mathbf{x})\, \rho(\mathbf{x})\, d\mathbf{x} \qquad [1 < i < N] \tag{2.1}$$

where δ is the Dirac delta function; and the $\mathbf{v_i}$ are the orthogonal vectors:

$$\mathbf{v_i} \cdot \mathbf{v_j} = 0, \qquad i \neq j \qquad [i, j = 1, \ldots, N]. \tag{2.2}$$

Unless we wish to rotate the feature space, in order to try and maximise the separability of the clusters, the $\mathbf{v_i}$ will be the basis vectors of the N–dimensional space. This means that the one dimensional feature spaces can, more simply, be produced by considering separately the components of the N–dimensional feature vectors, i.e. we can just drop the vector notation from equation 1.3 :

$$\rho_i(x) = [M \sum_{\Omega_a \in \Omega} p(\Omega_a)\, p(x_i|\Omega_a)] \bullet n(x_i) \qquad [1 < i < N]. \tag{2.3}$$

Prior to analysis, the density distributions will usually be converted into a discrete form (if the data was not discrete in the first place), by 'binning' the data according to which of a set of contiguous ranges of values it falls into. In its simplest form, this will involve shifting and scaling the data to lie in a suitable range of positive values, and then rounding them to the nearest integer. This will give a 'histogram' of the data set:

$$h_i(\mathrm{nint}(x^{'})) = [\rho_i(x^{'}) * \Pi(x^{'})] \times \mathrm{III}(x^{'}) \tag{2.4}$$

where $x^{'} = a(x-b)$, the shifted and scaled x; $\mathrm{nint}(x^{'})$ is the nearest integer to $x^{'}$; $\Pi(x^{'})$ is the square pulse function:

$$\Pi(x^{'}) = \begin{cases} 1 : & |x^{'}| \leq \frac{1}{2} \\ 0 : & |x^{'}| > \frac{1}{2} \end{cases} \tag{2.5}$$

and $\mathrm{III}(x^{'})$ is the Shah function:

$$\mathrm{III}(x^{'}) = \sum_{i=-\infty}^{+\infty} \delta(x^{'} - i) \tag{2.6}$$

2.2 Location of extrema in $h_i(x)$:

Traditionally, one dimensional histograms are analysed by using their maxima to locate the peaks, and their minima to separate (partition) them [Sezan, 1990]. Since $h_i(x)$ is a discrete function, the most straight forward way of locating its extrema is to convolve it with the real (as opposed to Fourier) space filters $\boxed{-1|0|+1}$ and $\boxed{+1|-2|+1}$. The first convolution will form the first derivative of $h_i(x)$, $h_i^{'}(x)$, and the second will form the second derivative, $h_i^{''}(x)$. The extremal points of $h_i(x)$ are, then, those values of x at which $h_i^{'}(x)$ takes a value of zero. Or, rather, changes sign, because of the discrete nature of the operations. The sign of $h_i^{''}(x)$ determines whether the extremal point is a maximum ($h_i^{''}(x) < 0$), or a minimum ($h_i^{''}(x) > 0$).

The main problem with this 'finite difference' approach is that the noise will cause a large number of extremal points to be found which are not extrema of the ideal density distribution. One way around this is to smooth $h_i(x)$ with a real space filter in order to reduce the effects of the noise. If an additive Gaussian noise model is assumed, it is best dealt with using a simple mean filter (e.g. $\boxed{0.2}\boxed{0.2}\boxed{0.2}\boxed{0.2}\boxed{0.2}$) [Bovik, et al., 1983], but other types of noise, particularly 'impulsive' (or 'salt & pepper') noise, can require more sophisticated (usually non-linear) techniques [Kundu, et al., 1984; Lee & Kassam, 1985; Lee & Tantaratana, 1990].

Another possibility is to perform the differentiation in Fourier space [Bracewell, 1978]. This approach uses the fact that (assuming $f^{'}(x)$ and $f^{''}(x)$ are square integrable):

$$\mathcal{F}\{f^{'}(x)\} = 2\pi i u F(u) \tag{2.7}$$

where $F(u) = \mathcal{F}\{f(x)\}$, and $\mathcal{F}$ denotes a Fourier transform.

$$\Rightarrow \mathcal{F}\{f^{''}(x)\} = -4\pi^2 u^2 F(u) \tag{2.8}$$

The derivatives are then obtained by applying the inverse Fourier transform.

The advantage of this approach is that, in Fourier space, a low-pass filter can be applied to reduce the effects of the noise. We must be careful, though, that the filter used does not introduce 'ringing' [Gonzalez & Wintz, 1987], as this will cause false peaks to be found. For this reason a Gaussian filter is the best to use.

A third technique, by which the extremal points may be located, is Haddon's multi-resolution approach [Haddon, 1987]. Initially the data is binned into two ranges of values – min(x) to [max(x)+min(x)]/2, and [max(x)+ min(x)]/2 to max(x) – and the bins classified as being maxima, minima, or neither. Each 'parent' bin is then divided into two 'child' bins, and the whole process is repeated until the data is binned at its full resolution.

We can distinguish real extremal points from those due to noise by looking at how far back in their 'lineage' they can be traced as being consistently classified as a maximum, or a minimum. The extremal points caused by noise will only be traceable over one or two 'generations', whereas the real ones will be traceable back to a point close to the data's 'genesis'.

The main advantage of this technique is that it removes the need for any filtering.

The result of using the above three techniques on several $h_i(x)$ is shown in figures 2.1 to 2.9. There are nine data sets used for these comparisons. Three different sets of (roughly) Gaussian peaks, with the positions and widths shown in table 2.1, and three different levels of noise : low (corresponding to 10^5 data points); medium (10^4 data points); and high (10^3 data points).

The filter sizes used with the finite difference and Fourier differentiation algorithms were chosen by finding, for each noise level, the most smoothing which could be applied that still resulted in all three of the well spaced peaks being located.

Data Set	$\bar{x}_1$	σ_1	$\bar{x}_2$	σ_2	$\bar{x}_3$	σ_3
Well Spaced	32	16	128	12	192	8
Moderately Spaced	96	16	128	12	152	8
Closely Spaced	104	16	128	12	140	8

Table 2.1: Peak positions & widths used for peak location tests.

There are three main problems with the various techniques. Firstly, with widely spaced peaks, both of the filter based techniques fail to find minima between the peaks(figures 2.1, 2.4, & 2.7). This is because the gaps between the peaks are so broad that the first derivative just becomes zero, rather than

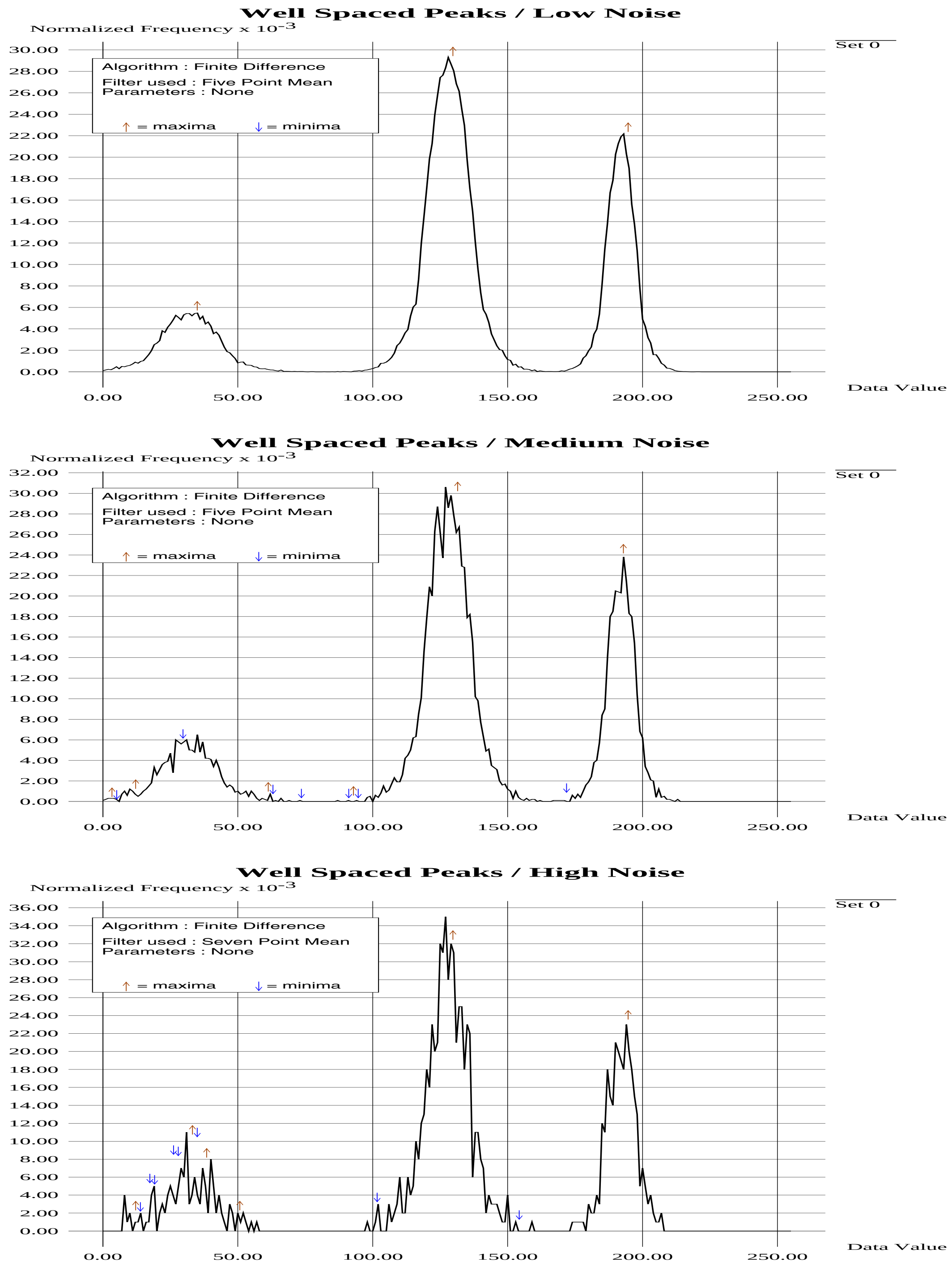


Figure 2.1: Finite difference analysis, for widely spaced peaks, with varying degrees of noise.

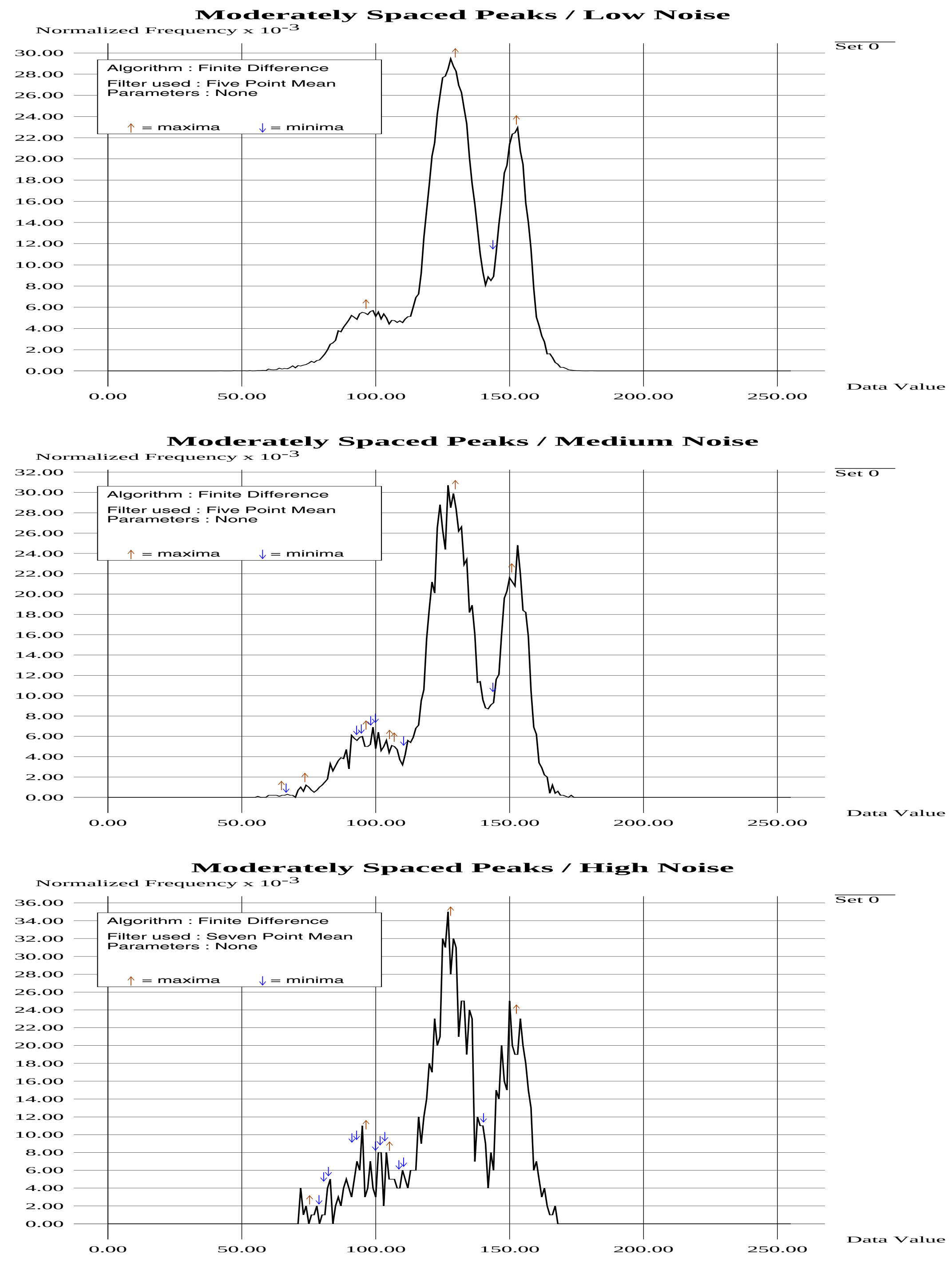


Figure 2.2: Finite difference analysis, for moderately spaced peaks, with varying degrees of noise.

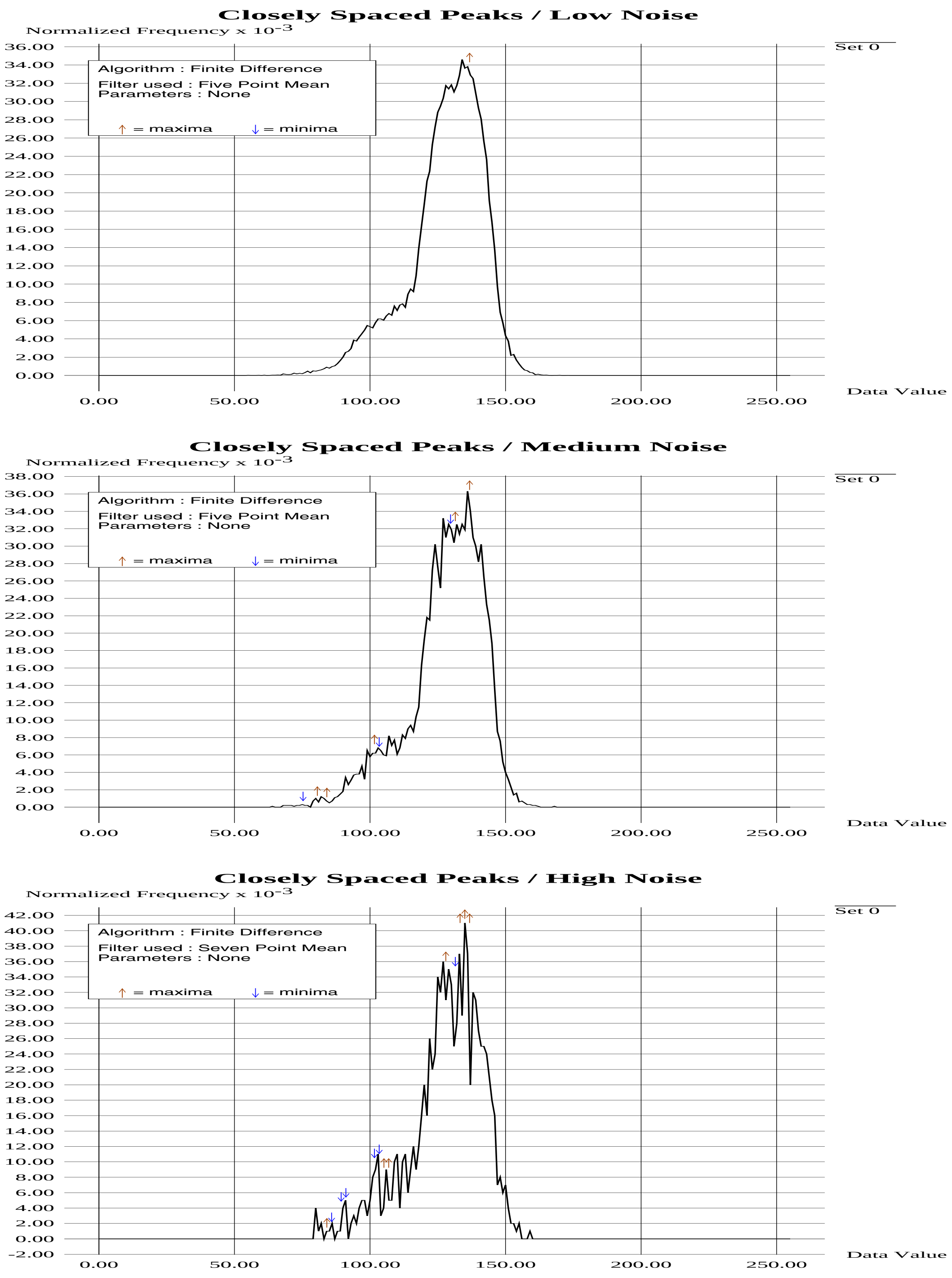
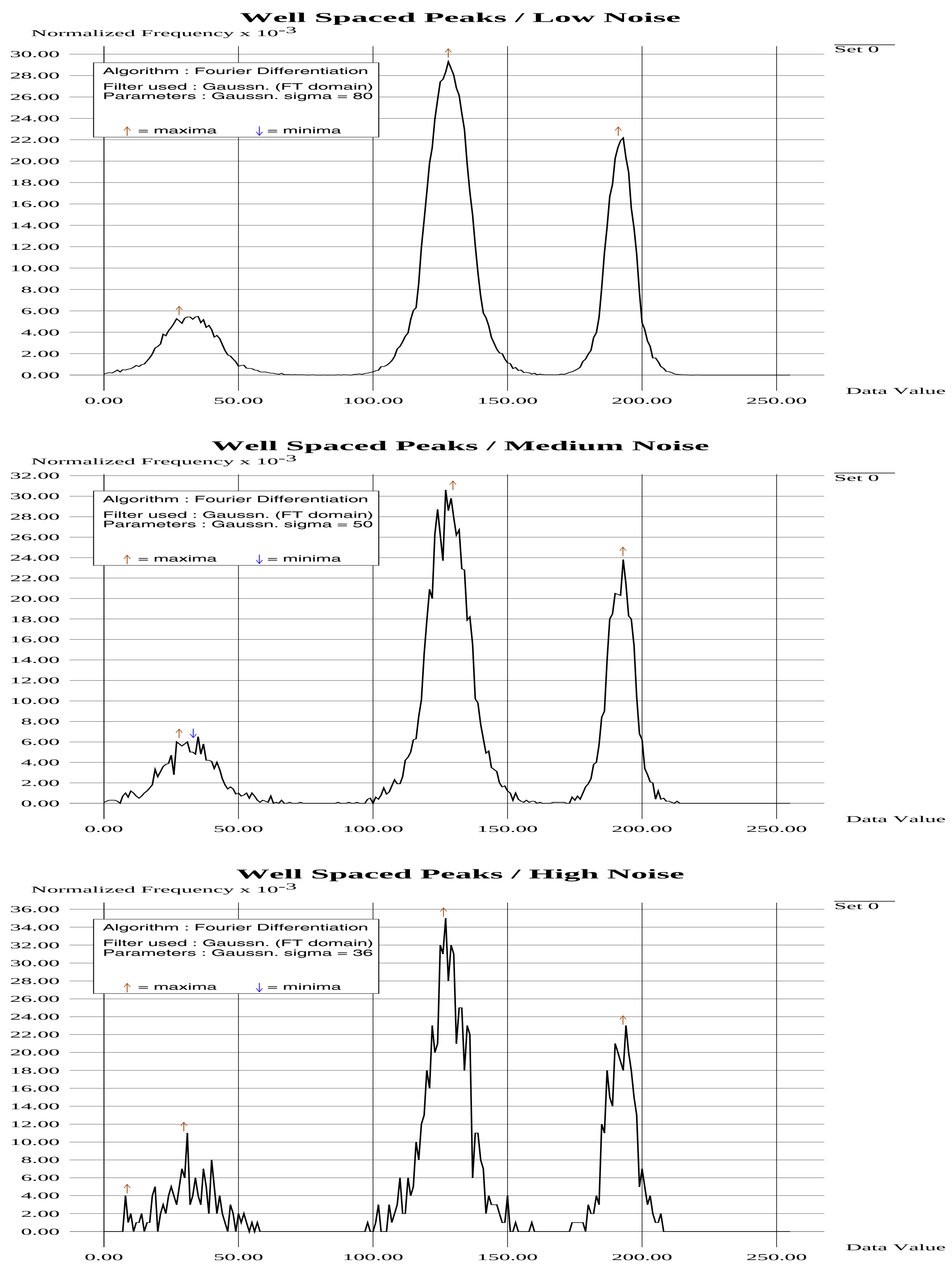


Figure 2.3: Finite difference analysis, for closely spaced peaks, with varying degrees of noise.



Algorithm : Fourier Differentiation
Filter used : Gaussn. (FT domain)
Parameters : Gaussn. sigma = 80

Algorithm : Fourier Differentiation
Filter used : Gaussn. (FT domain)
Parameters : Gaussn. sigma = 50

Algorithm : Fourier Differentiation
Filter used : Gaussn. (FT domain)
Parameters : Gaussn. sigma = 36

Figure 2.4: Fourier differentiation analysis, for widely spaced peaks, with varying degrees of noise.

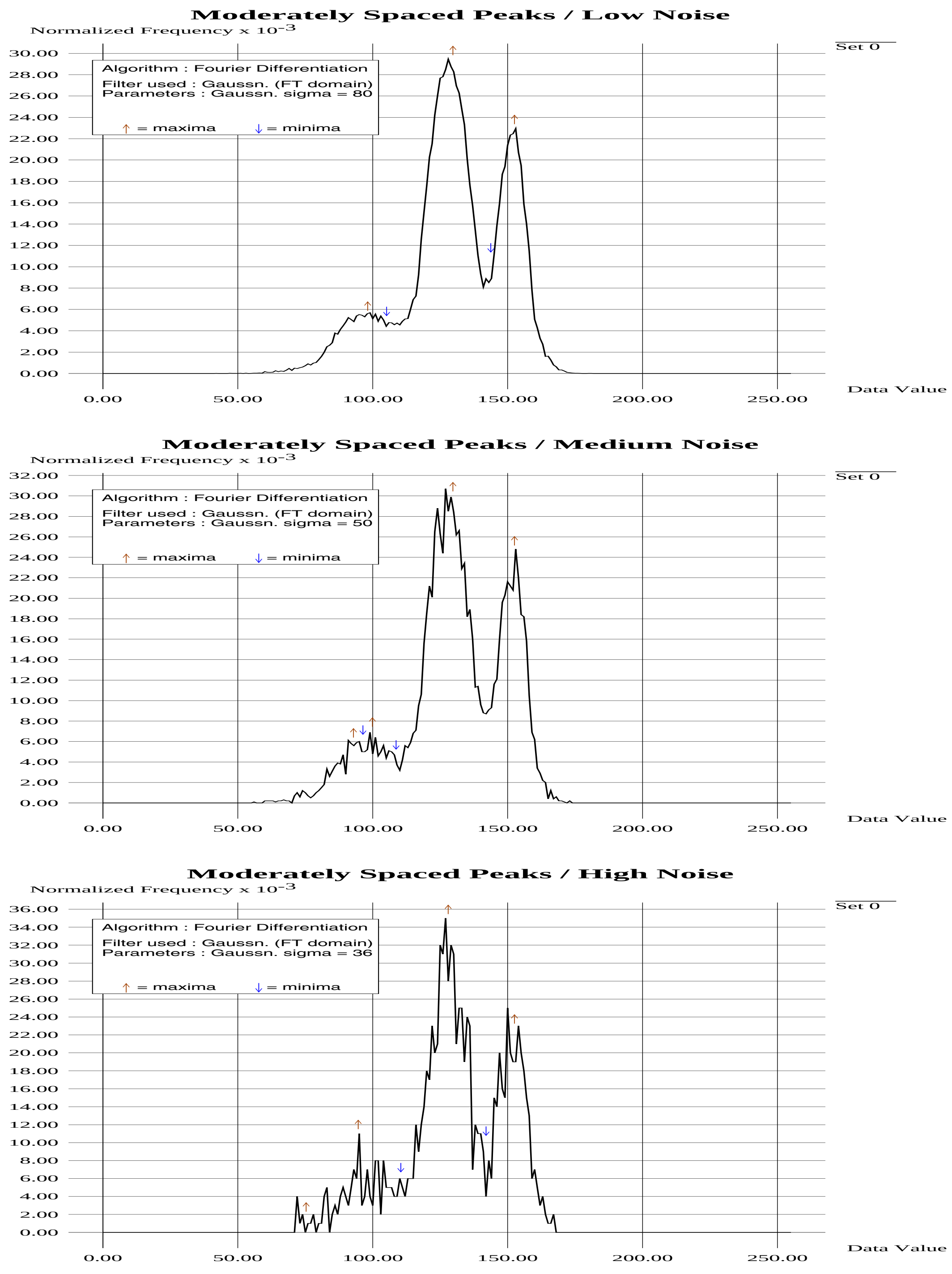


Figure 2.5: Fourier differentiation analysis, for moderately spaced peaks, with varying degrees of noise.

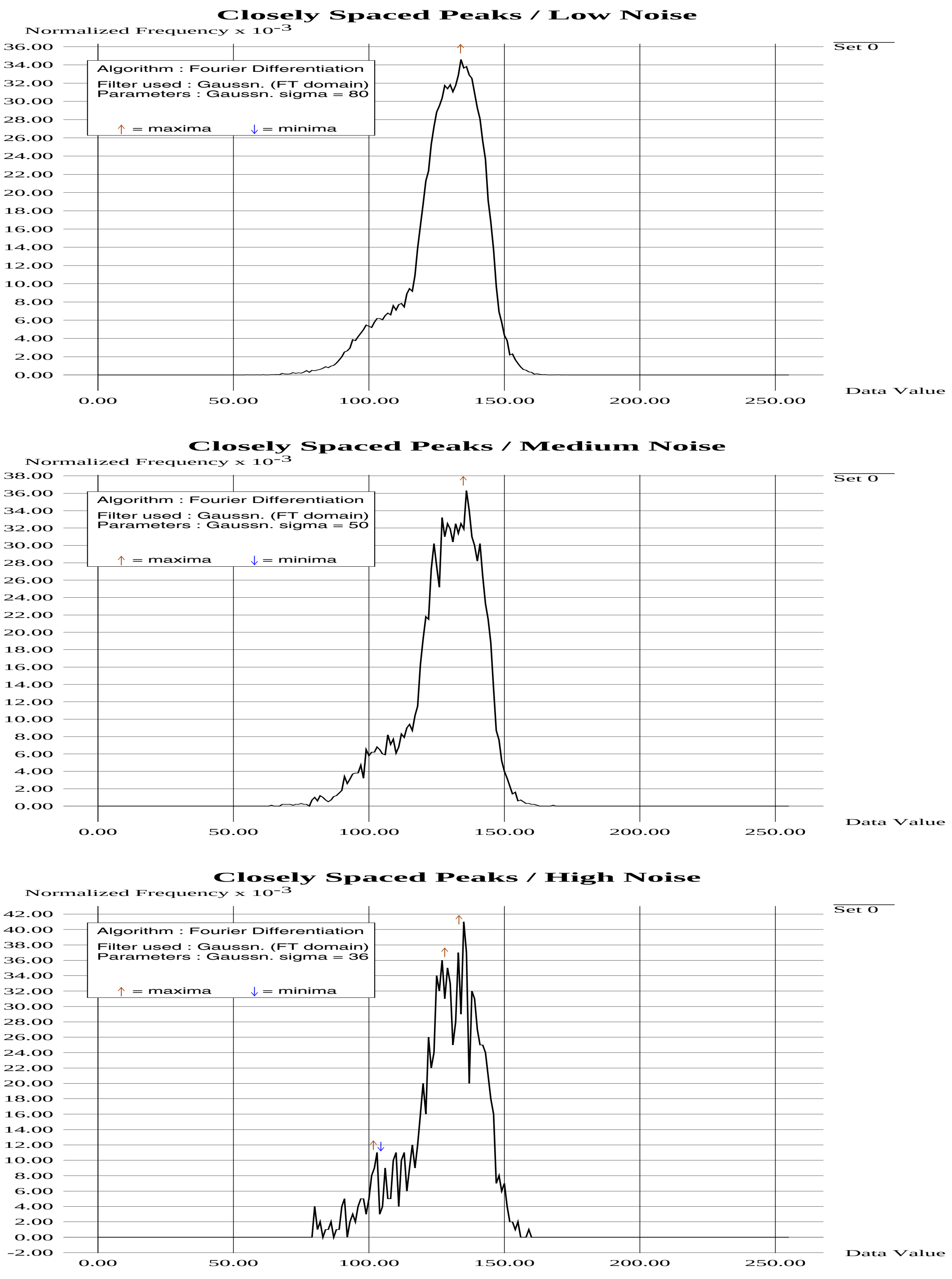


Figure 2.6: Fourier differentiation analysis, for closely spaced peaks, with varying degrees of noise.

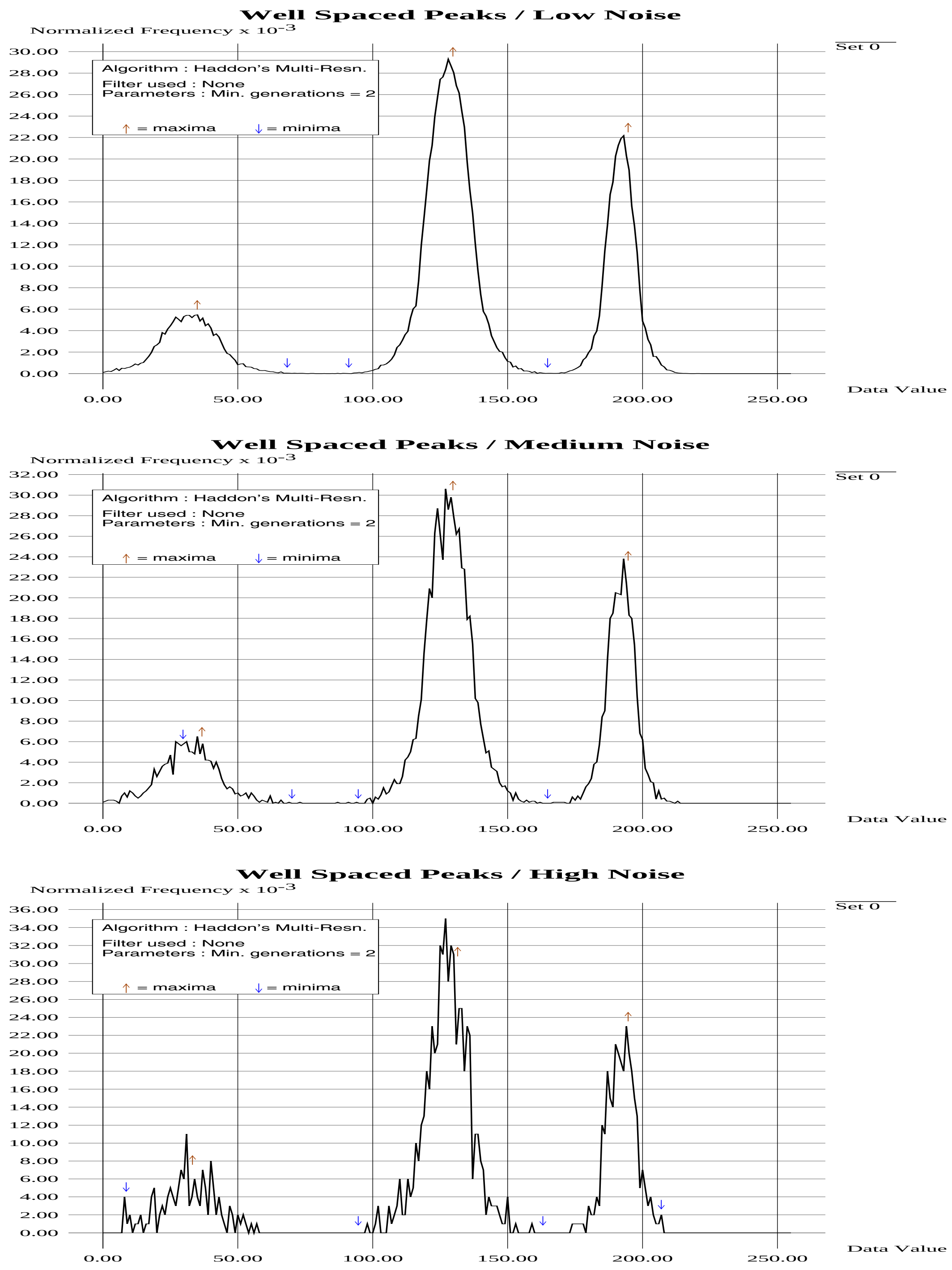


Figure 2.7: Haddon's multi-resolution analysis, for widely spaced peaks, with varying degrees of noise.

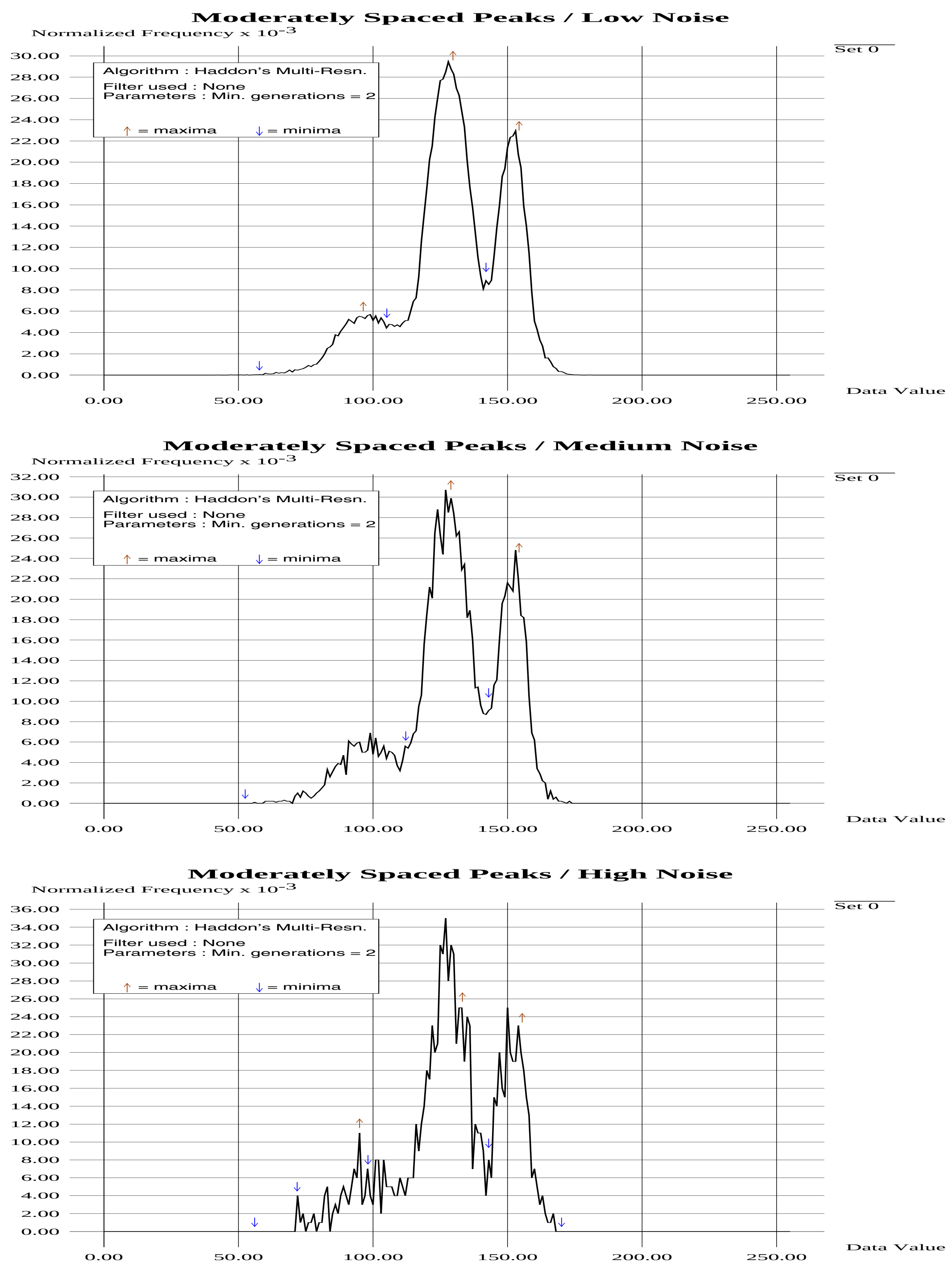


Figure 2.8: Haddon's multi-resolution analysis, for moderately spaced peaks, with varying degrees of noise.

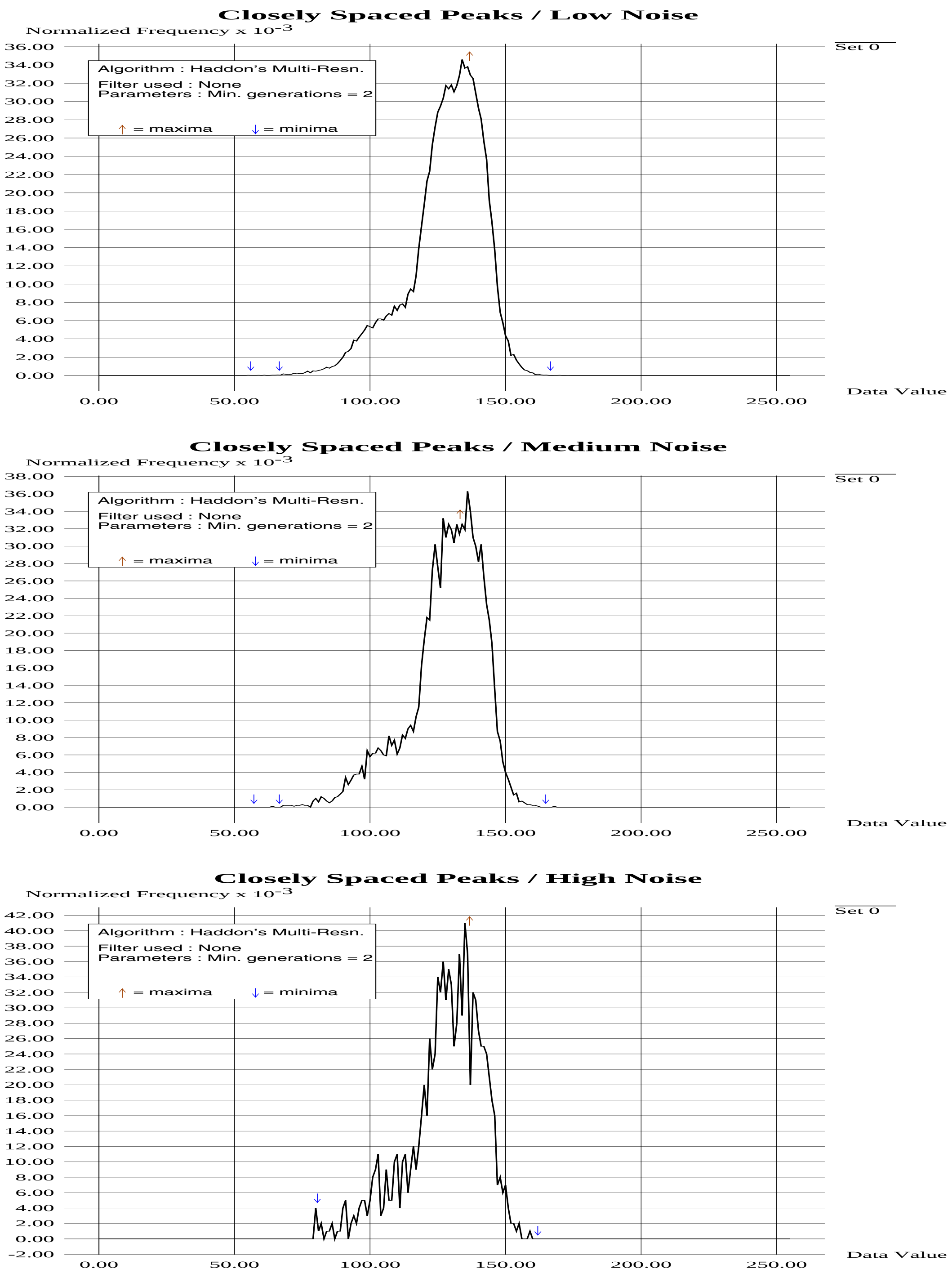


Figure 2.9: Haddon's multi-resolution analysis, for closely spaced peaks, with varying degrees of noise.

changing sign.

Secondly, both of the filter based techniques find far too many false extrema at high noise levels (figures 2.1 to 2.9). This indicates that the histograms need to be smoothed even more than they have been, but then we run the risk of missing some of the real extrema.

Finally, when spurious extrema are discounted, all of the techniques find too few extrema when the peaks are closely spaced (figures 2.3, 2.6, & 2.9). That is, predictably, a finite difference approach will always fail when two peaks become so close together that there is no longer a local maximum associated with each of them.

What we really need is a technique which will give proper estimates of the peak widths (and other parameters, if necessary), and which, as was stated in chapter one, can cope when the peaks overlap significantly.

2.3 Adaptive deconvolution of probability distribution functions :

Let us consider equation 2.3 and examine what it means. The heights of the peaks in $\rho_i(x)$ are determined by the values of $M\,p(\Omega_a)$, and the shapes of the peaks by $p(x|\Omega_a)$. If we can represent the $p(x|\Omega_a)$ as functions of a set of parameters ($\Lambda = \{\lambda\}$), then equation 2.3 can be rewritten as:

$$\rho_i(x) = \sum_{\Omega_a \in \Omega} P_{\Omega_a}(\Lambda_{\Omega_a}) \tag{2.9}$$

where $P_{\Omega_a}(\Lambda_{\Omega_a})$ is the parameterised probability distribution function (PDF) associated with cluster Ω_a, and the noise term ($\bullet n(x_i)$) has been omitted in order to simplify the subsequent analysis – i.e. it is assumed that we have the ideal case (adequate sampling statistics). Two of the parameters will be the amplitude of the peak, $\lambda_1(\Omega_a) \propto M\,p(\Omega_a)$, and its position, $\lambda_2(\Omega_a)$). If the P_{Ω_a} are redefined so that they are normalised and take their maximum value at $x = 0$ (i.e. they are no longer dependent on $\lambda_1(\Omega_a)$ or $\lambda_2(\Omega_a)$), then equation 2.9 becomes :

$$\rho_i(x) = \sum_{\Omega_a \in \Omega} P_{\Omega_a}(\Lambda_{\Omega_a}) * [\lambda_1(\Omega_a)\delta(x - \lambda_2(\Omega_a))]. \tag{2.10}$$

That is, we can consider $\rho_i(x)$ to be constructed from a set of delta functions, located at the positions of the centres of the peaks, and with the same heights as each of the peaks, each of which is convolved with a function which describes the shape of that particular peak.

If the parameters of the PDF's are the same for all the clusters, i.e. the peaks are all the same shape, then the P_{Ω_a} could be deconvolved from the delta functions by making use of the convolution theorem:

$$\begin{aligned}
\mathcal{F}\{\rho_i\} &= \mathcal{F}\{\sum_{\Omega_a} P_{\Omega_a}(\Lambda_{\Omega_a}) * \lambda_1(\Omega_a)\delta(x - \lambda_2(\Omega_a))\} && (2.11)\\
&= \mathcal{F}\{P * \sum_{\Omega_a} \lambda_1(\Omega_a)\delta(x - \lambda_2(\Omega_a))\} && (2.12)\\
&= \mathcal{F}\{P\} \times \mathcal{F}\{\sum_{\Omega_a} \lambda_1(\Omega_a)\delta(x - \lambda_2(\Omega_a))\} && (2.13)\\
\Rightarrow \sum_{\Omega_a} \lambda_1(\Omega_a)\delta(x - \lambda_2(\Omega_a)) &= \mathcal{F}^{-1}\{\mathcal{F}\{\rho_i\}\frac{1}{\mathcal{F}\{P\}}\} && (2.14)
\end{aligned}$$

Actually, equation 2.14 will only be true if ρ_i is of narrower bandwidth than P (otherwise division by zero will occur). This will, almost certainly, not be true, because of the noise term. Therefore, we would have to apply a damping term at the high spatial frequencies. That is, we would need to use a Wiener filter [Helstrom, 1967]. With this, equation 2.14 becomes:

$$\sum_{\Omega_a} \lambda_1(\Omega_a)\delta(x - \lambda_2(\Omega_a)) = \mathcal{F}^{-1}\{\mathcal{F}\{\rho_i\}\frac{1}{\mathcal{F}\{P\}}\frac{|\mathcal{F}\{P\}|^2}{|\mathcal{F}\{P\}|^2 + K}\} \tag{2.15}$$

where K is a constant, the value of which depends on the amount of noise present. The left hand side of the equation gives us the positions, and heights, of the peaks, and all of the other parameters are known (since we fixed them all, in order to be able to do the deconvolution).

In general, however, the parameters will vary with Ω_a, and so an adaptive (and, therefore, non-linear) approach must be used. The CLEAN algorithm [Högbom, 1974] is just such a technique, much used in radio astronomy. The operation of the original algorithm is as follows :

> let $D = P_a * C$, where D is the 'dirty image', P_a is the actual point spread function of the imaging system, and C is the 'clean image',
>
> **1)** find the brightest point in $|D|$, at x_m, if $|D(x_m)|$ is less than some threshold go to (6),
>
> **2)** fit a copy of P_a to the region of D centred on x_m,
>
> **3)** subtract a fraction of the fitted P_a from D (centred on x_m),
>
> **4)** add a delta function of amplitude $\alpha D(x_m)$ to C at x_m,
>
> **5)** repeat from (1),
>
> **6)** convolve C with an ideal point spread function (P_i).
>
> α is normally around 0.5, and the threshold is normally a few percent of the intensity of the first brightest point found.

As originally formulated, the only fitting parameter in stage (2) is the amplitude ($\lambda_1(\Omega_a)$). This was because the problem that it was designed to solve, in radio astronomy, has all the other parameters of P_a fixed. Also, the original formulation allows for the deconvolution of negative peaks. This, along with the subtraction of only a fraction of the peak at each iteration, provides a powerful mechanism for the gradual correction of 'mistakes' made early on in the process. This does mean, though, that a lot of radio astronomical images do have small patches where the sky appears to be of negative intensity !

Whilst working on multi-resolution deconvolution of radio astronomical images [Naylor, 1988] it was realised that any, or all, of the parameters of P_a could be included in the fitting process, in particular the width, $\lambda_3(\Omega_a)$. This, along with the fact that we do not need to produce a 'clean' version of $h(x)$, and the fact that we do not want to have the possibility of negative peaks, results in the following algorithm for the analysis of the one dimensional feature sub-spaces :

1) find the highest point in $h_i(x)$, at x_m, if $h_i(x_m)$ is less than some threshold go to (6).

2) fit $P(\Lambda_{\Omega_a})$ to the region of $h_i(x)$ around x_m,

3) record the values of the parameters (Λ) which give the best fit,

4) subtract the fitted $P(\Lambda_{\Omega_a})$ from $h_i(x)$, centred on x_m, setting any resulting negative values to zero,

5) repeat from (1),

6) finish.

In this new version, the whole of the peak is subtracted in one go, since, otherwise, it would be difficult, if not impossible, to recombine the Λ determined for each peak so as to get the Λ_{Ω_a}.

The restrictions on this method are as follows. Firstly, all of the $P_{\Omega_a}(\Lambda_{\Omega_a})$ must be of the same general form, $P(\Lambda_{\Omega_a})$, (e.g. all Gaussians).

Secondly, since the peaks are being subtracted out in one piece, and negative peaks are no longer allowed, we have no mechanism for the correction of early mistakes. In particular, if two peaks are so close together that the region of overlap forms a peak which is higher than either of the real ones (see figure 2.10), then this will be found as a peak and deconvolved out first. This may well result in the distortion of the real peaks in such a manner that either their properties are not correctly determined, or one of them is not found at all.

If the two peaks are Gaussians of the same height and width, then this sort of problem will occur when their separation is less than about 1.25 standard deviations. This limit will vary if the peaks are of different heights and/or widths, but it is probably safe to state that this approach may suffer problems when a histogram contains two peaks which are separated by a distance which is less than 1.25 times the standard deviation of the wider of the two. Having said that, we will still have found at least one peak which lies inside the limits of both of the real ones – the peak caused by the overlap. So, when it comes to the peak linkages stage, which is to follow, we will still get at least one candidate cluster centre located within each cluster, but it will not, necessarily, be particularly close to the real cluster centre.

If all of the parameters of $P(\Lambda_{\Omega_a})$ are determined by fitting to h_i over a large enough range of x, then reasonably accurate values will be returned for them all. However, since the fitting of m parameters involves performing an optimisation in an m–dimensional space, the fewer parameters which have to be found the better. To this end, the position, and height, of the highest point may be used as approximations to $\lambda_2(\Omega_a)$, and $\lambda_1(\Omega_a)$. But now, depending on the severity of the noise around x_m, there may be appreciable errors in the values, unless the histogram has a smoothing filter applied to it prior to the deconvolution.

The other point that we have to pay attention to is the selection of an appropriate value for the termination threshold. Taking the data set with

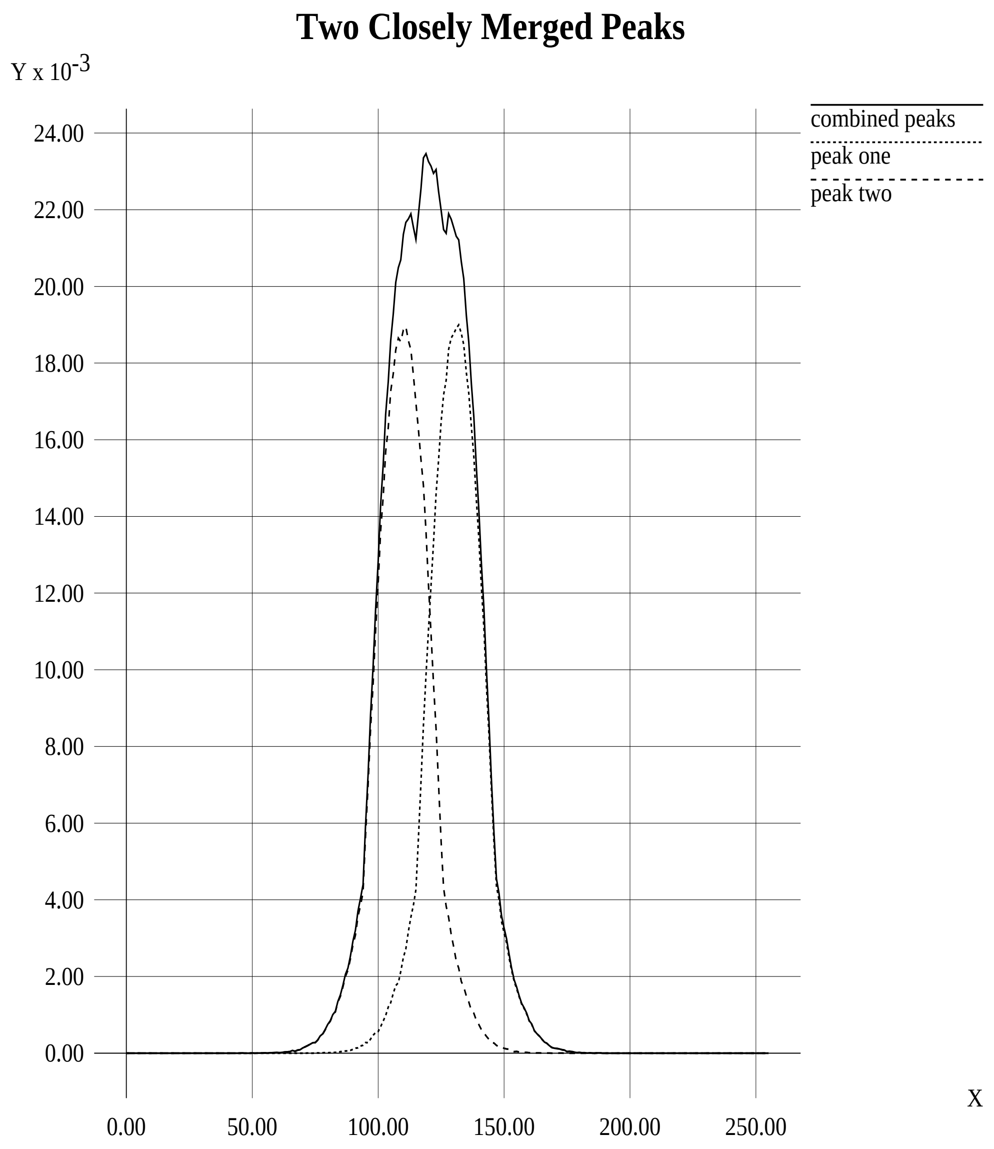


Figure 2.10: Two merged peaks, showing a false peak between them, due to overlap.

Termination threshold	Low noise case	Medium noise case	High noise case
5%	128, 152, 96	129, 152, 93, *122*,*106*, *155*	127, 152,*134*, 93, *106*,*140*,*115*
6%	128, 152, 96	129, 152, 93, *122*,*106*	127, 152,*134*, 93, *106*,*140*,*115*
7%	128, 152, 96	129, 152, 93, *122*,*106*	127, 152,*134*, 93, *106*,*140*
8%	128, 152, 96	129, 152, 93, *122*,*106*	127, 152,*134*, 93, *106*,*140*
10%	128, 152, 96	129, 152, 93, *122*,*106*	127, 152,*134*, 93, *106*
12%	128, 152, 96	129, 152, 93, *122*	127, 152,*134*, 93, *106*
14%	128, 152, 96	129, 152, 93, *122*	127, 152,*134*, 93, *106*
16%	128, 152, 96	129, 152, 93, *122*	127, 152,*134*, 93
18%	128, 152, 96	129, 152, 93	127, 152,*134*, 93,
20%	128, 152, [96]	129, 152, [96]	127, 152,*134*, [96]

Table 2.2: Locations of peaks found by deconvolution method, using various termination thresholds.

the medium separation peaks (see figure 2.2) and applying the deconvolution analysis with various termination thresholds (expressed as a percentage of the height of the first peak found) results in the peaks being located at the positions shown in table 2.2. Peaks which do not correspond to real ones are listed in italics, and peaks which are missed are listed in brackets. The peaks are listed in the order in which they were found, i.e. highest first.

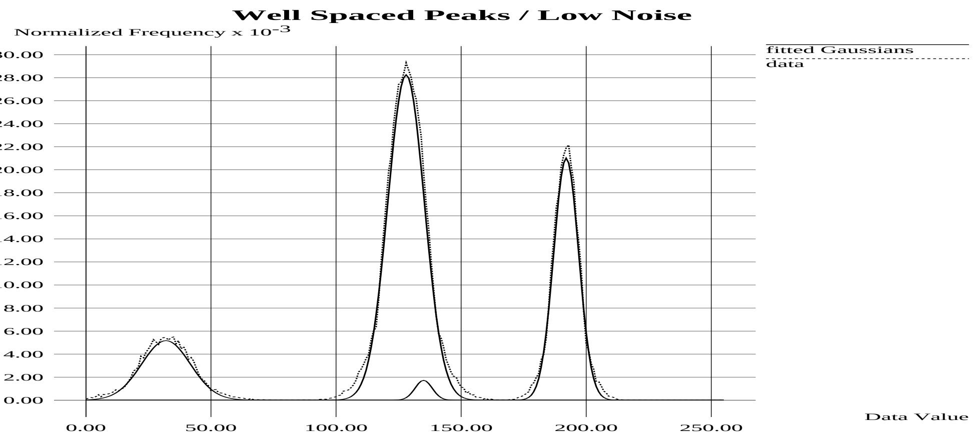
Although, from these results, it would appear that a threshold of about 18% of the height of the first peak found would be ideal, this is only because the data set used does not contain a wide range of peak heights. Since, in this work, the job of discarding peaks falls on the peak linkage stage (see chapter 3) it was decided that, if anything, the peak location stage should result in too many peaks being found, rather than risk a real, but small, peak being missed.

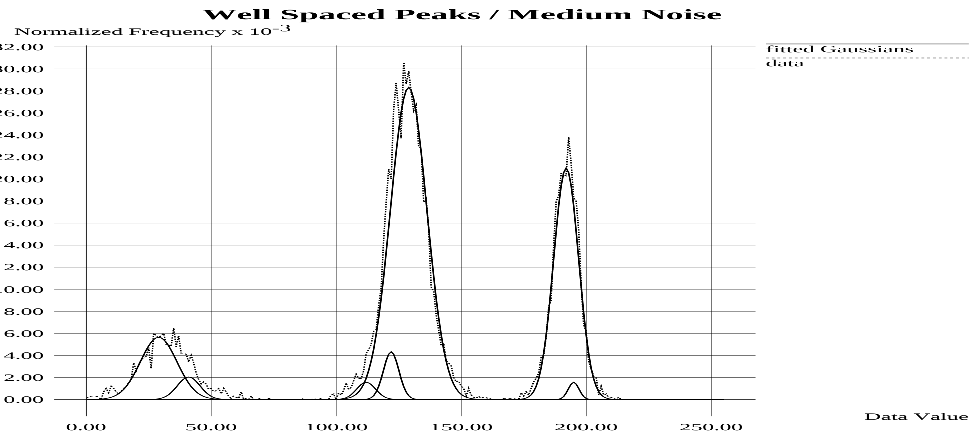
For this reason a termination threshold of 0.5% of the height of the first peak found has been used throughout this work, except when only one dimensional data is being analysed, then the threshold is set to 5% – strictly speaking the technique being outlined in this work should not be used on one dimensional data sets, but it has been necessary to do this for some of the examples and comparisons.

The result of using this technique on the same $h_i(x)$ as used in §2.1 is shown in figures 2.11 to 2.13. These results were produced using a Gaussian model for the PDF, with only the standard deviation (λ_3) as a fitting parameter. The termination threshold used was 5% of the height of the first peak found, and the region around x_m was defined as $x_m \pm 10$. Also, in each case, $h_i(x)$ was smoothed with a five point mean filter, prior to processing, in order to reduce the effects of noise.

As should be expected, this technique always finds the three peaks, even when they are quite overlapped. In none of these data sets are the peaks so close that they give rise to the problems described above. Any extra peaks that are found are partly the result of noise, and partly the result of differences between the actual and fitted PDF's. This is will be largely due to the smoothing of the histogram causing a flattening of the peaks – so that they are no longer exactly Gaussian. This problem could be reduced by either accounting for the flattening in the model used for the PDF; or by using a mean filter which has weights which are in the form of a Gaussian, rather then all the same.

When the peaks are close together, the amplitudes, and widths, which are obtained are not always correct. This is because what overlap is occuring is causing too much to be subtracted from the first peak found, so that not all of the second peak remains. This is unavoidable, unless we go back to





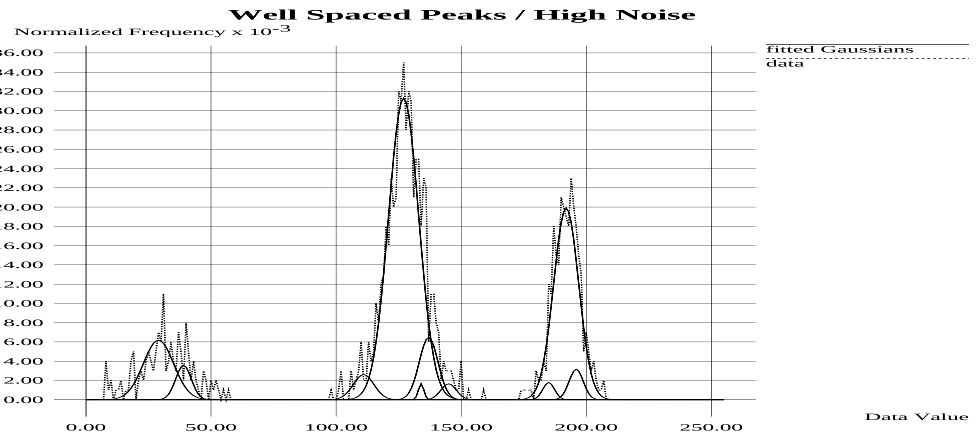
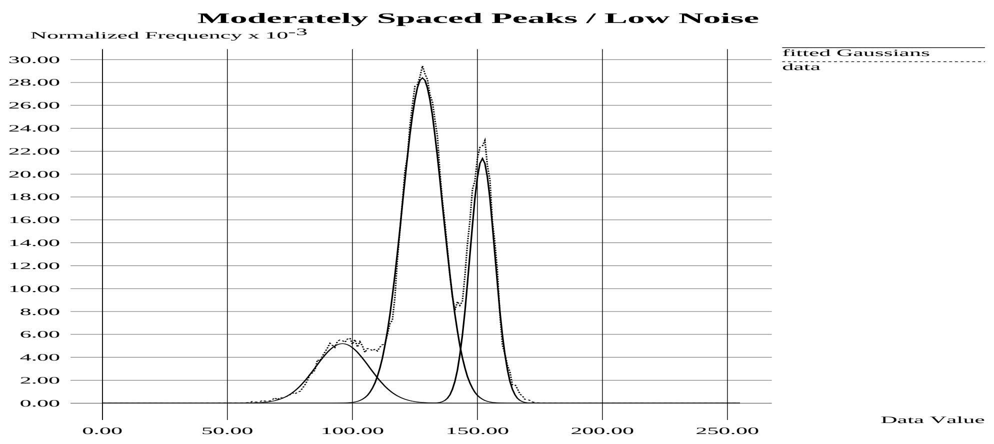
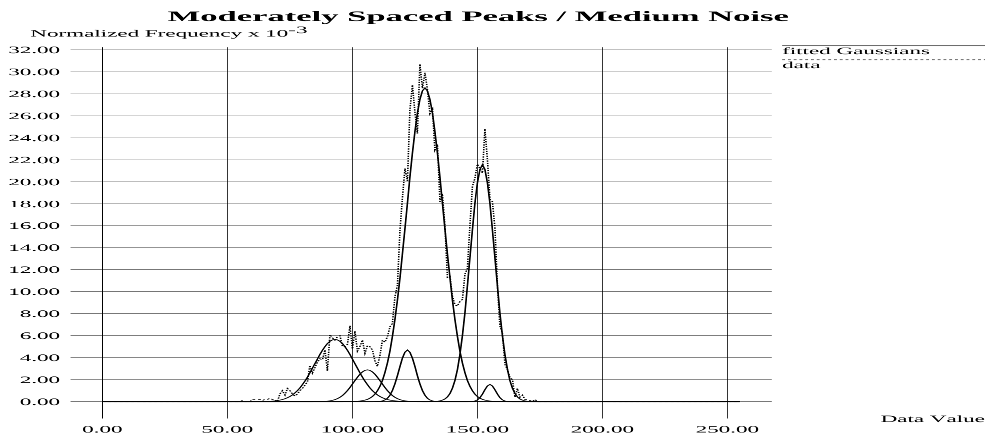


Figure 2.11: Deconvolution analysis, for widely spaced peaks, with varying degrees of noise.





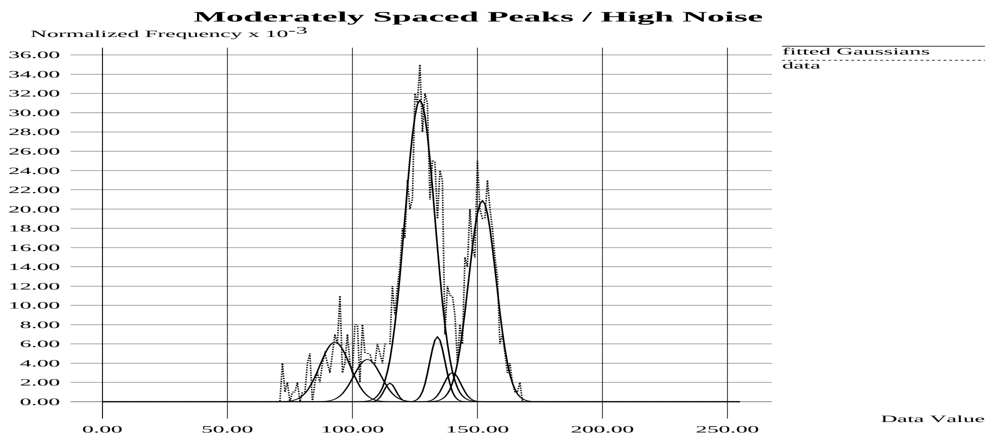


Figure 2.12: Deconvolution analysis, for moderately spaced peaks, with varying degrees of noise.

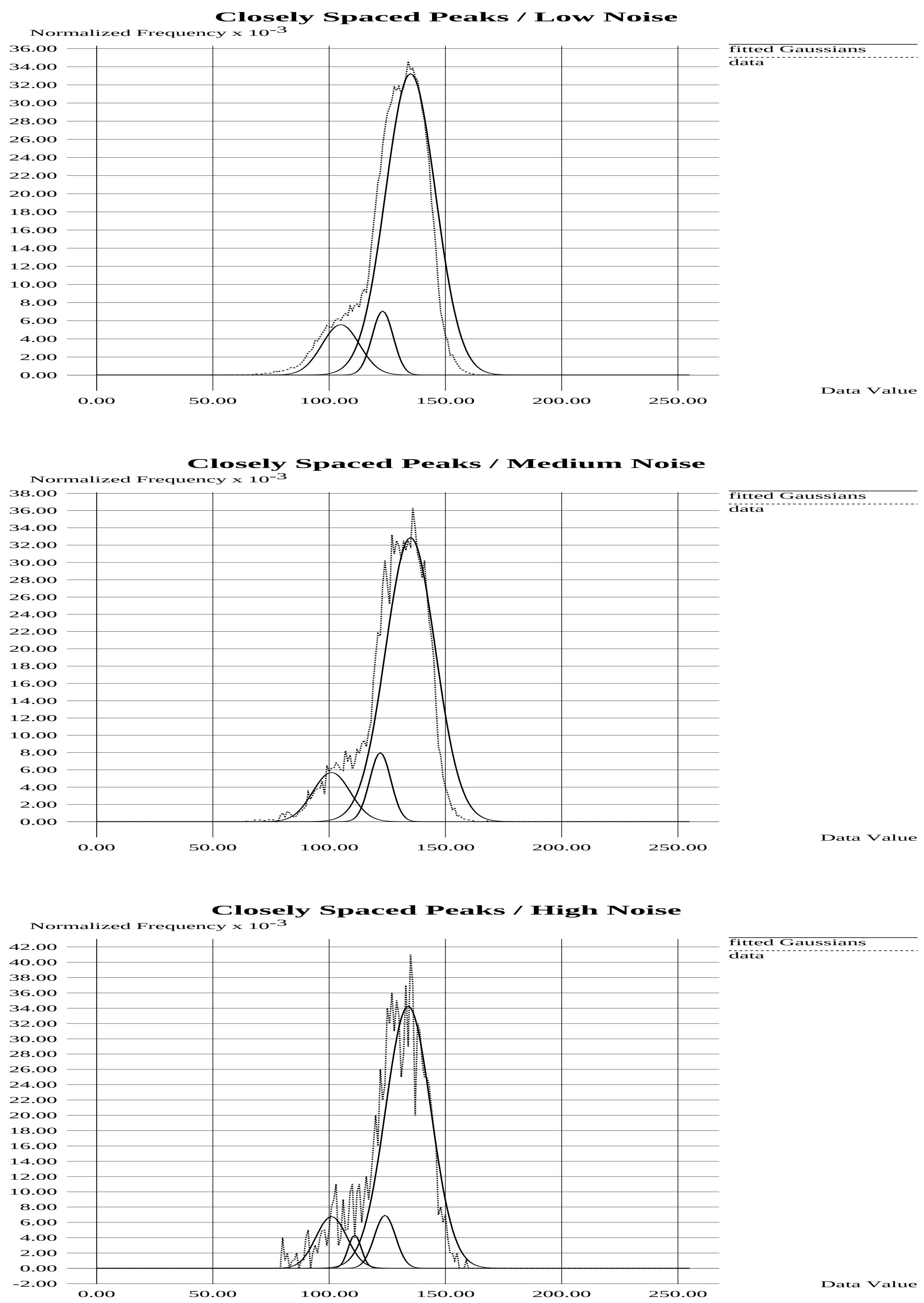


Figure 2.13: Deconvolution analysis, for closely spaced peaks, with varying degrees of noise.

subtracting fractions of peaks, and allowing the possibility of negative peaks. However, we are getting reasonable estimates for the positions of the peaks, and at least some indication of the widths – the heights are not so important. The values for the widths, and the heights (if necessary), can always be refined once we start to look at the data set in the full N–dimensional space.

In all cases the set of real peaks is a subset of the peaks found. This is vital, since it means that, when it comes to the next stage of the algorithm, linking the peaks together to form candidate cluster centres in the full N–dimensional space, we can be quite certain that there will always be at least one candidate contained in each cluster. None of the existing histogram analysis techniques, based solely on the location of extremal points, can ever hope to give us this guarantee.

CHAPTER 3
The Linking of Peaks in N–dimensional Feature Spaces

An answer to the question ‘How should the peaks be combined, in order to give the locations of the local maxima ?’ is sought. This involves considering all the possible combinations of peak positions (and a solution to the problem of dealing with large numbers of combinations is presented). Having discussed how to eliminate combinations which are not even located within clusters, the problem of determining which of the remainder are local maxima is considered.

Having found all of the peaks (i.e. the projections of the clusters) in each of the one dimensional sub-spaces we need to find out which of the peaks are associated with each of the clusters in the full N–dimensional feature space. This will then enable us to obtain the set of N–dimensional vectors which describe the positions of the cluster centres.

Working with pen and paper this would be done by extended the locations of the peaks across the two dimensional density plots and seeing which of the intersections between them occur near local maxima. See figures 3.1 to 3.3. From these figures it can be seen that there are, in general, three types of intersection : those which occur at, or near, local maxima/cluster centres (marked A); those which occur within clusters, but not near local maxima (marked B); and those which do not occur within clusters (marked C). We need an algorithm which will either select the A type, or reject the B and C types.

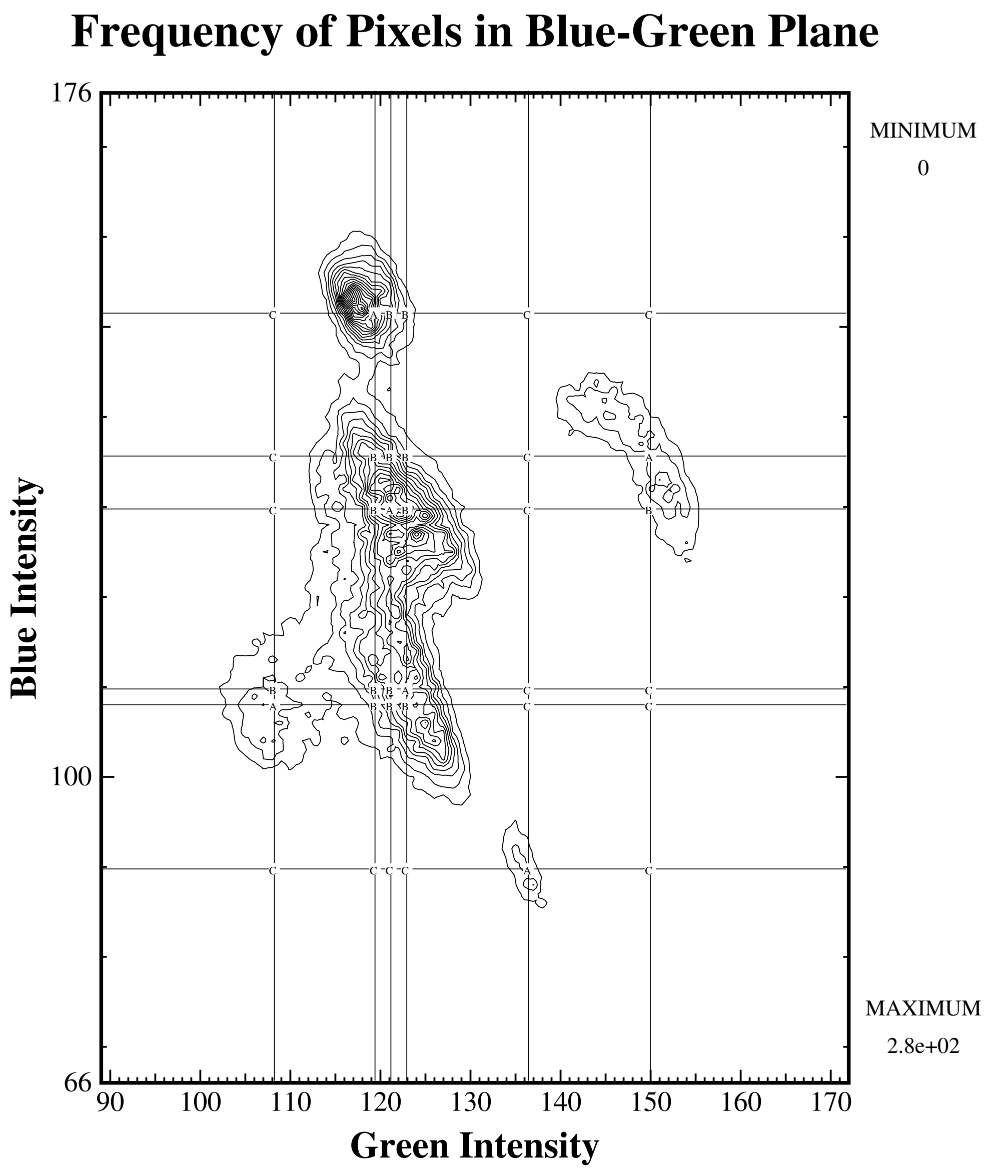


Figure 3.1: Cluster centre candidates in BBC testcard dataset, viewed down the Red axis.

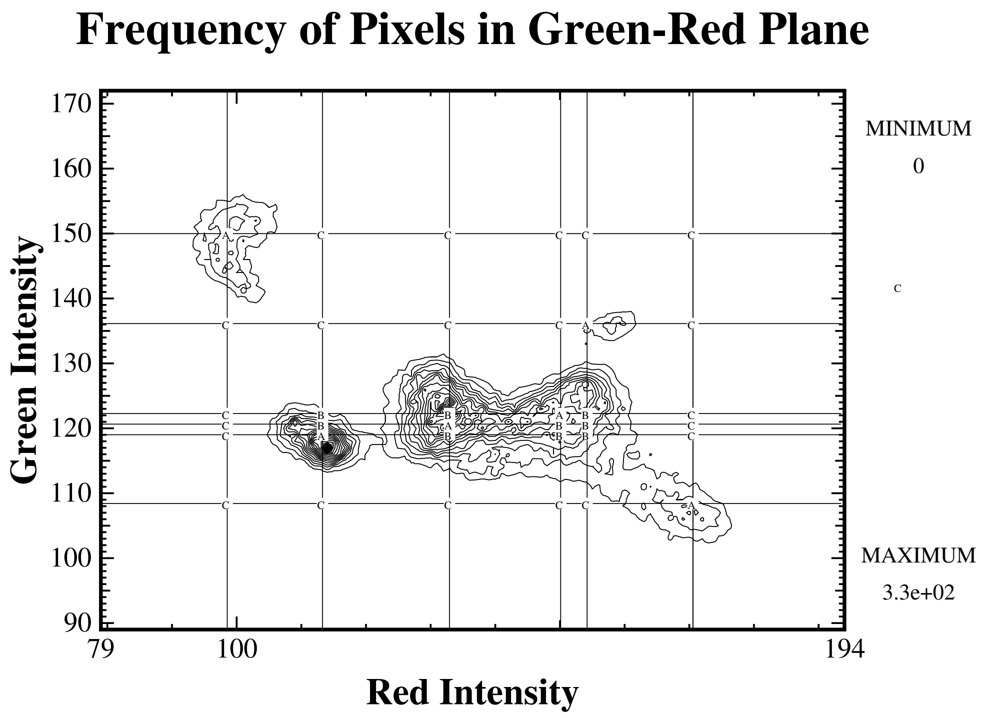


Figure 3.2: Cluster centre candidates in BBC testcard dataset, viewed down the Blue axis.

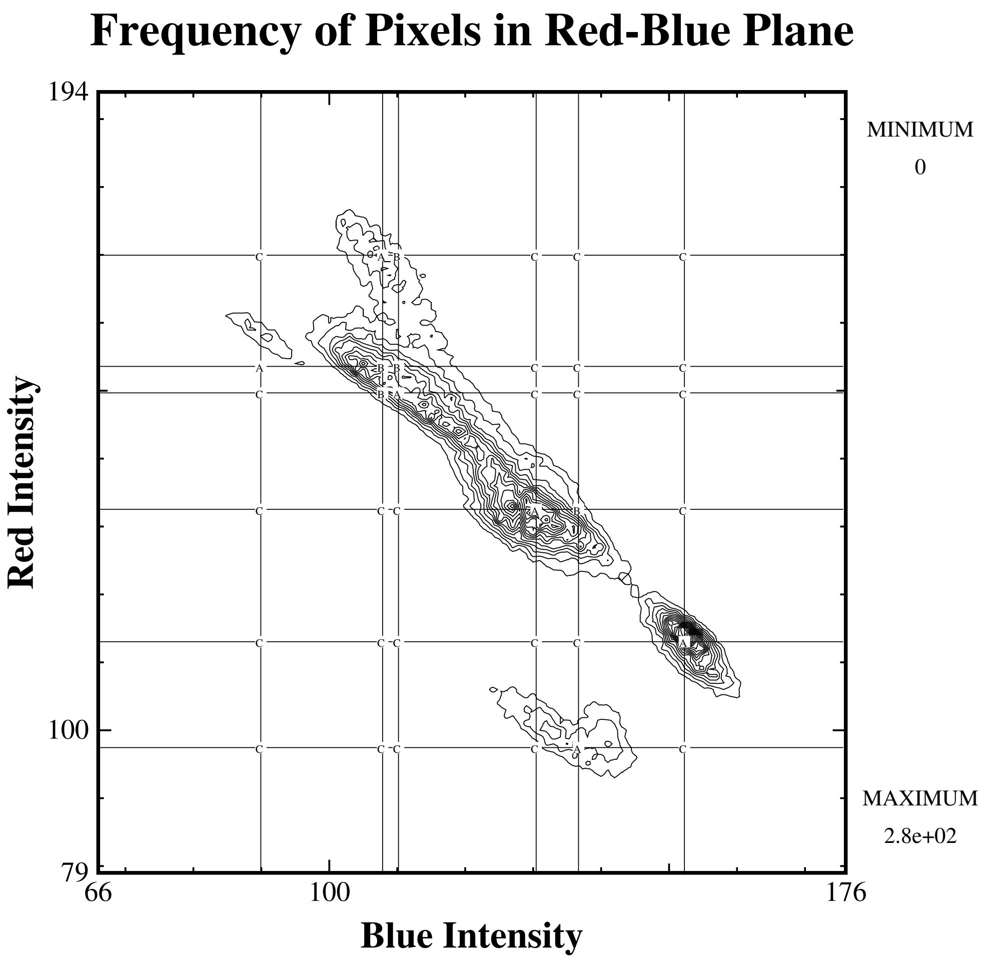


Figure 3.3: Cluster centre candidates in BBC testcard dataset, viewed down the Green axis.

However, before we can consider what form this algorithm might take, we need to overcome a problem. Originally we had b^N possible cluster locations (where b is the number of data bins in each dimension), i.e. any of the cluster centres could be anywhere in the feature space. Even for a three dimensional feature space, with 256 bins per dimension, this would come to 2^{24} possible locations, for a seven dimensional space, with the same number of bins per dimension, this increases to 2^{56} possible locations. Now we have reduced the possible locations to the number of intersections between the extensions of the peak locations into the N–dimensional feature space (or, more simply, the number of possible combinations of peak location). That is $\prod_{s=1}^{N} n_s$ (where n_s is the number of peaks found in the one dimensional sub-space s, $n_s \leq b$, always, and $n_s \ll b$, usually). Now, two, or more, clusters may give rise to a single peak, due to superpositioning, but a single cluster should never give rise to multiple peaks (assuming that the noise level is not too high), so n_s should always be less than, or equal to, k. Therefore, the upper limit on the number of combinations will be k^N. Unfortunately, this may still be a large number. For, whilst with eight clusters in a three dimensional space we only need to consider upto 2^9 combinations of peak locations, in seven dimensions the figure would rise to 2^{21} combinations (for the same number of clusters). Therefore, with more clusters, or higher numbers of dimensions, it is quite possible that we could exceed the computer's memory limits when it actually comes to implementing an analysis of the combinations.

There is also a problem in that the usual way of going through a set of combinations in a computer program would be to have a set of nested loops, one loop per set of objects which we can choose from. In our case, we need loops nested to a depth N, with the loop at level s stepping from 1 to n_s. This would be fine if N was fixed for all applications, or the program was being written in a language which allows recursion. But, in trying to write as general a program as possible, in `FORTRAN 77`, neither of these avenues is open to us, and we are faced with an, apparently, impossible programming task.

3.1 Labeling of combinations :

Fortunately, both of these problems can be overcome by assigning a unique label to each of the possible combinations of peaks. Thus, only a single loop is required (stepping through the labels), and the combinations can be considered a few at a time (where 'a few' may be anything up to a million, say, depending on the available computer memory), with only those combinations which are of interest being recorded for later use.

A simple method of labeling the combinations of peaks is to use the equation :

$$L = l_1 + \sum_{s=1}^{N-1} (\prod_{p=1}^{s} n_s)(l_{s+1} - 1) \tag{3.1}$$

where l_s is the number labeling a peak within sub-space s ($1 \leq l_s \leq n_s$).

Each possible combination of peaks is then labeled with a number in the range 1 to $\prod_{s=1}^{N} n_s$. This is an extension, into an arbitrary number of dimensions, of the standard programming technique for mapping between the indices of a two dimensional array and a one dimensional one (`n=x+((y-1)*xsize)`). In this, two dimensional, case we can think of the rows of the array being placed end to end. The `y` index (in the two dimensional space) then gives the number of complete row sections which we must move through the one dimensional array, and the `x` index gives the offset within the next row section (see figure 3.4). Table 3.1 gives an example of the application of the labeling procedure in three dimensions.

Although this equation is not analytically invertible it is computation-

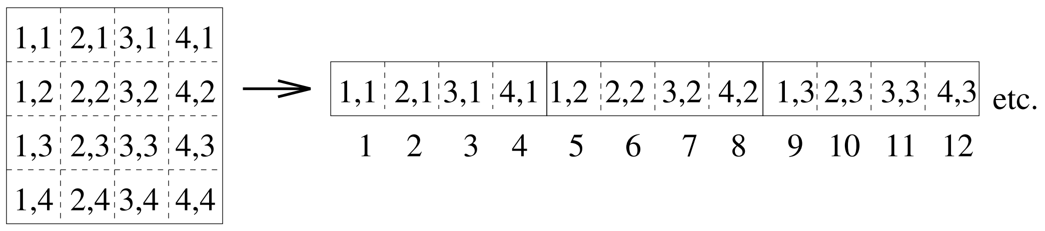


Figure 3.4: Mapping between two dimensional and one dimensional array element numbers.

Peak Numbers	Label	Peak Numbers	Label	Peak Numbers	Label
(1,1,1)	1	(1,1,2)	7	(1,1,3)	13
(2,1,1)	2	(2,1,2)	8	(2,1,3)	14
(3,1,1)	3	(3,1,2)	9	(3,1,3)	15
(1,2,1)	4	(1,2,2)	10	(1,2,3)	16
(2,2,1)	5	(2,2,2)	11	(2,2,3)	17
(3,2,1)	6	(3,2,2)	12	(3,2,3)	18

Table 3.1: Example of peak numbers and combination labels ($n_1 = 3$, $n_2 = 2$, $n_3 = 3$).

ally invertible. However, to make things easier we need to label the peaks from 0 to $n_s - 1$ (rather than 1 to n_s) and use combination labels from 0 to $(\prod_{s=1}^{N} n_s) - 1$ (rather than 1 to $\prod_{s=1}^{N} n_s$). Then equation 3.1 becomes :

$$L' = l'_1 + \sum_{s=1}^{N-1} (\prod_{p=1}^{s} n_p) l'_{s+1} \tag{3.2}$$

where $L' = L - 1$ and $l'_s = l_s - 1$, but n_p is still the number of peaks in sub-space p **not** the maximum peak label number (which is now $n_p - 1$).

Now, the component of the combination label which deals with the first $N - 1$ dimensions must be smaller than the possible number of combinations in those dimensions, so :

$$l'_1 + \sum_{s=1}^{N-2} (\prod_{p=1}^{s} n_p) l'_{s+1} < (\prod_{p=1}^{N-1} n_p) \tag{3.3}$$

Also :

$$L^{'} = l_1^{'} + \sum_{s=1}^{N-2} (\prod_{p=1}^{s} n_p) l_{s+1}^{'} + (\prod_{p=1}^{N-1} n_p) l_N^{'} \tag{3.4}$$

(expressing the last step of the summation in equation 3.2 explicitly). This means that the following equation will hold true for integer division :

$$l_N^{'} = \frac{L^{'}}{(\prod_{p=1}^{N-1} n_p)} \tag{3.5}$$

The remainder from this division will contain just the contributions to the combination label from the first $N-1$ dimensions, and so, since equation 3.3 still holds for $N \leftarrow (N-1)$, the whole inversion process can be described by :

$$l_s^{'} = \frac{L_s^{'}}{(\prod_{p=1}^{s-1} n_p)} \qquad (s = N, N-1, \ldots, 2, 1) \tag{3.6}$$

where $L_s^{'}$ is the partially inverted combination label :

$$L_s^{'} = \begin{cases} L^{'} & (s = N) \\ L_{s+1}^{'} - (\prod_{p+1}^{s} n_p) l_{s+1}^{'} & (s = N-1, N-2, \ldots, 2, 1) \end{cases} \tag{3.7}$$

Thus, given a combination label, L, we can always determine which set of peaks is associated with it. There will still be a limit to how many combinations can be considered, however, since the maximum value of L is limited by the largest number that can be stored as an integer by the programming language being used. This may present a problem for very high dimensional feature spaces containing large number of clusters.

3.2 Discarding voids :

Having labeled all of the possible combinations of peak locations, we can now consider them to be a list of candidate cluster centres. We now need to go through this list and apply a set of rules to determine which of the candidates are of type A, i.e. actual cluster centres.

The most obvious rule, and the easiest to implement, is that the candidate vector must lie within a cluster, i.e. it should not be of type C. The simplest way to employ this rule is to see whether or not the density of data points in the vicinity of the candidate is greater than some, small, threshold. If it is not, then the candidate lies in the void between the clusters, and cannot, therefore, be a cluster centre.

There are several possible definitions of ‘in the vicinity’ which we might use. Figure 3.5 shows, in two dimensions, the possibilities : hyper-cuboids, or hyper-ellipsoids; of either fixed size, or of a size determined by the proximity of neighbouring candidates. Of these, using fixed sized hyper-cuboids is the least computationally intensive. But the division of the feature space into hyper-cuboids of variable size (in such a way that the whole feature space is covered, with no overlap) is used in the program, since this removes the need to determine how large a fixed size hyper-cuboid should be used for the best results. Also, this is the only approach which ensures that every data point is assigned to exactly one candidate.

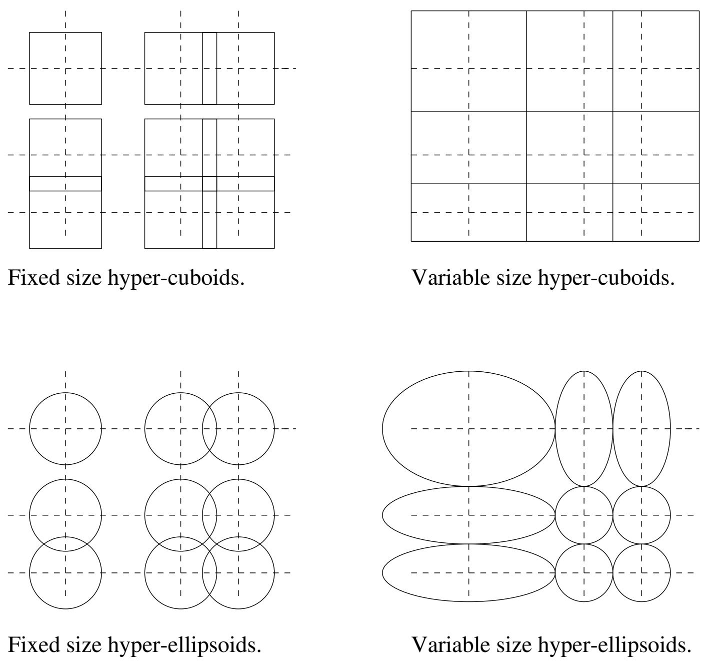

Fixed size hyper-cuboids.
Variable size hyper-cuboids.
Fixed size hyper-ellipsoids.
Variable size hyper-ellipsoids.

Figure 3.5: Possible definitions for the ‘neighbourhood’ of a candidate vector.

Ideally the threshold should be set to zero, so that only those candidates which lie in totally unpopulated parts of the feature space are discarded. However, in practice, a small, but non-zero, threshold will be used. This has the advantage that some of the type B candidates which are in the extremities of clusters will be discarded, as well as all the type C ones, and it enables us to set a limit on the smallest size of cluster which we are interested in detecting. For the purposes of the program used in this work the threshold is set to 0.025% of the total number of data points, or 200 data points, whichever is the larger.

3.3 Determination of local maximality :

The application of our first rule should have considerably reduced the number of candidate cluster centres. In the ideal case (compact, well separated, clusters) the remaining candidates would actually be the cluster centres, i.e. there would be just k of them. However, it is likely that one or two candidates, to either side of each of the cluster centres in each dimension, will have survived. If we say that in a bad, but not necessarily the worst, case we still retain all of the candidates which are neighbours of each of the actual cluster centres, then we will have $3^N k$ candidates remaining. For our eight cluster example this would mean roughly $2^{7.8}$ candidates remaining (down from 2^9) in the three dimensional case, and roughly $2^{14.1}$ (down from 2^{21}) in the seven dimensional one. The worst case would, of course, be to still have all the candidates that we started with.

Now we need a second rule to distinguish between the type A candidates and the type B ones. This is provided by their definitions – type A candidates are at, or near, local maxima in the density distribution, and type B candidates are not. The same rule could, of course, be used to distinguish between type A candidates and type B & C candidates directly, without the need for our first rule. But, the second rule will be far more computationally time consuming than the first, so it is best to try and minimise the number of candidates to which it will be applied.

One possible method for determining whether or not a candidate vector satisfies the second rule is to use it as the starting point for a steepest ascent algorithm [Polak, 1971]. At each step we can picture a 'climber' considering all of the points adjacent to that on which it currently 'stands', it then moves to the highest, in the hope that this will eventually lead it to a maximum. In our case, if the climber manages to get too far away from the starting point before reaching a 'summit', then the starting point is not at, or even near, a local maximum. This has the nice advantage that it automatically gives us a refinement of the estimates of the cluster locations.

However, there are several problems with this technique. The first is that, at each step of the steepest ascent, $3^N - 1$ comparisons have to be made. This makes the algorithm increasingly unsuitable as N gets larger. The problem can be alleviated, somewhat, by only allowing the climber to move parallel to the axes of the feature space (i.e. no diagonal moves). This reduces the number of comparisons to $2N$, which is rather more manageable. See figure 3.6.

The second problem is that, for N greater than two, or possibly three, the feature space cannot be stored in the computer's memory in its entirety (as was discussed in §1.3). For slightly larger N we could get away with only constructing the region of the feature space about which the climber is to be allowed to 'roam'. But this would have to be done separately for each candidate cluster centre, and it is still not suitable for arbitrarily large N.

One way around this storage problem would be to run the steepest

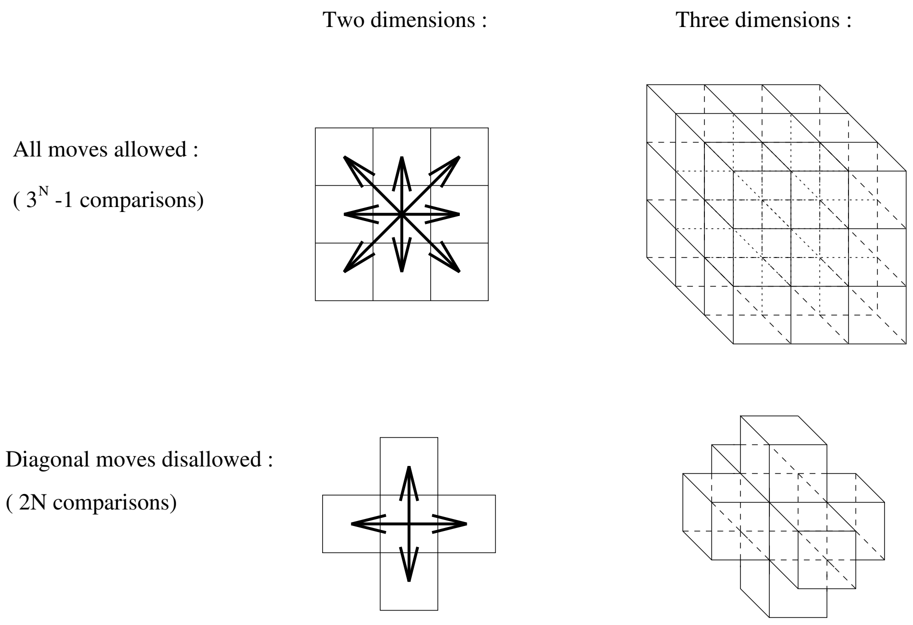


Figure 3.6: The number of locations a ‘climber’ has to look at in the steepest ascent algorithm.

ascent algorithm in each of the possible two-dimensional sub-spaces of the feature space. Provided that the number of data bins is not too large, each of these sub-spaces can reside in the computer’s memory (one at a time). Once the local maximum has been reached in one of the sub-spaces the climber will set off again from the projection of its current position into the next sub-space. See figure 3.7.

Unfortunately, the number of possible two-dimensional sub-spaces is $\frac{N(N-1)}{2}$, and, for large N, this number will become prohibitively large. A solution to this would be to use only the N sub-spaces $s = \{1,2\}, \{2,3\}, \ldots, \{N,1\}$. However, once we start considering the sub-spaces individually, there will be instances when the climber gets ‘side-tracked’ in one of the sub-spaces, and there is no ascending path in any of the remaining sub-spaces which will allow the error to be corrected. In the example shown in figure 3.8, when we come to consider the sub-space BC all the candidates will be drawn towards the centre

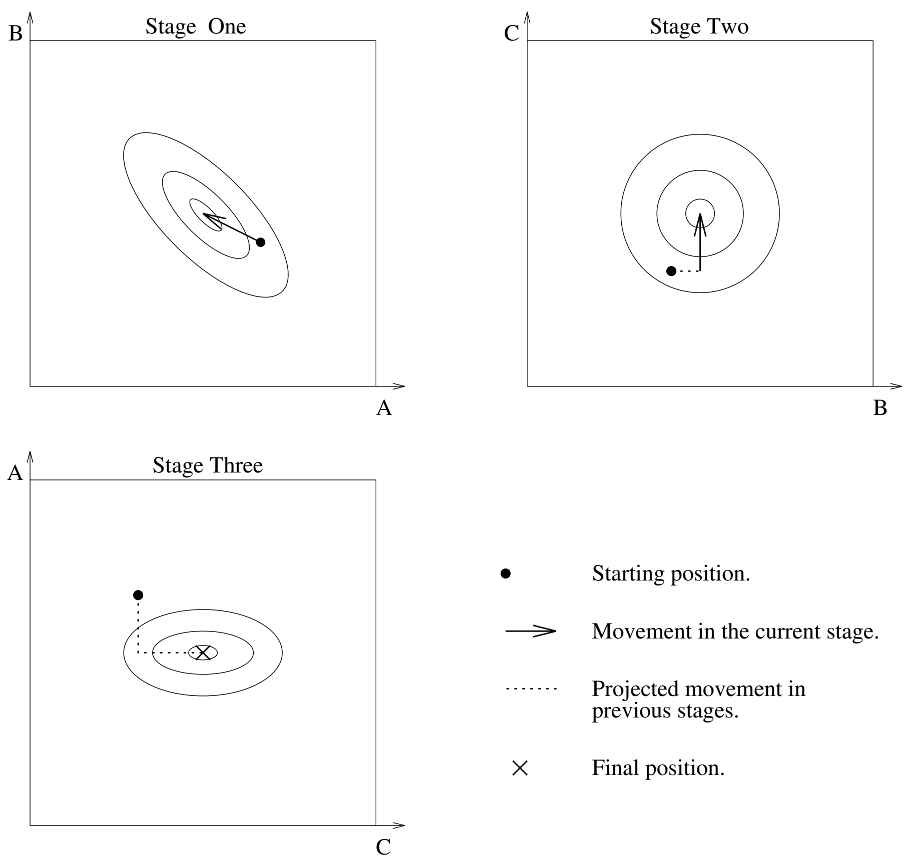


Figure 3.7: The movement of the 'climber' in the modified steepest ascent algorithm.

of the merged clusters. This has no effect on case *a*, but case *b* shows how a candidate could be erroneously discarded for having travelled too far from its start point, and case *c* shows how a candidate can still survive despite having migrated from one cluster to another – thus duplicating a cluster centre that has already been found, whilst leaving the original cluster undetected.

Finally, any optimisation type approach will suffer from the problem that the peaks which the climber is expected to ascend are not going to be very smooth (only a very large dataset would have sufficiently good statistics for this to be so). This will result in the climber getting stuck on local sub-maxima, and, hence, providing a false positive result for the test of local maximality. This could be overcome by either smoothing the data, or modify-

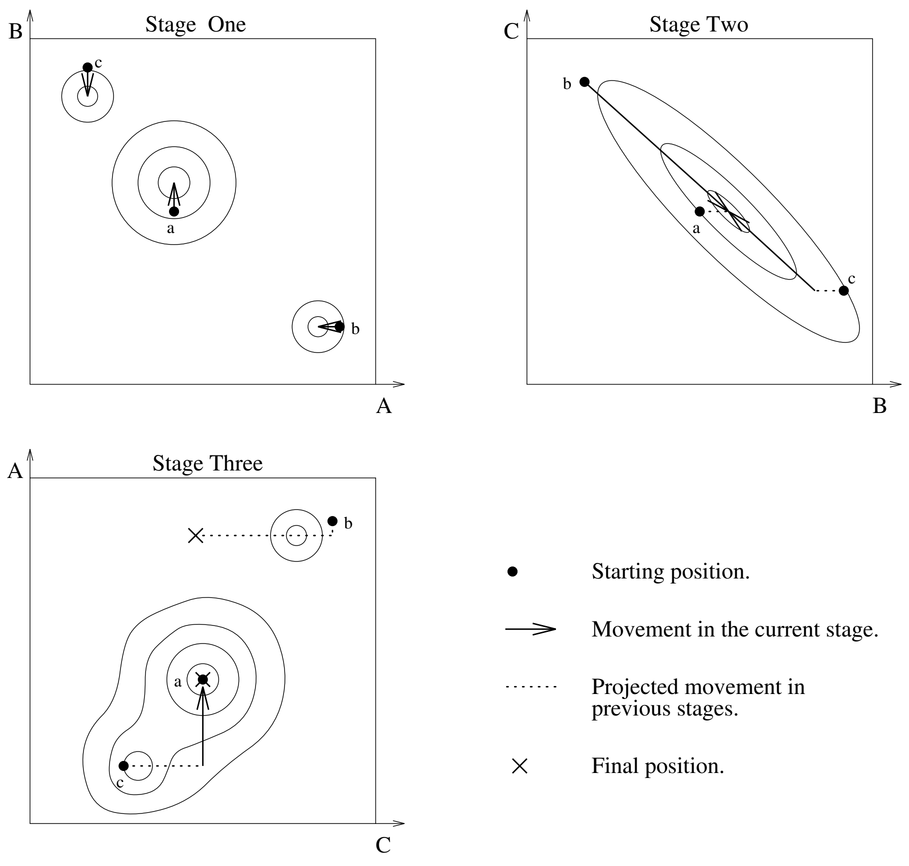


Figure 3.8: Examples of the 'climber' getting sidetracked in the modified steepest ascent algorithm.

ing an optimisation algorithm which is very good at finding global maxima so that it finds local maxima, but does not stop at the sub-maxima. For example, we could probably produce a scheme for controlling the 'temperature' of the 'simulated annealing' algorithm [Kirkpatrick, et al., 1983] that would do this.

Although each of these possible approaches has only been presented here in a theoretical way, all of them, with the execption of simulated annealing, were attempted in practice, and the problems described were all encountered when working with real data. Also, so far, we have ignored the situation, described in chapter 2, where very closely spaced peaks, in the one dimensional analysis, could result in a single off-centre candidate being found for a cluster. The whole premise of using an optimisation approach was that the candidates which we want to preserve would always be near the local maximum, but we have already seen that this may not necessarily be the case. For these reasons it was decided to try looking in a completely different direction for a way of implementing the test of local maximality.

3.4 The 'IKOC' algorithm :

The implementation of the second rule which has been devised for this work could be referred to as the 'I'm the King of the Castle' (IKOC) approach. Quite simply, we take the candidate vector which most nearly coincides with the global maximum in the density distribution and compare the density of data points at this position in the feature space with that at the positions of the neighbouring candidates. Any neighbouring candidate which has a lower density of data points associated with it is replaced by the central candidate (at this, first, stage this will be all of them). We then take the candidate vector which has the next highest density of data points associated with it (ignoring those candidates which have been replaced) and repeat the process (with replacement candidates used instead of the neighbours, if they exist). This is repeated for all of the remaining (non-replaced) candidates.

We, of course, need to define the set of neighbouring candidates. These may be just those which are more similar to the one under consideration than any other. Or, better still, they could be those which lie within a certain range of similarity about the candidate.

The important thing is that the neighbourhood is not too large, otherwise the centres of small clusters will be discarded if they are too close to larger ones, or too small, otherwise not all of the type B candidates will be discarded. An example of the IKOC process, using idealised neighbourhoods, is shown in figure 3.9. Candidate 7 would be discarded as a result of the first

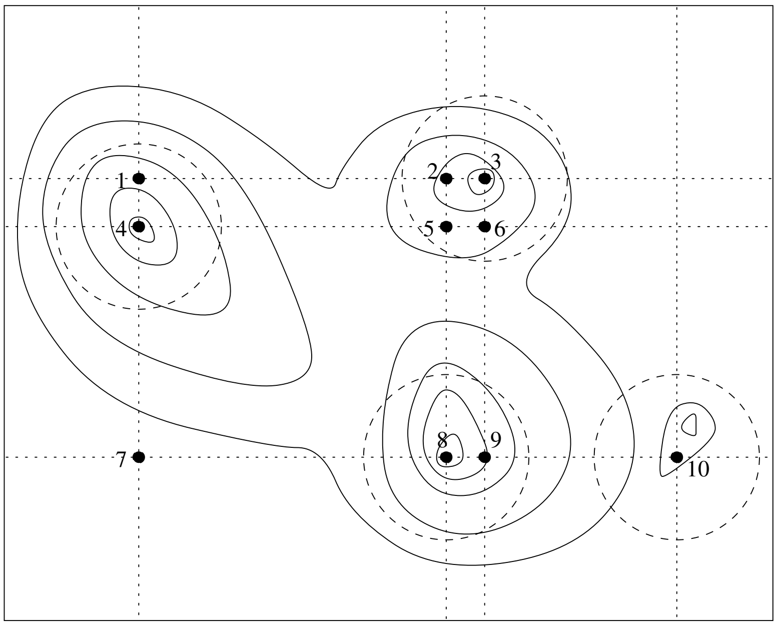


Figure 3.9: Idealised example of the operation of the IKOC algorithm.

rule, candidate 1 will be discarded as being sub-maximal in the neighbourhood of candidate 4, then candidate 9 will be discarded as being sub-maximal in the neighbourhood of candidate 8, and, finally, candidates 2, 5 & 6 will be discarded as being sub-maximal in the neighbourhood of candidate 3. This leaves candidates 3, 4, 8 & 10 as the actual cluster centres. Although some sort of refinement of the locations will need to be performed before candidate 10 actually coincides with its corresponding cluster centre.

In the program, the data are assigned to the nearest candidate vector, and the 'transformed divergence' [Swain & Davis, 1978] between each pair of vectors is calculated :

$$d_{IJ}^T = 2(1 - e^{-d_{IJ}/8}) \tag{3.8}$$

where d_{IJ} is the divergence between the sets of data points I and J :

$$\begin{aligned} d_{IJ} &= \frac{1}{2} T_r\{(\Sigma_I - \Sigma_J)(\Sigma_J^{-1} - \Sigma_I^{-1})\} \\ &+ \frac{1}{2} T_r\{(\Sigma_I^{-1} + \Sigma_J^{-1})(\mathbf{m}_I - \mathbf{m}_J)(\widehat{\mathbf{m}_I - \mathbf{m}_J})\end{aligned} \tag{3.9}$$

where Σ_I and $\mathbf{m}_I$ are the covariance matrix and mean vector of I, respectively, and $T_r\{\}$ is the matrix operation 'trace', i.e. the sum of the terms on the leading diagonal.

The neighbours of a candidate are then those for which the transformed divergence is less than some threshold. Now, since the value of d^T_{IJ} saturates at 2.0 near the edge of a cluster, and, since we only want to include in a neighbourhood those candidates which lie within the same cluster, ideally the threshold value should be just less than 2.0. However, in the program which implements this work, approximations are used in the calculation of the covariance matrices, such that a value of, say, 1.95 will often result in candidates being falsely discarded, so the threshold of 1.7 was used for all the results presented in this work.

This approach has not proved to be ideal (see chapter 7), but it is faster, and more reliable, than any of the optimisation based techniques. Also, if there is only one candidate present in a cluster, the IKOC algorithm will not discard it, no matter how far it is from the real centre.

After the second rule has been applied to all of the candidate vectors, the few which remain are considered to coincide with the centres of the clusters of data points in the feature space. Some adjustment may be necessary depending on the desired definition of 'centre' – in the program the data points are each assigned to the nearest remaining candidate and the mean positions of the data points in each cluster determined. The number of surviving candidates should be equal to the number of clusters (k), removing any need for iteration or backtracking, as required by the existing cluster analysis techniques.

These vectors may then be used for partitioning, or otherwise characterising, the data.

CHAPTER 4
Multi-spectral Image Segmentation

An introduction is made to multi-spectral imagery, and the purposes of image segmentation. The problems caused by shading in images is discussed, and a generalised method for its removal, from multi-spectral images, is presented. The use of cooccurrence matrices in image segmentation, and edge detection, is reviewed. Finally, the new algorithm is used as part of an image segmentation technique based on the location of clusters in a multi-dimensional cooccurrence space, and some results are presented.

4.1 Multi-spectral images :

A multi-spectral image is one which is quantised not only in its spatial coverage ('pixels') but also in its spectral coverage ('bands'). Whilst it is possible to have a two band multi-spectral image, the simplest one is usually taken to be composed of three bands, an example of such a three band image is that produced by a video camera. The three bands correspond roughly to the three regions of the visible spectrum to which the three colour receptors of the human eye are sensitive (blue: 0.40–0.50 μm; green: 0.50–0.60 μm; red: 0.52–0.65 μm). However, for technical reasons, a video camera does not actually record using the red/green/blue (RGB) system, but a luminance value and two chrominance values which are linear combinations of R, G, & B [Wright, 1969].

Other imaging systems utilise more (sometimes many more) than three bands, covering not only the visible region of the spectrum but also going into the infra-red and, more rarely, the ultra-violet. For example, the multi-spectral scanner (MSS) on the LANDSAT 1, 2, & 3 Earth observation satellites (see table 4.1), and the thematic mapper (TM) on LANDSAT 4 & 5 (see table 4.2). At the more extreme end there are the, so called, 'imaging spectrometers' which operate with up to around 256 spectral bands. To date their use has been limited to airborne observations, but it is only a matter of time before one is included as part of a spaceborne system.

Band	Wavelength (μm)	Resolution (m)
4	0.50–0.60	80
5	0.60–0.70	80
6	0.70–0.80	80
7	0.80–1.10	80
8†	10.40–12.50	120

Table 4.1: LANDSAT 1, 2, & 3 MSS characteristics (†- LANDSAT 3 only).

Band	Wavelength (μm)	Resolution (m)
1	0.45–0.52	30
2	0.52–0.60	30
3	0.63–0.69	30
4	0.76–0.90	30
5	1.55–1.75	30
6	10.40–12.50	120
7	2.08–2.35	30

Table 4.2: LANDSAT 4, & 5 TM characteristics.

Synthetic aperture radar (SAR) systems extend the spectral coverage into the microwave region ($\sim$ 3 cm), and, with the launch of the ERS-1 satellite, the routine availability of SAR images should see their increasing use for remote sensing [NRSC pamphlet G 06]. However, by default, SAR systems produce images whose coordinates are azimuth and 'slant-range' (distance from the SAR system along the line of sight), rather than the usual azimuth and projected 'ground-range' (distance from the SAR system along a terrainless ground plane, or geoid). This means that a SAR image has to be converted to azimuth/projected ground-range (geocoded) before it can be sensibly combined with images from the more common sensors.

This process of co-referencing data from different sensors can, in fact, be applied to data obtained by non-imaging processes, thus allowing their inclusion as 'non-spectral' bands in a multi-band image. For example, data obtained from laser range-finding or magnetic field measurements, or even from actual 'in the field' examination of parts of a scene being imaged. The multi-band image thus becomes part of a data fusion process.

4.2 The information contained in multi-spectral images :

As will be discussed in more detail in §4.4, one can make the (slightly simplistic) generalisation that the intensity component of a multi-spectral image of a 'natural' scene contains information about the shape of the objects in the scene, and that the colour component contains information about the identity of the objects. For example: grass is green; the sky is blue; and oil bearing shales are dark in the visible part of the spectrum, and slightly brighter in the near infra-red. This generalisation breaks down most when man-made objects are introduced into the scene. Such objects are rarely characterised by their colour (not all biscuits are yellow, and not all large red objects are London Transport omnibuses).

Chapter 5 contains a fuller description of the information to be gleaned from multi-spectral images of 'natural' scenes in the field of remote sensing.

4.3 The purpose of segmenting multi-spectral images :

The segmentation (or classification) of an image involves dividing it up into 'regions' containing pixels which all exhibit the same characteristics. The regions are then labelled in such a way that regions whose pixels share the same characteristics have the same label (they are in the same 'class').

This process can be seen as either one of data reduction (see chapter 6), or feature extraction (see chapter 5). Data reduction is an end in itself, but feature extraction is an important first stage in the quest to 'understand' a scene.

If the attempt to understand the contents of an image is being made by a computer program, then the feature extraction converts the image data into a form which the program can more easily 'comprehend'. That is, one is transforming from a set of features which is based purely on pixel values to one which is based on such things as: the colour of a region; a parametric representation of the shape of a region; and details of one region's relationship to those around it (e.g. does it surround, or is it surrounded by, another region ?).

The idea of easing comprehension is also important in the case where the image is to be analysed by a human being. Because of the limitations of the human eye, there is no way of making a combined presentation of more than three spectral bands to a human analyst at any one time. The usual ways of getting around this problem are to either make a false colour composite of the three spectral bands which are most relevant to the problem in hand, or to perform a 'principal components' transformation on the data [Gonzalez & Wintz, 1987]. The transformed data is such that most of the variation in the data is contained within the first few transformed bands. The first three of these are then false colour composited for analysis.

A segmented image, however, contains information from all the spectral bands in, what is effectively, a single band image (the class labels). This can be pseudo-coloured to help distinguish between regions, and the spectral characteristics of the various classes can be provided in tabular form.

The importance of performing feature extraction prior to attempting to interpret the data cannot be over estimated. It is not merely a question of making the interpretation easier, particularly where computer interpretation is involved, it is a question of making it possible at all. For the interpretation process to work properly there must be both a ‘low level’ feature extraction stage, and a ‘high level’ analysis of the relationships between the features resulting in the recognition of complex objects. Whilst it is still possible to make, limited, sense of the data if the second stage is either missing or of limited functionality, if there are problems with the first stage then, at best, we will have an erroneous interpretation of the data, or, at worst, no concept of the data at all. The title essay in “The Man Who Mistook his Wife for a Hat” [Sacks, 1986] gives an example of a man who is only able to do visual feature extraction. He is still capable of carrying on a fairly normal life, however, even though, for example, he might see something as a ‘convoluted red object with a linear green attachment’ without recognising it as a rose.

4.4 Processing of multi-spectral images prior to segmentation :

As was stated in §4.2, one can consider the intensity component (the sum of the spectral bands) of a multi-spectral image to be due to the shape of the objects in a scene. Indeed, this is the basis of the so-called ‘shape from shading' techniques which aim to get three-dimensional information from non-stereo two-dimensional images [Horn, 1986]. Not only does the intensity depend on the (gross) shape of the objects, but it also depends on the texture of their surfaces. Rough surfaces will show a much wider local variation in intensity than smooth ones.

Both of these effects are something of a nuisance when attempting to segment a multi-spectral image. One does not want the parts of an object facing away from the light source to be put in a different class to those which face towards the light. And, one does not want a rough surface to be broken up into a collection of small regions spread over several classes. Therefore, it would be advantageous to be able to remove these 'lighting effects' prior to starting the segmentation. Of course nothing can be done about regions of the image which are in complete shadow (i.e. the value of all of the spectral bands is at, or near, zero), there is just no information available on which to base any corrective measures.

If one knows the shape of all the objects in a scene, and the positions and characteristics of all the sources of illumination, then one can, in theory, determine what the lighting effects are for an image, using the reverse of the shape from shading techniques. However, in practice, the necessary calculations are prohibitively complex. If simplifying assumptions are made (e.g. single scatter, Lambertian scattering), the results are of limited use (see Forsyth & Zisserman [1989] for a discussion of the effects of multiple scattering on shape from shading results).

In most cases, though, only the image data itself will be available. This does not mean that the lighting effects cannot be dealt with, but it does mean that simplifying assumptions **have** to be made if any progress is to be made.

In his 'retinex' theory Land [1977] makes the assumption that smooth variations in the intensity of an image are due to lighting effects, and that rapid variations are due to changes in the objects themselves (i.e. the reflectivity of the scene). This approach allows lighting effects to be removed from even mono-spectral images. Put as simply as possible, one follows a track across the image ignoring any change in intensity (from one pixel to the next) which is less then some threshold. If there is a change which is greater than the threshold, then all the pixels between this abrupt change and the last one (or the start of the track, if this is the first abrupt change) are set to their mean value. Once the end of the track is reached all the pixel values are renormalised so that the largest intensity encountered along the track is scaled to some predefined value – this is because we cannot assume any of the pixels to be unshaded. Figure 4.1 shows the result of applying the retinex algorithm in an idealised case, similar to the intensity variation across the middle of the synthetic image in figure 4.2.

This aproach has three important shortcomings. Firstly, it is not obvious how to arrange the track (or tracks) so as to remove the lighting effects from the whole of a two dimensional image (Land was only interested in how the eye determined the 'true' colour of individual points in an image). Secondly, it assumes that the surfaces of the objects, in a scene, are fairly smoothly varying. It would, for example, assume that the change in intensity between two faces of a cube, lit from one side, is due to a change in the reflectivity of the surface of the object, not due to lighting effects. Finally, the renormalisation process is thrown into disarray if a light source is included in the image [Horn, 1986]. This may occur through the direct imaging of the source, indirect imaging of the source via specular reflection, or (as in the case of thermal infra-red images) the objects in the scene may be self-illuminating.

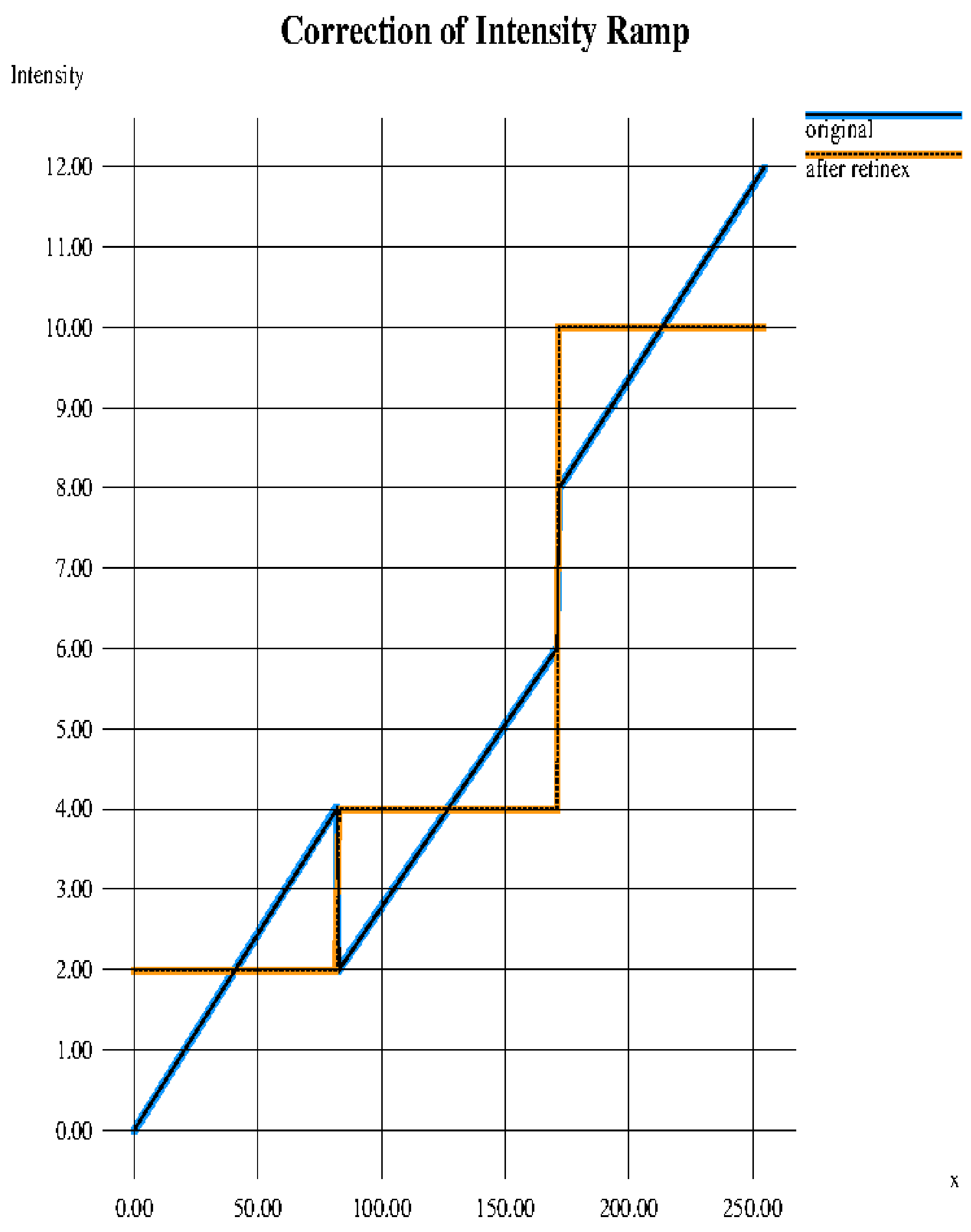


Figure 4.1: Idealised application of the retinex algorithm to the removal of lighting effects.

An alternative assumption which can be made is that the intensity variations are **entirely** due to lighting effects. One then only needs to convert the image to one which has uniform intensity, to remove all the lighting effects.

The simplest way of doing this is to divide each spectral band by the sum of all the spectral bands (the intensity), this results in an image with a uniform intensity of unity (each band may then be multiplied by a constant scaling factor, if necessary). Where all the spectral bands take a value of zero their new values should all be set to the inverse of the number of spectral bands, for consistency. Although quick, and simple to implement, this technique will break down in regions of the image which are almost, but not quite, completely black. If all but one of the spectral bands take a value of zero the processed image will have that spectral band set to the highest possible value, and all the rest to zero. This means that any noise in, what should be, a completely black region, will result in the processed image containing a random mixture of pure 'colours' (spectral responses), as in figure 4.2.

A better way to produce an image of uniform intensity is to transform the data from the set of spectral bands to a set of bands wherein one band contains only the intensity information, and the other bands contain only the colour information, shared between them. The intensity band is then set to a constant value, and the inverse transform applied. See figure 4.3.

With three band (RGB) images the transformation normally used to get all the intensity information into one band is the RGB↔HIS transform. A possible set of expressions for this are :

$$\begin{aligned}
H &= \tan^{-1}[\frac{\frac{1}{\sqrt{6}}R - \frac{2}{\sqrt{6}}G}{-\frac{1}{\sqrt{6}}R - \frac{1}{\sqrt{6}}G + \frac{2}{\sqrt{6}}B}] \\
I &= \frac{1}{3}[R+G+B] \\
S &= [(-\frac{1}{\sqrt{6}}R - \frac{1}{\sqrt{6}}G + \frac{2}{\sqrt{6}}B)^2
\end{aligned}$$

$$+(\frac{1}{\sqrt{6}}R - \frac{2}{\sqrt{6}}G)^2]^{\frac{1}{2}} \tag{4.1}$$

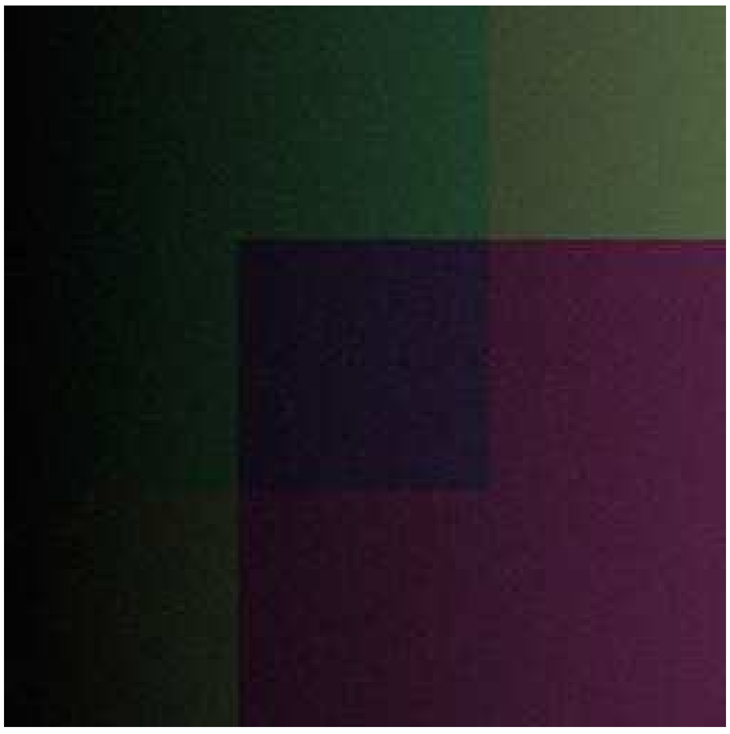
where H is related to the physical quantity 'hue' (a measure of the spectral position of the colour), I is the mean intensity, and S is the 'saturation' (a measure of the purity of the colour). The inverse is :

$$\begin{aligned} R &= \frac{4}{3}I - \frac{2\sqrt{6}}{9}S\cos H + \frac{\sqrt{6}}{3}S\sin H \\ G &= \frac{2}{3}I + \frac{\sqrt{6}}{9}S\cos H - \frac{\sqrt{6}}{3}S\sin H \\ B &= I + \frac{\sqrt{6}}{3}S\cos H \end{aligned} \tag{4.2}$$

[Pratt, 1991]. Several variations on this transformation are possible, depending on whether I is defined as the mean intensity (as above) or the RMS intensity, and what colour is used for the origin of the measurement of H (the above uses the shade of blue defined by $R = G = 0$).

As far as the author is aware the RGB↔HIS transform is only, usually, used for colour image enhancement purposes [see Ruiz, et al., 1977, for an example involving Viking orbiter imagery], no one appears to have considered using it for improving the segmentation of colour images in the manner proposed here. Possibly, this is because of a fear of losing information (although the intensity data can be made use of at a later stage), or because most multi-spectral image segmentation work takes place in the field of remote sensing. Here there are usually more than three spectral bands involved, and equations 4.1 & 4.2 do not easily generalise into an arbitrary number of dimensions. However, given that we are not concerned with the transformed values having physical meaning (other than the intensity), a suitable transform is :

$$\begin{aligned} I_1 &= \sum_{m=1}^{N} i_m \\ I_n &= (n-1)i_n - \sum_{m=1}^{n-1} i_m \qquad [2 \leq n \leq N] \end{aligned} \tag{4.3}$$



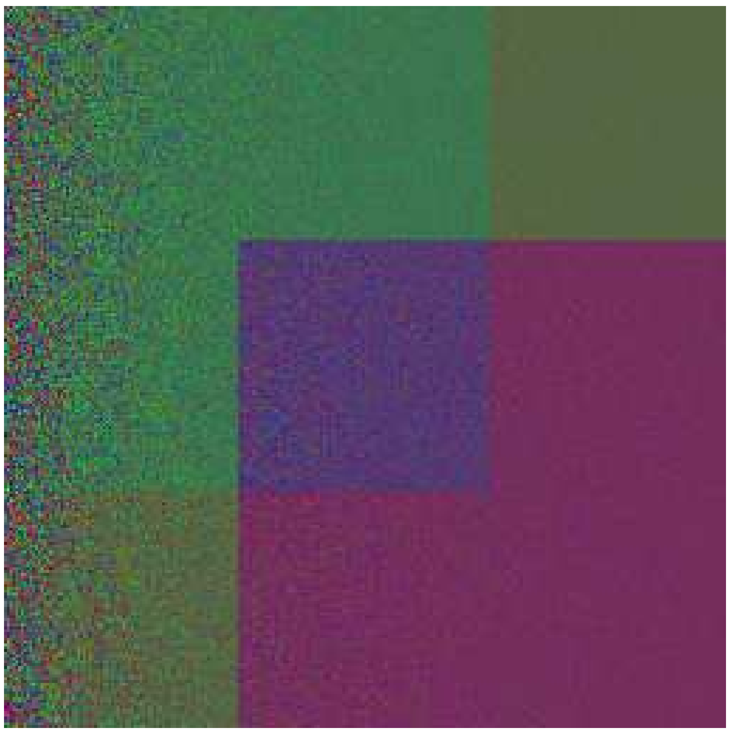


Figure 4.2: Original image [top], result of removing lighting effects by band ratioing [bottom].

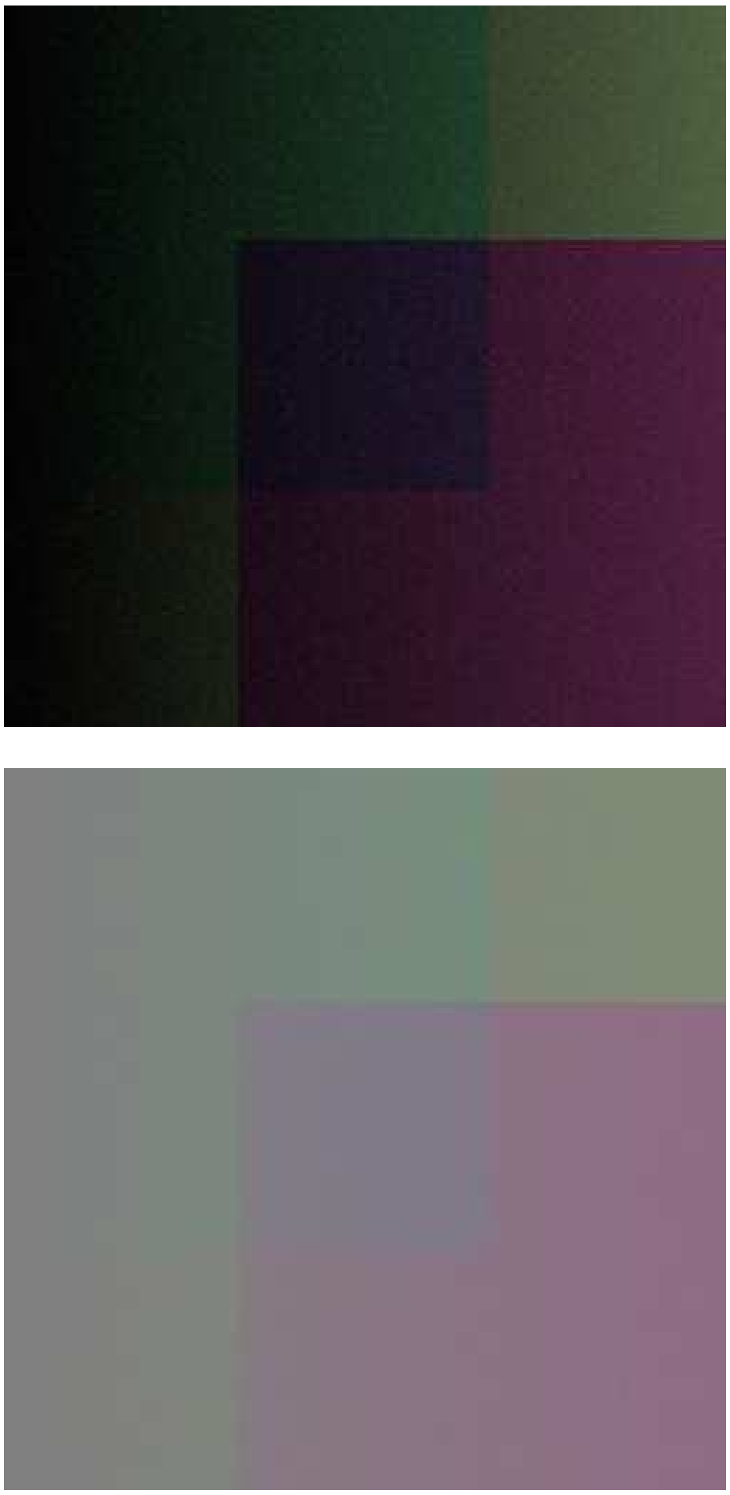


Figure 4.3: Original image [top], result of removing lighting effects by transformation and removal of intensity component [bottom].

which has the inverse:

$$\begin{aligned}
i_N &= \frac{1}{N}(I_1 + I_N) \\
i_n &= \frac{1}{n}(I_1 + I_N - \sum_{m=n+1}^{N} i_m) \qquad [N-1 \geq n \geq 2] \\
i_1 &= I_1 - \sum_{m=n+1}^{N} i_m
\end{aligned} \tag{4.4}$$

where the i_n are the pixel values in the spectral bands, the I_n are the pixel values in the transformed bands, N is the number of bands, and the inverse transform **has** to be done from $n = N$ to $n = 1$.

For three spectral bands the forward transform becomes:

$$\begin{aligned}
I_1 &= i_1 + i_2 + i_3 \\
I_2 &= i_2 - i_1 \\
I_3 &= 2i_3 - i_2 - i_1
\end{aligned} \tag{4.5}$$

and the inverse is :

$$\begin{aligned}
i_3 &= \frac{1}{3}(I_1 + I_3) \\
i_2 &= \frac{1}{2}(I_1 + I_2 - i_3) \\
i_1 &= I_1 - i_2 - i_3
\end{aligned} \tag{4.6}$$

It can be seen that, since the i_n are taken to be independent, the I_n are also independent. This means that the intensity information is uncoupled from the colour information in the transformed space. Thus, the intensity can be modified without the colour being affected. This is true for any number of spectral bands.

In the program (see appendix A, subroutine `NOLIGHT`) the forward transform, removal of intensity variations, and inverse transform are combined into a single transformation matrix (which is normalised so as to keep the spectral values within the allowed range). This matrix is not defined beforehand, but calculated at run-time, based on the number of spectral bands present in the data.

Figure 4.4 shows the result of applying the lighting removal process to a real image (a part of the BBC testcard). When looking at it, it is interesting to note how much the human eye relies on intensity variations for information about the three dimensional nature of objects.

Of course the above assumption means that all the intensity variations which are intrinsic to objects will also be removed, along with those due to lighting effects. However, rather than having one object broken up over several classes (as would happen if the lighting effects were not removed), this will result in several, different, objects being put in the same class. Such under-classification can be corrected for by adding a second stage segmentation process (which takes account of information other than just the colour), whereas spurious over-classification is much more difficult to cure.

The one remaining case in which over-classification may still occur is where a region, which a human would consider as being part of one object, actually contains colour variations. This could be due to either to its being composed of many different coloured sub-objects, or to its having a 'colour texture'. Most man-made objects are of the first type, and a grainy piece of wood is an example of the second type. The correction of this type of problem has to be left to the second (more 'intelligent') stage of any image understanding process.

There are two other, minor, drawbacks to this technique. Firstly, it assumes that, if there is more than one light source, they all have the same spectrum – i.e. spurious results would be obtained if the technique was applied to a scene illuminated from one side by predominantly red light, and from the

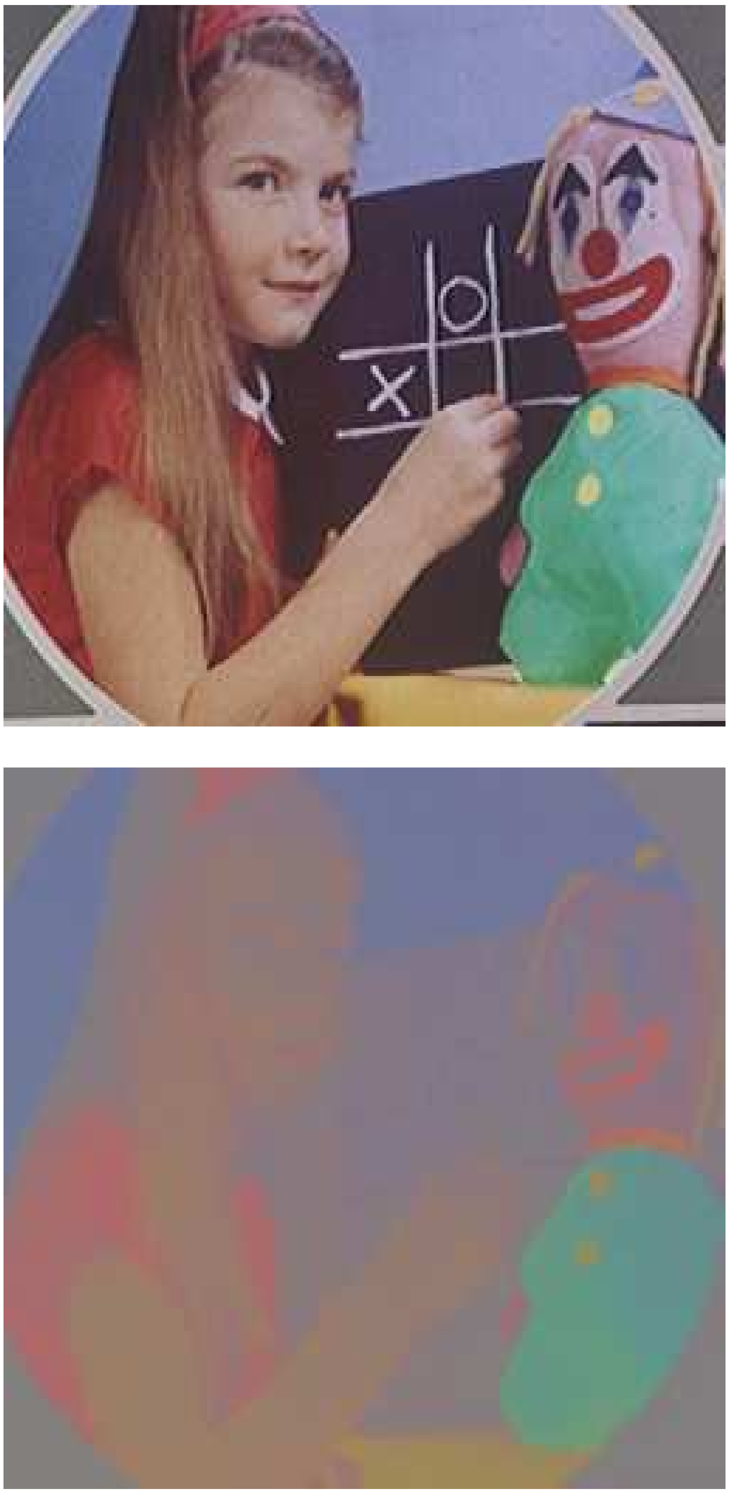


Figure 4.4: BBC testcard image [top], and the result of removing the lighting effects [bottom].

other by predominantly blue light. Secondly, it assumes that the light source is, near enough, white in colour – i.e. a scene illuminated by predominantly red light would still have a reddish cast to it after processing, unless further steps were taken to remove any bias in the colour of the image. Neither of these problems should seriously affect images of natural scenes however, unless they are subject to unusual manmade lighting, or are taken near dawn or dusk.

4.5 The use of cooccurrence matrices, and edge detection :

The cooccurrence matrix, of a single band image, is defined as:

$$S_{\mathbf{\Delta}}(i,j) = \sum_{\mathbf{x}} \delta(I(\mathbf{x}), i)\delta(I(\mathbf{x} + \mathbf{\Delta}), j) \tag{4.7}$$

where the δ are Kronecker delta functions, $\mathbf{x}$ is some position, (x, y), in the image, $\mathbf{\Delta}$ is a displacement vector, and $I(\mathbf{x})$ is the intensity of the image at position $\mathbf{x}$. Also, the summation has to be performed in such a way that both $\mathbf{x}$ and $\mathbf{x} + \mathbf{\Delta}$ lie within the image. Thus, the cooccurrence matrix element $S_{\mathbf{\Delta}}(i,j)$ contains the number of pairs of pixels (separated by the vector $\mathbf{\Delta}$) for which the first pixel has intensity i, and the second has intensity j.

Normally the cooccurrence matrix of a small part of an image is used to characterise the texture within that region. Consequently, most previous uses of cooccurrence matrices for image classification have involved segmenting the image on the basis of texture, either through the calculation of the Haralick texture measures [Haralick, et al., 1973], or by a direct comparison of the matrices for different parts of an image [Carbon & Ebel, 1988].

This work, however, involves an extension of Haddon & Boyce's use of whole image cooccurrence matrices for segmentation [Haddon & Boyce, 1990]. The idea behind this technique is that (provided $|\mathbf{\Delta}|$ is much less than the region size) homogeneous regions within an image will give rise to clusters of data points along the leading diagonal of the matrix, and that discontinuities in the image (edges) will contribute to clusters lying off of the leading diagonal.

Figure 4.5 shows an idealised whole image cooccurrence matrix, for an idealised image. The image contains reasonably homogeneous regions which fall into three different classes, these give rise to the three on-diagonal clusters, A, B, & C. The number of data points in each of the clusters is determined by the number of pixels in each of the classes. The off-diagonal widths of the clusters are determined by the overall noise level in the image, and their along-diagonal widths by the the sum of the noise level and the variation due to texture within the regions of that class. Hence, the on-diagonal clusters are elliptical in outline, with their major axes aligned along the diagonal. It can be seen, from figure 4.5, that the regions in classes A & C are textured, but that those in class B are not. The off-diagonal cluster AB is due to regions of classes A and B having at least one common boundary at which motion in the direction of the displacement vector takes us from a region of class A to a region of class B. The off-diagonal cluster at CA is due to regions of classes A and C having at least one common boundary at which motion in the direction of the displacement vector takes us from a region of class C to a region of class A. The off-diagonal clusters BC and CB show that regions of classes B and C have common boundaries which involve movement from B to C, in some cases, and from C to B, in others, when travelling in the direction of the displacement vector. Finally, the size, and position, of the off-diagonal clusters indicates that all of the boundaries are in the form of 'step' edges.

Figure 4.6 shows the whole image cooccurrence matrix for a synthetic image. None of the regions in this image are textured, so all of the on-diagonal clusters are roughly circular. Apart from that, all of the features are similar

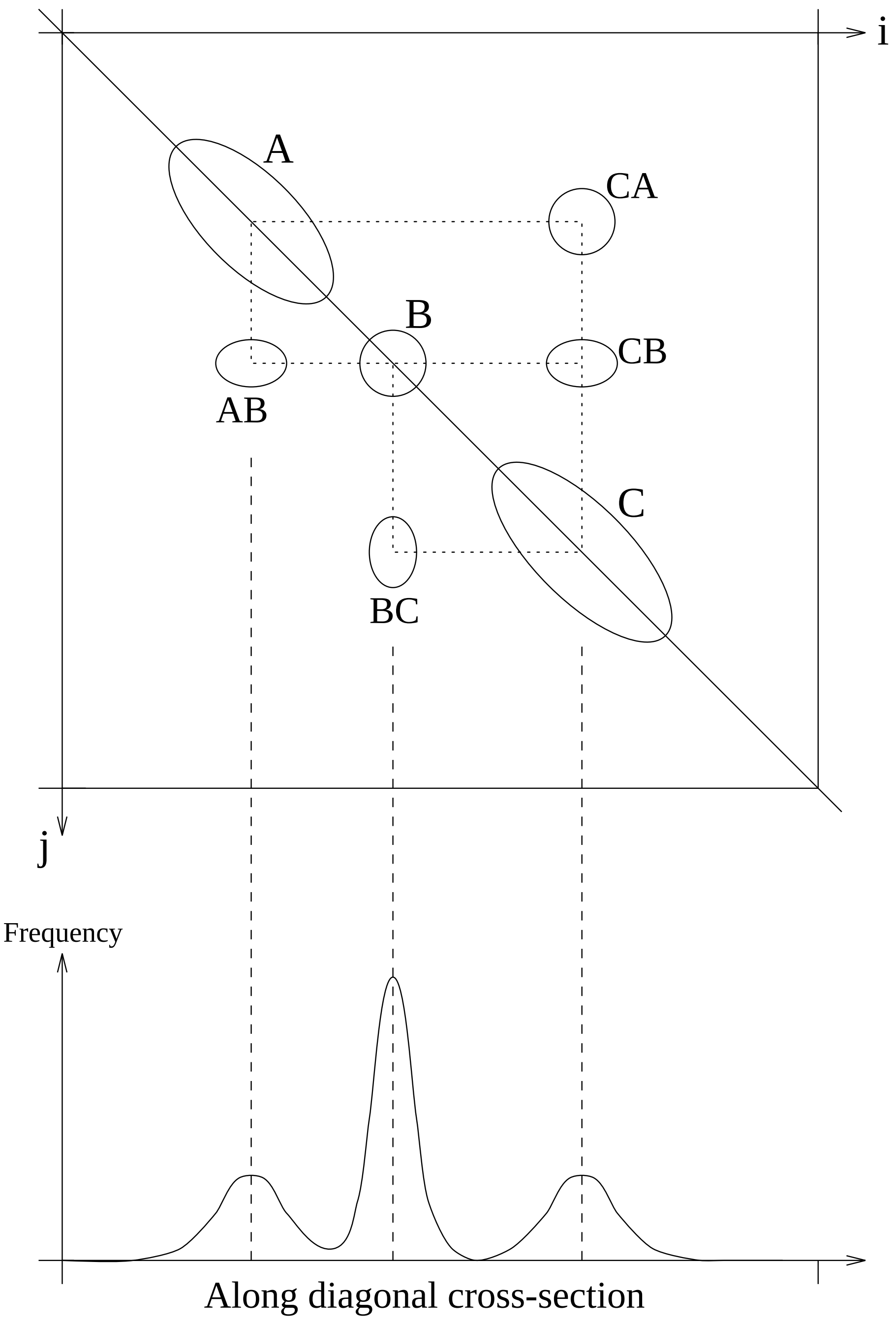


Figure 4.5: The elements of an ideal cooccurrence matrix.

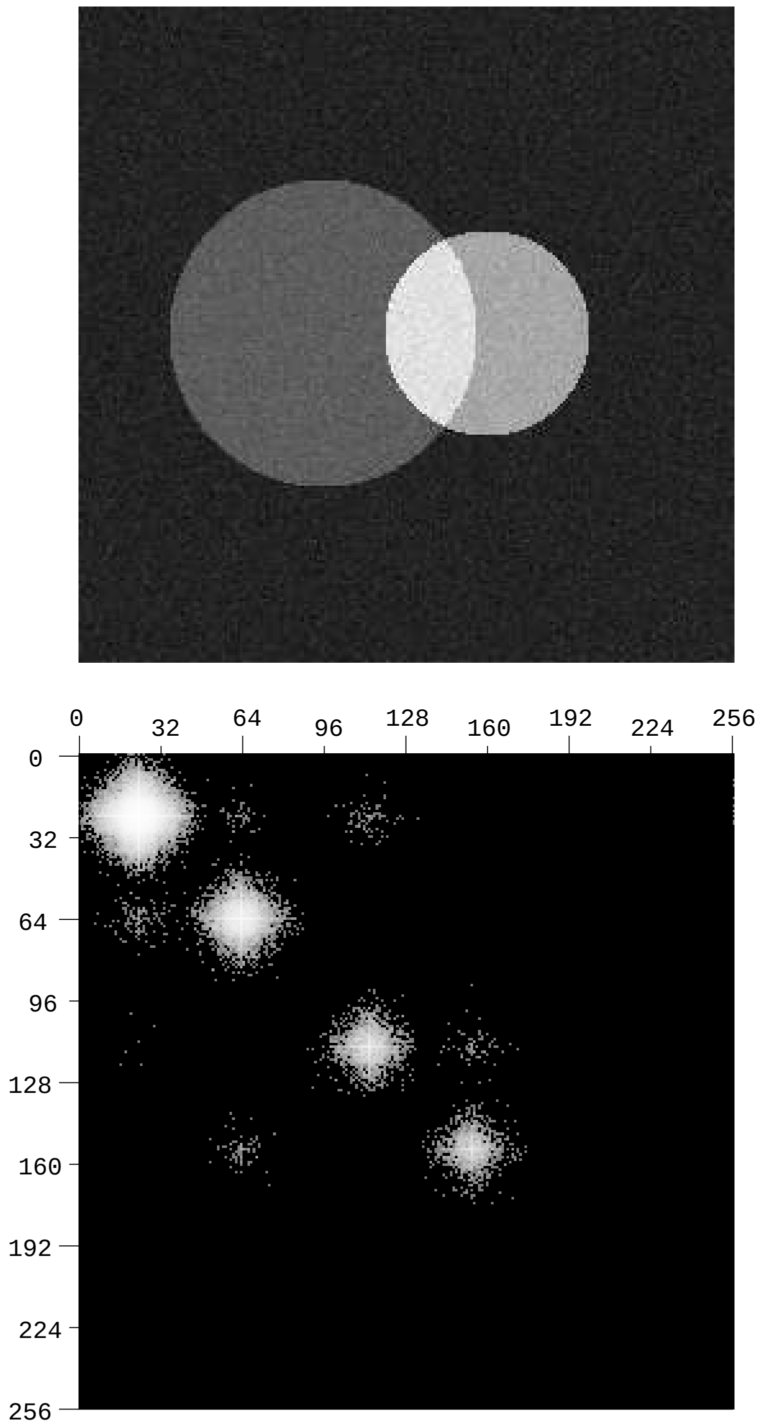


Figure 4.6: Synthetic image [top], and its cooccurrence matrix, $\Delta = (1, 0)$, (logarithmic plot) [bottom].

to those seen in figure 4.5.

Haddon & Boyce's work also involved using cooccurrence matrices produced with $\mathbf{\Delta}$ pointing in different directions to provide information for an iterative improvement of the segmentation ('relaxation labelling'). This makes use of the fact that the shapes of the on-diagonal clusters give an indication of the probability of a pixel, of a given intensity, being in a particular class, and the off-diagonal clusters give an indication of the probability of a pixel being on a boundary in the direction of the displacement vector. These probabilities are then used as the starting values in the calculation of an 'entropy of local information'. The relaxation algorithm seeks to minimise this entropy by iteratively modifying the classifications given to the pixels in the segmentation. At convergence, the final segmentation should be the most locally homogeneous one possible, which is still consistent with the original segmentation. That is, single pixels, which differ in class from their neighbours (which are all in the same class), will be given the same class as their neighbours (assuming that they are not, actually, very much different from them), but the overall relationships between the dominant regions should remain unchanged.

The cooccurrence matrices of a multi-band image may be defined to be :

$$S^{\mu\nu}_{\mathbf{\Delta}}(i,j) = \sum_{\mathbf{x}} \delta(I_\mu(\mathbf{x}), i)\delta(I_\nu(\mathbf{x}+\mathbf{\Delta}), j) \tag{4.8}$$

where everything is as before except that $I_\mu(\mathbf{x})$ is the intensity of the image at $\mathbf{x}$ in band μ. However, it is debateable whether or not any useful information is gained by considering $\mu \neq \nu$. Looking at different pixels in the same band, and the same pixel in different bands will both, obviously, give useful information (the first about spatial homogeneity, the other about spectral composition). But, it is difficult to see how looking at different pixels in different bands will be of any use. So, in this work, only cooccurrence matrices for which $\mu = \nu$

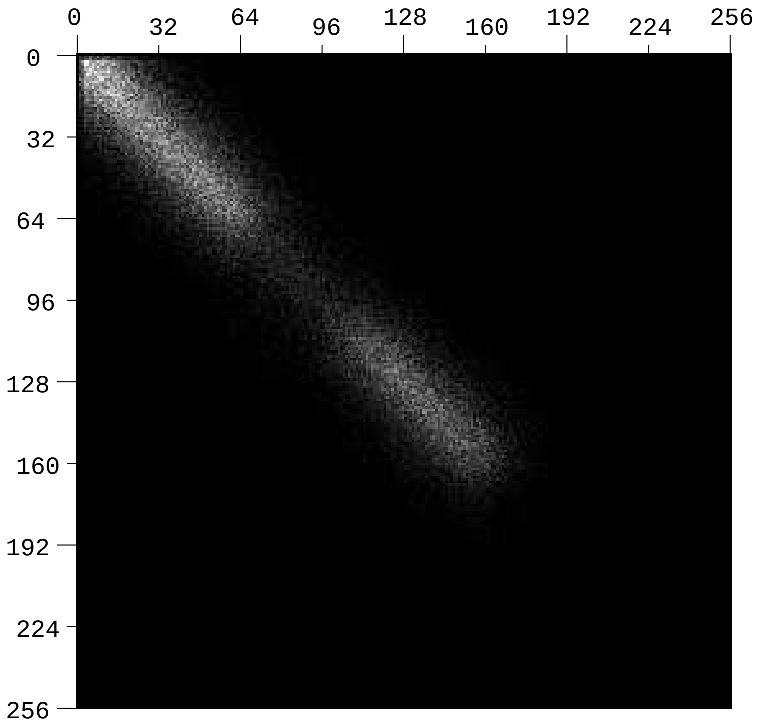
will be considered. This has the added advantage of reducing the problem from being proportional to N^2 to being proportional to $2N$.

The cluster location part of the problem can be further reduced to N dimensions by using the fact that the region clusters must lie along the leading diagonals of the matrices, and that their centres will be coincident in the cooccurrence matrices and in the multi-dimensional histogram space (i.e. data giving rise to a cluster centred at (a, a, b, b, c, c) in a six dimensional cooccurrence matrix will also give rise to a cluster centred at (a, b, c) in the corresponding, three dimensional, histogram space). Thus, the N leading diagonals may be used as the one dimensional sub-spaces in the first stage of the cluster analysis (the peak location), and the second stage (the tests of local maximality) can be performed in the N–dimensional histogram space. The actual pixel classification will still have to be performed in the full $2N$–dimensional cooccurrence space however.

Apart from extending the use of cooccurrence matrices from single band images to multi-band images, there are three other important differences between this work and that of Haddon & Boyce.

Firstly, the removal of lighting effects (as discussed above) ensures that the regions are more homogeneous than they would be in a single band image, and so the clusters in cooccurrence space will be more compact (see figures 4.7 to 4.9 – n.b. the dark cross in the centre off figure 4.9 is a binning artefact).

Secondly, rather than doing relaxation labelling to improve the segmentation, the information from different directions of $\mathbf{\Delta}$ is used in such a way that the most region-like direction is used for segmentation purposes, and the most edge-like direction is used for edge detection purposes. For example, in figure 4.10 displacement vectors 1 & 4 both correspond to the most region-like directions, because both ends of each vector lie within the same region and



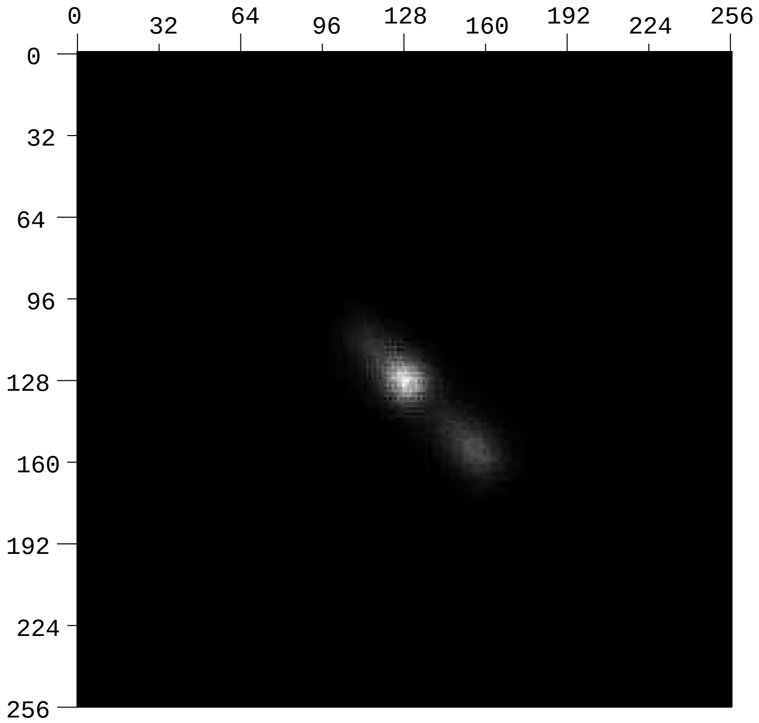
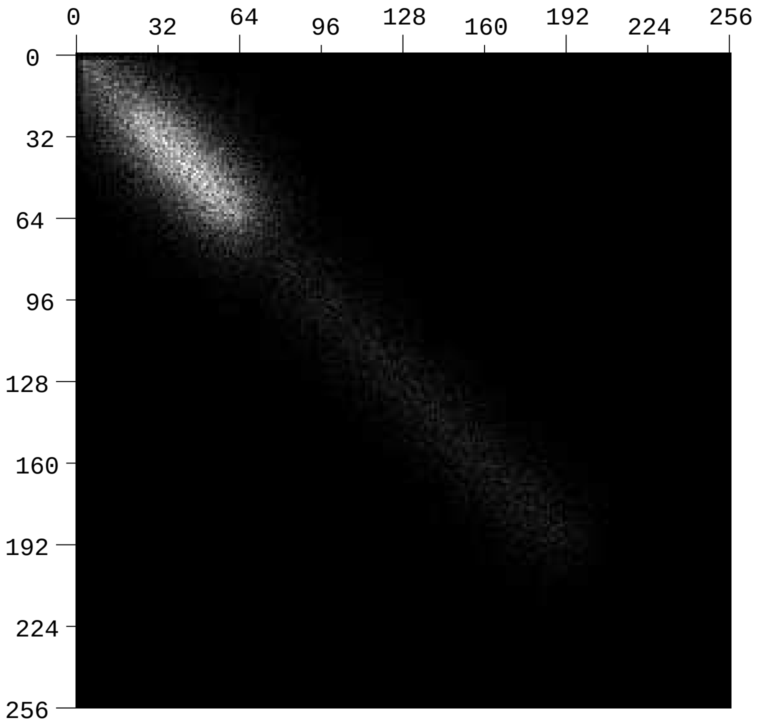


Figure 4.7: Grey level cooccurrence matrices ($\mathbf{\Delta} = (1, 0)$) for band 1 of the image in figure 4.2: without [top] and with [bottom] removal of lighting effects.



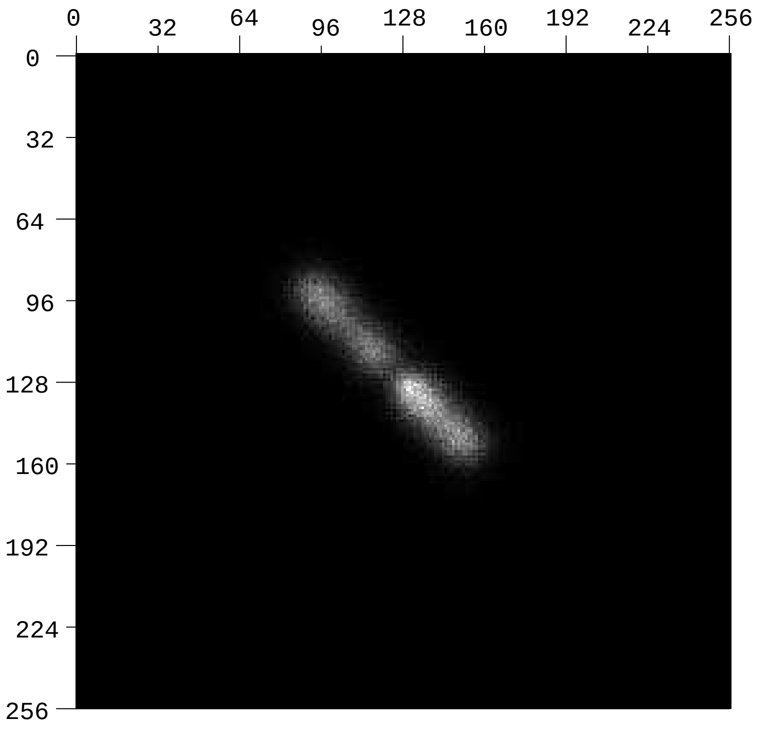
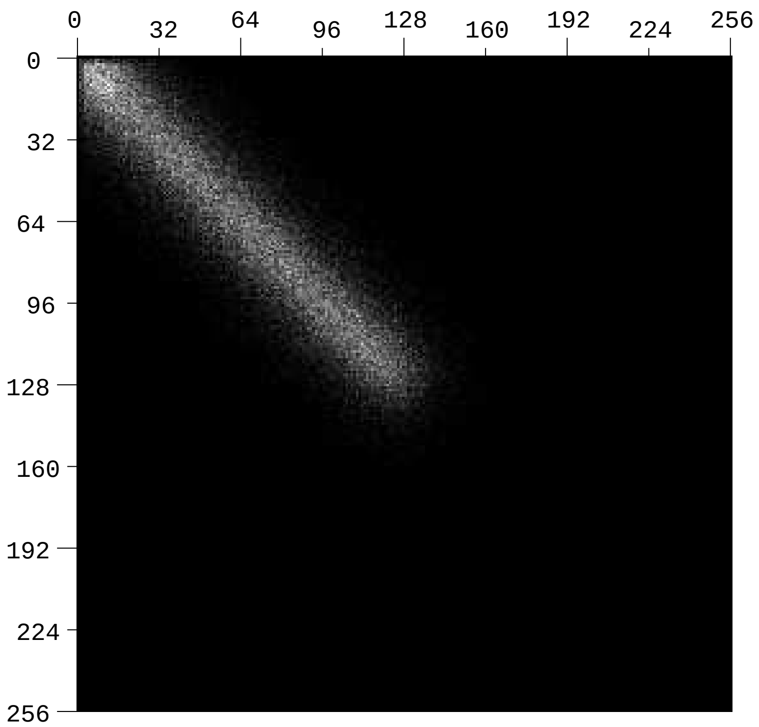


Figure 4.8: Grey level cooccurrence matrices ($\mathbf{\Delta} = (1, 0)$) for band 2 of the image in figure 4.2: without [top] and with [bottom] removal of lighting effects.



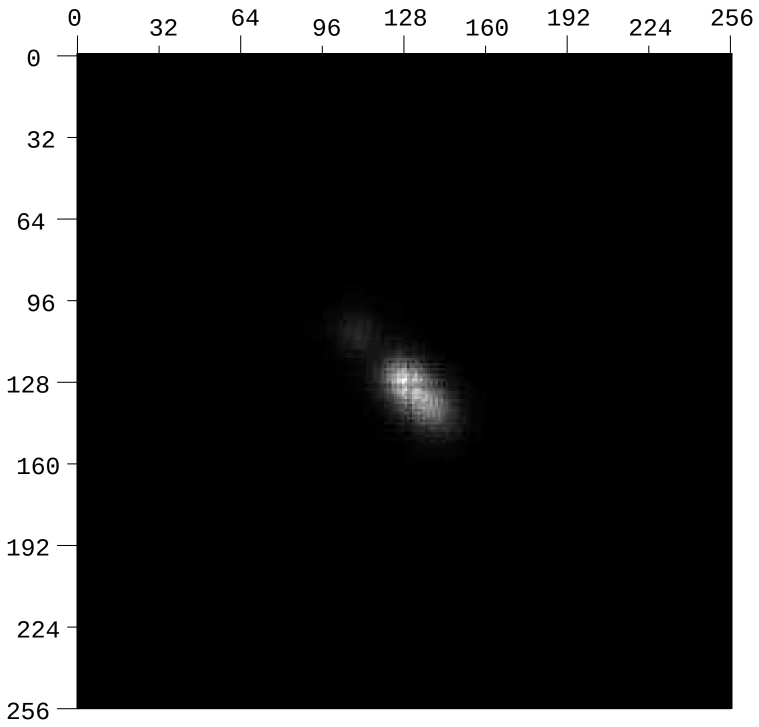


Figure 4.9: Grey level cooccurrence matrices ($\mathbf{\Delta} = (1, 0)$) for band 3 of the image in figure 4.2: without [top] and with [bottom] removal of lighting effects.

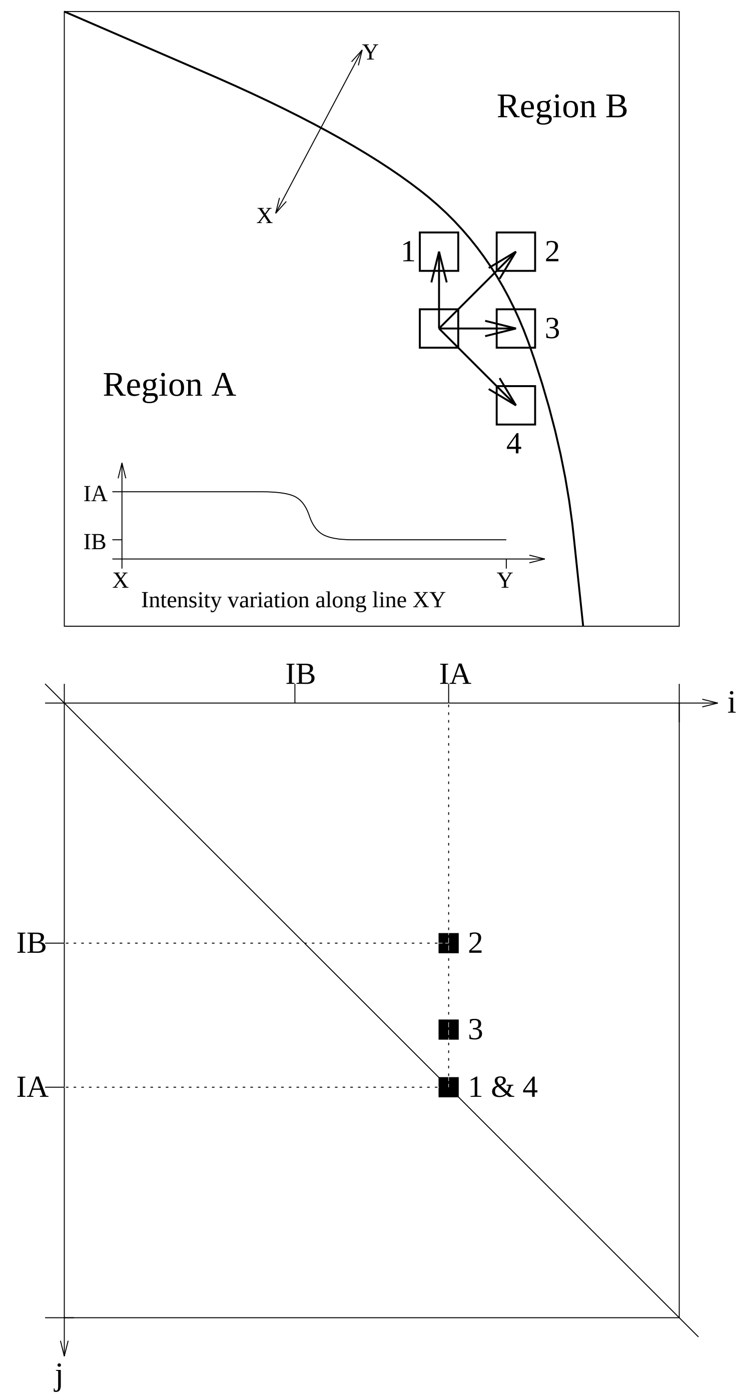


Figure 4.10: Edge-like and region-like pixel pairings in image space [top], and their corresponding contributions to cooccurrence space [bottom].

they, thus, give rise to the matrix entries which lie closest to (in fact, on) the leading diagonal. Displacement vector 2 corresponds to the most edge-like direction, because its two ends lie in different regions and, therefore, it gives rise to the matrix entry which lies furthest from the leading diagonal. Thus, either displacement vector 1 or displacement vector 4 would be used in determining which class the pixel should belong to (both vectors will give the same result), and displacement vector 2 would be used in determining the edge strength at that point in the image. We could also record information about the edge direction, but this has not been considered as part of this work, largely because of memory restrictions on early versions of the program. As in figure 4.10, the four displacement vectors used throughout this work (unless otherwise stated) correspond to 'north', 'north-east', 'east', and 'south-east'.

Thirdly, unless an image contains very sharp edges there will be no well defined off-diagonal clusters. Figure 4.11 shows the sorts of off-diagonal distributions which we get in cooccurrence space, for a variety of different edge profiles, and figure 4.12 shows the whole image cooccurrence matrix for a the BBC testcard image.

So, rather than trying to look for edge clusters which are unlikely to be there, this work uses an edge probability function, in the cooccurrence space, which is determined from the positions of the cluster centres (for an example, with a single band image, see figures 4.13 & 4.14). Those parts of the cooccurrence space which are far from the diagonal have a high probability of being edge, as do those which lie between clusters which are close together. Those parts of the space which lie close to the diagonal have a low probability of being edge, as do those which lie within the clusters. The construction of the function is, at present, entirely empirical, but it does give reasonably good results, provided that the correct number of clusters are found during

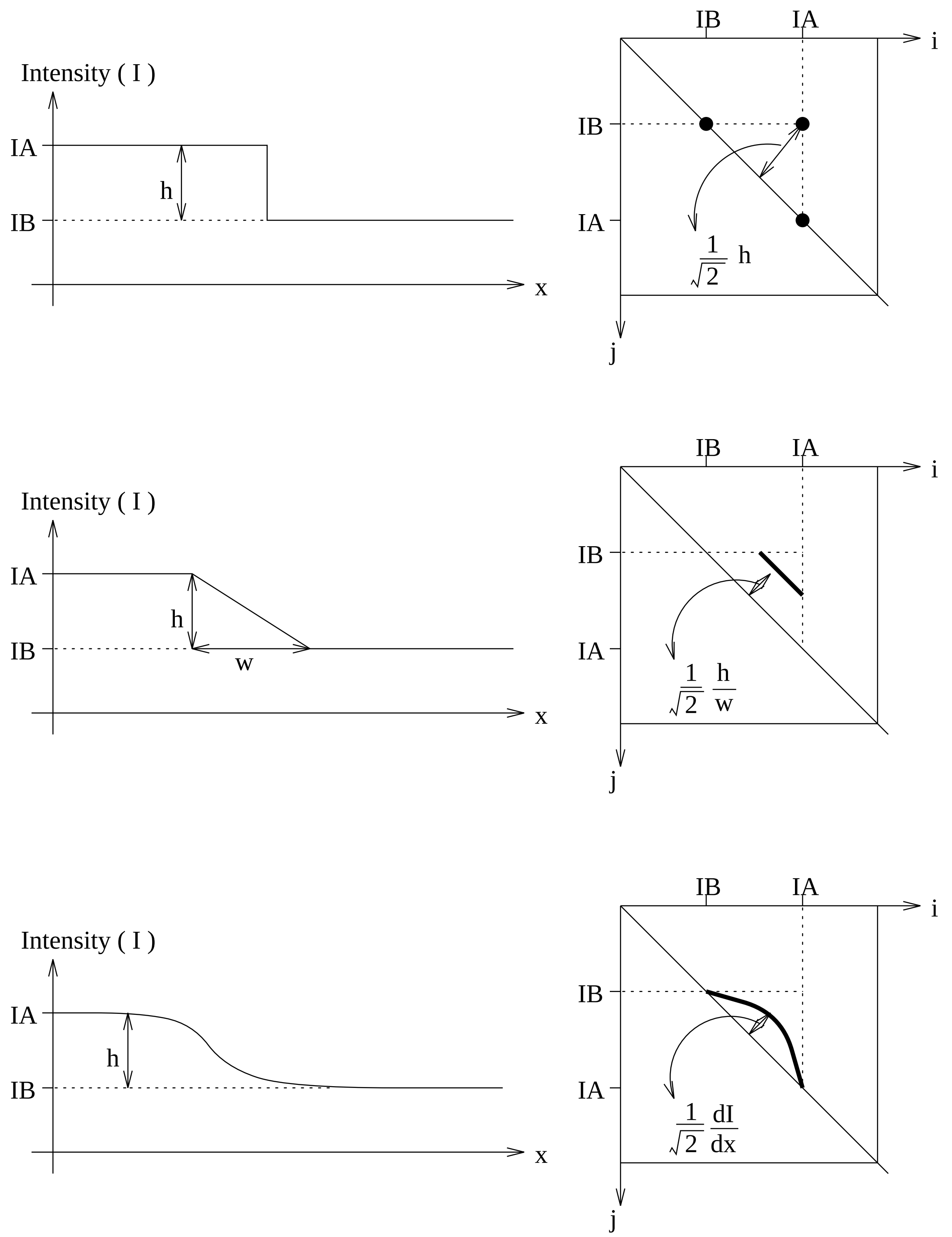
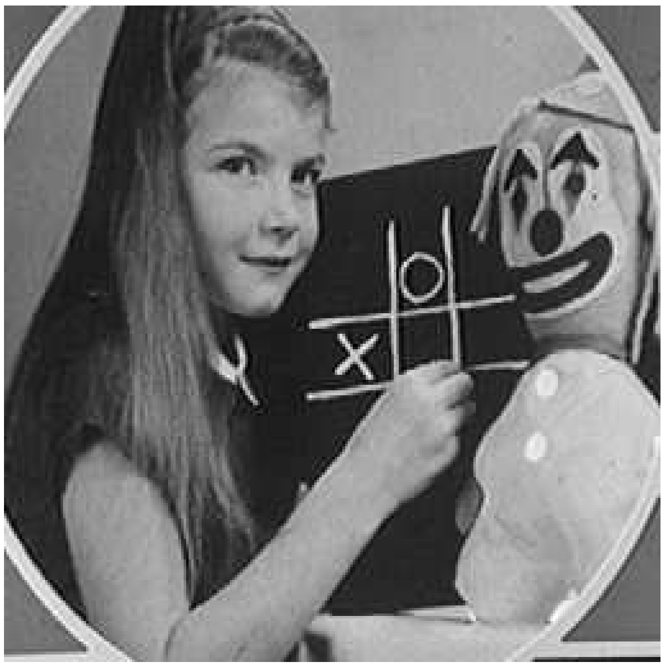

Intensity (I)
IA
IB
h
w
x
IB
IA
i
j
$\frac{1}{\sqrt{2}}$ h
$\frac{1}{\sqrt{2}} \frac{h}{w}$
$\frac{1}{\sqrt{2}} \frac{dI}{dx}$

Figure 4.11: Edge profiles, and their corresponding contributions to a cooccurrence matrix.



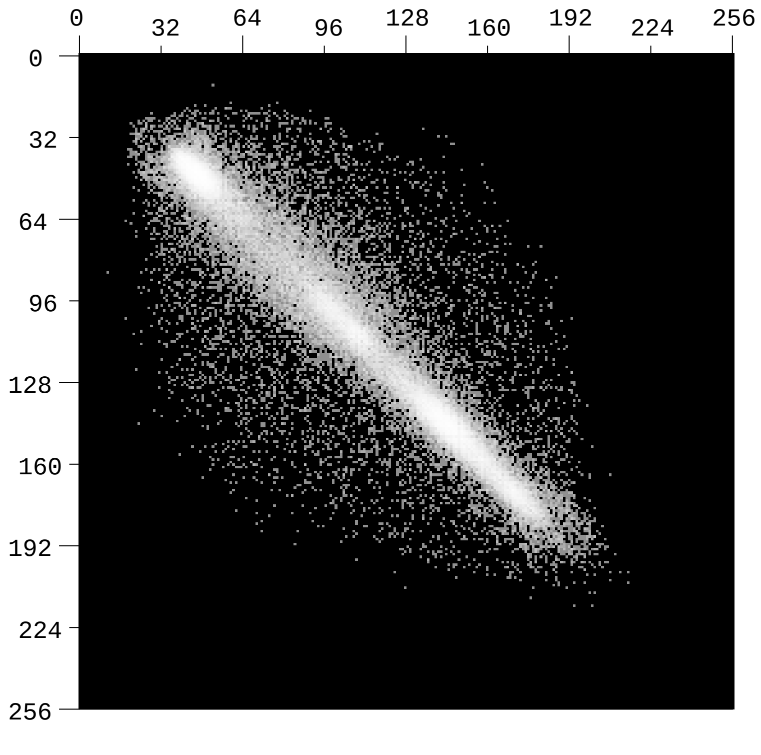


Figure 4.12: BBC testcard image [top], and its cooccurrence matrix, $\Delta = (1, 0)$, (logarithmic plot) [bottom].

segmentation (figure 4.14).

In order to improve the edge detection properties of a cooccurrence matrix based segmentation (and reduce the effects of noise on the region detection) one can use the two halves of an edge detection filter in place of the two pixel values, $I(\mathbf{x})$ & $I(\mathbf{x}+\mathbf{\Delta})$.

The one drawback of using a wide filter for edge detection is that the edges, so detected, get wider. To overcome this one can apply an edge thinning algorithm such as Canny's hysteresis thresholding [Canny, 1986], as shown in figures 4.14 & 4.15.

The program (see appendix A) allows the user a choice of several different filter types. A Gaussian filter, a cubic spline filter, the Petrou filter [Petrou, 1989], and the Spacek filter [Spacek, 1986]. Figures 4.16 to 4.21 show the result of using the different filters, at different sizes, on the image from figure 4.6. The edge maps have all had hysteresis thresholding applied to them (with a low threshold of 0.1 and a high threshold of 0.9) before being overlain on the segmentation.

In all the cases where more than the actual four classes has been found the edge detection has suffered as a result. This is because the practice of introducing a high edge probability between close clusters (see the top left corner of figure 4.13) means that large numbers of non-edge pixels receive high edge probabilities if one actual cluster is split between several false clusters. This results in whole regions of the image appearing to be of high edge probability (rather than just linear features) which causes problems for the edge-following and edge-thinning parts of the HYSTER algorithm. The same effect can occur even if the correct number of clusters is found. Usually this is because the high edge probability incursions into the vicinity of the close clusters should really be in the form of a pair of wedges, rather than a band right across the matrix

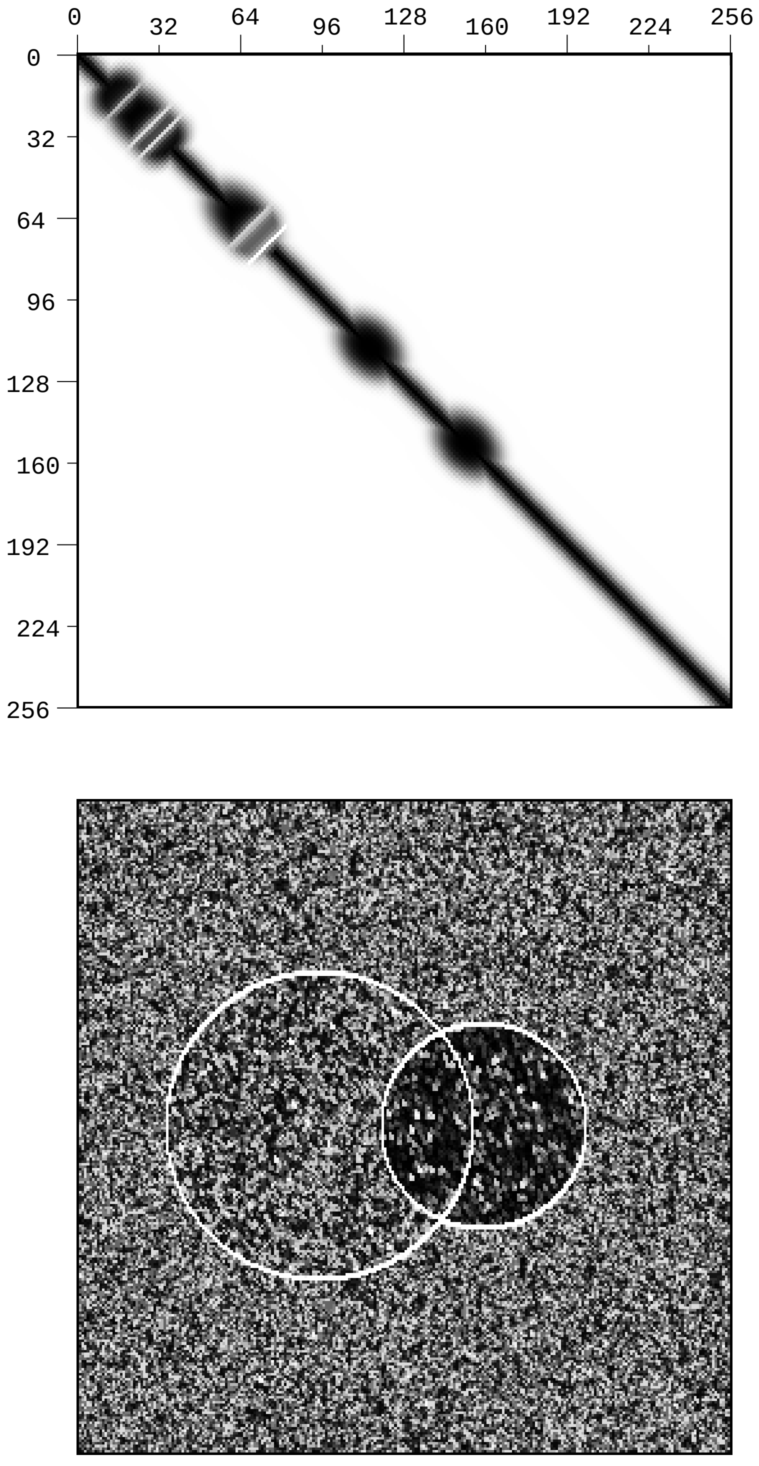


Figure 4.13: The edge probability function, in cooccurrence space, for the image in figure 4.6 (8 clusters found during segmentation) [top], and the edge probability image produced by mapping the image through it [bottom].

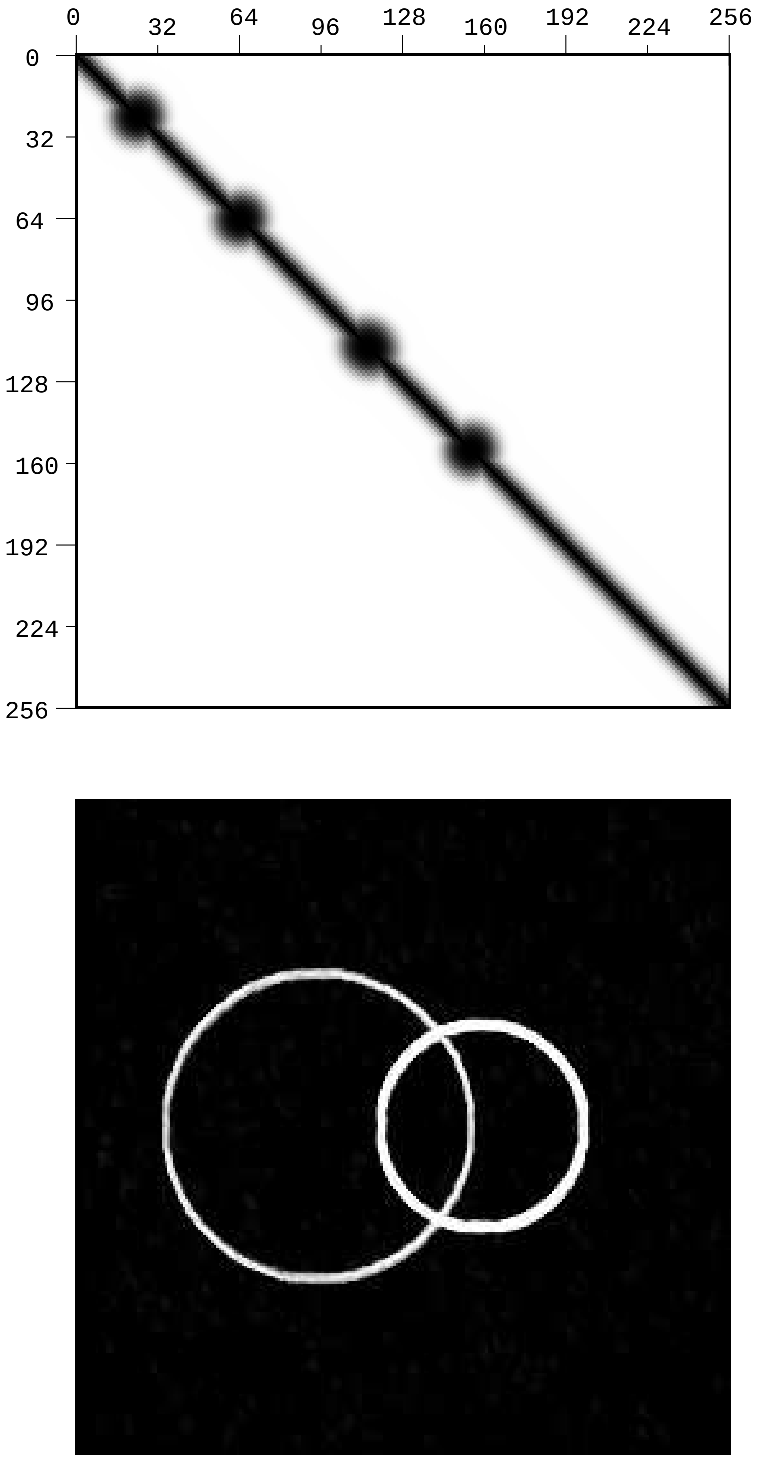


Figure 4.14: The edge probability function, in cooccurrence space, for the image in figure 4.6 using a 7–point Spacek filter (4 clusters found during segmentation) [top], and the edge probability image produced by mapping the image through it [bottom].

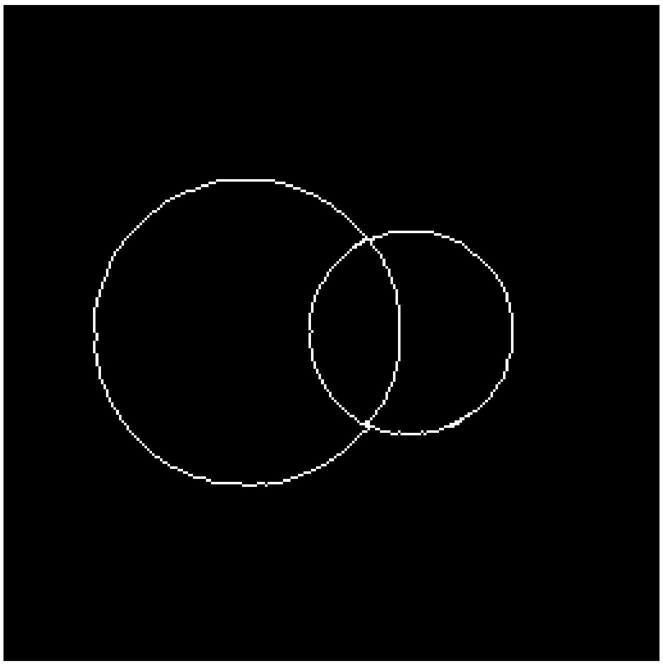
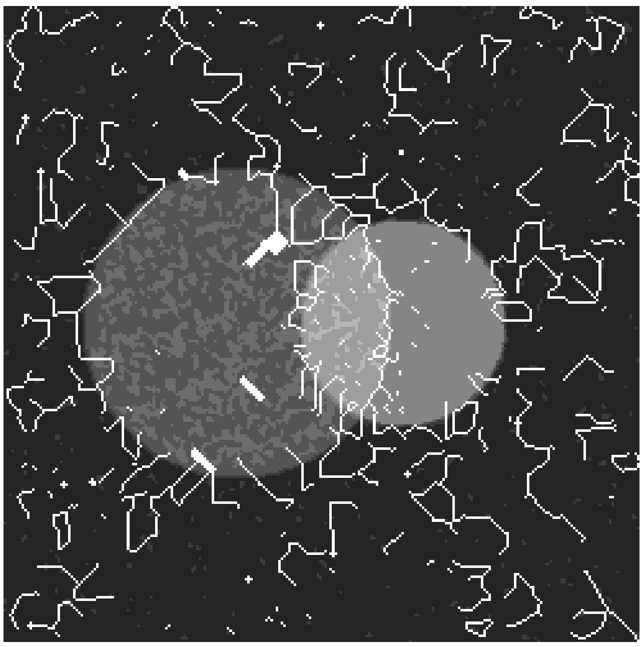


Figure 4.15: The result of applying Canny's HYSTER algorithm to the edge probability image in figure 4.14 (low threshold = 0.1, high threshold = 0.9).

(see figure 4.22). This would more accurately reflect the fact that a pair of pixels corresponding to a point near the leading diagonal are extremely unlikely to be part of an edge, without losing the ability to find the edges between regions which are spectrally similar. However, the wedges are rather difficult to formulate in the multi-dimensional space under consideration. When time permits a more rigorous consideration of the form of the edge probability mapping this is the main point which will need to be dealt with. Even if the correct number of clusters are found, and the close clusters are dealt with sensibly, regions of the image may still be given spuriously high edge probabilities. This is because one component of the edge probability can be considered to be a measure of the inaccuracy of the pixel's classification. Under ideal circumstances this is all very well, and may even be useful. However, if the locations, or widths, of the clusters are not accurately determined, for some reason, then the edge



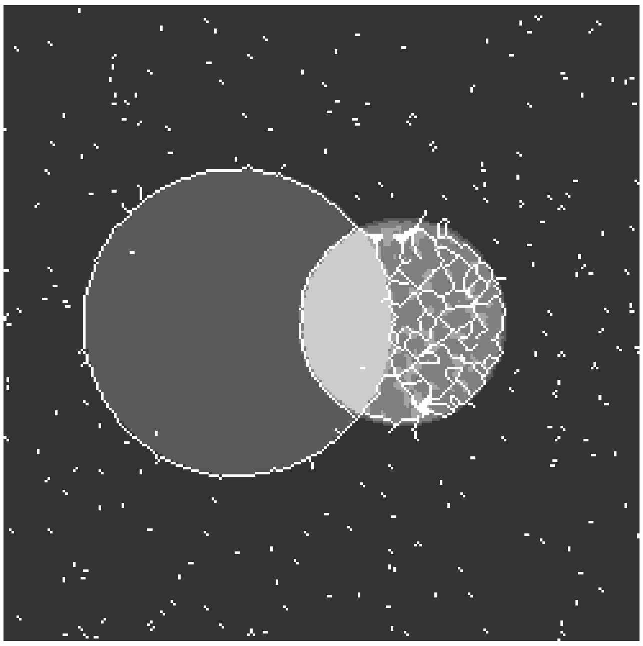
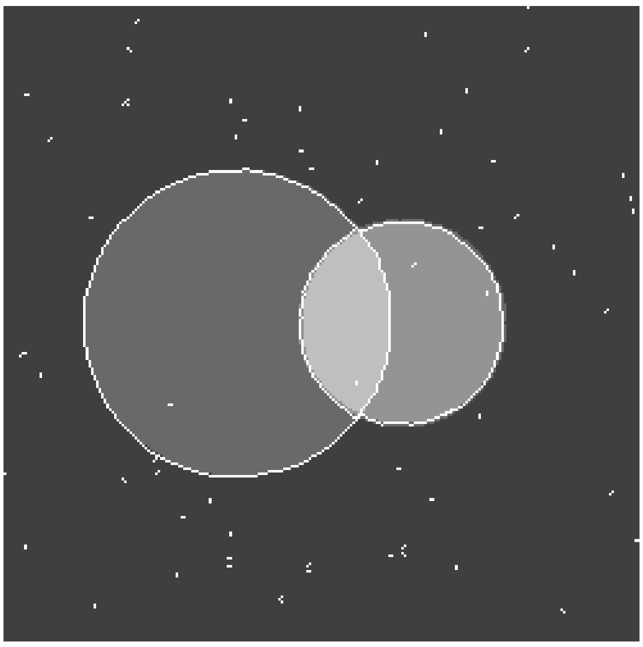


Figure 4.16: Combined region segmentation and edge detection, using a 5–point Gaussian filter (7 classes found during segmentation) [top], and a 5–point cubic spline filter (5 classes found during segmentation) [bottom].



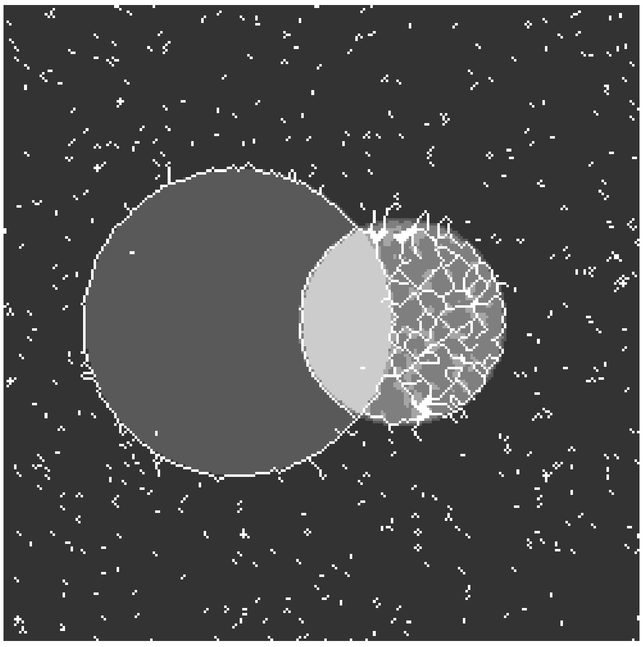
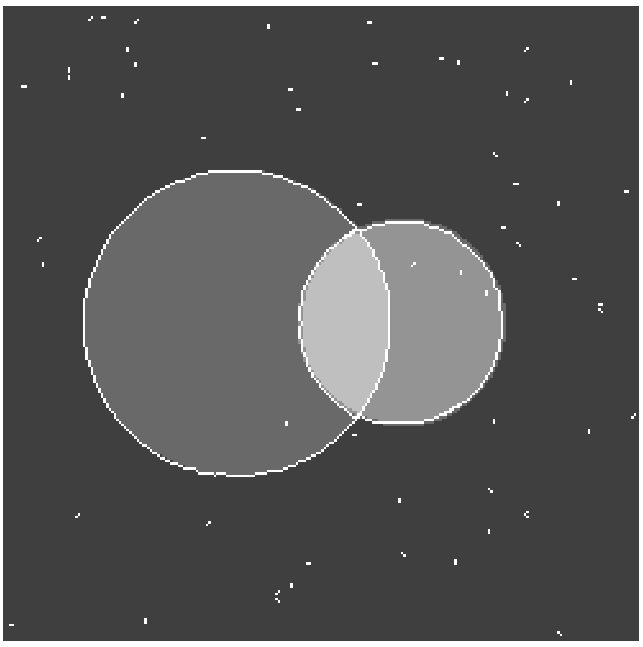


Figure 4.17: Combined region segmentation and edge detection, using a 5–point Petrou filter (4 classes found during segmentation) [top], and a 5–point Spacek filter (5 classes found during segmentation) [bottom].



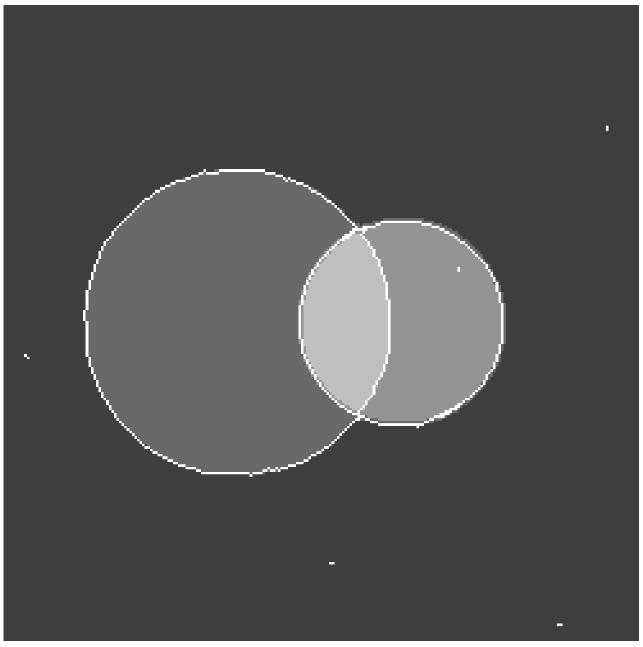
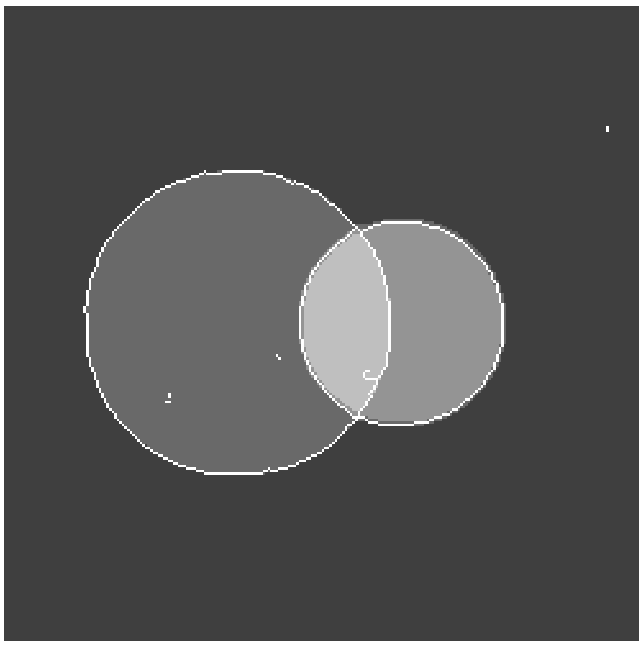


Figure 4.18: Combined region segmentation and edge detection, using a 7–point Gaussian filter (4 classes found during segmentation) [top], and a 7–point cubic spline filter (4 classes found during segmentation) [bottom].



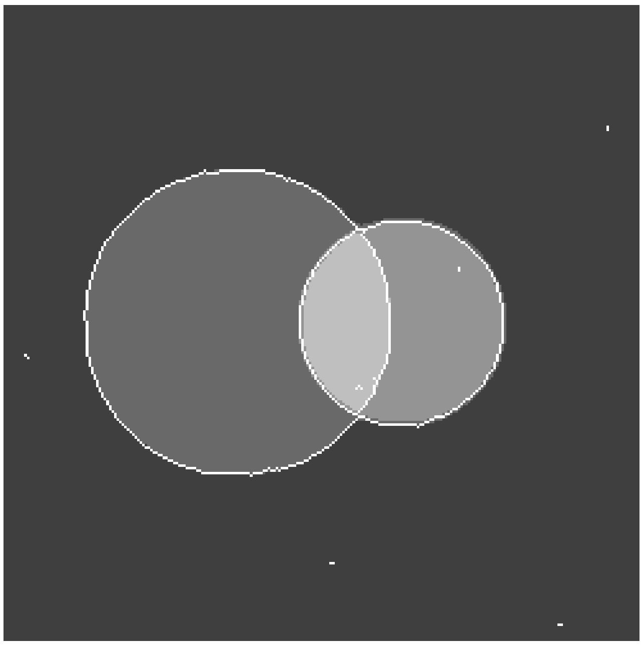
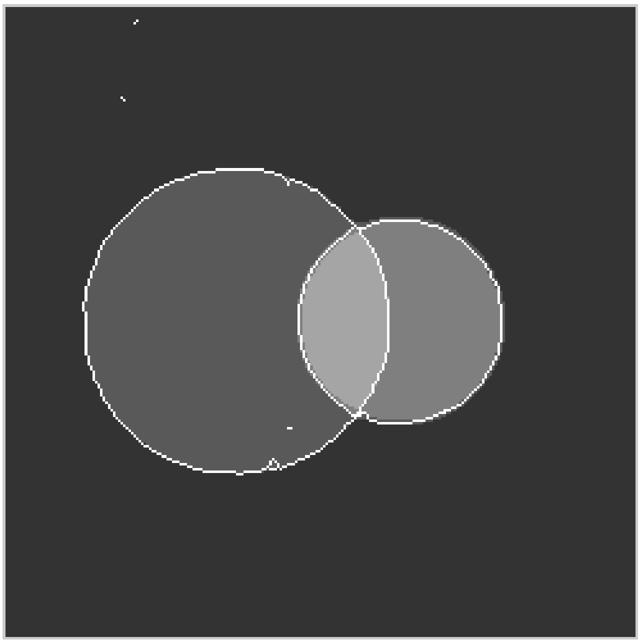


Figure 4.19: Combined region segmentation and edge detection, using a 7–point Petrou filter (4 classes found during segmentation) [top], and a 7–point Spacek filter (4 classes found during segmentation) [bottom].



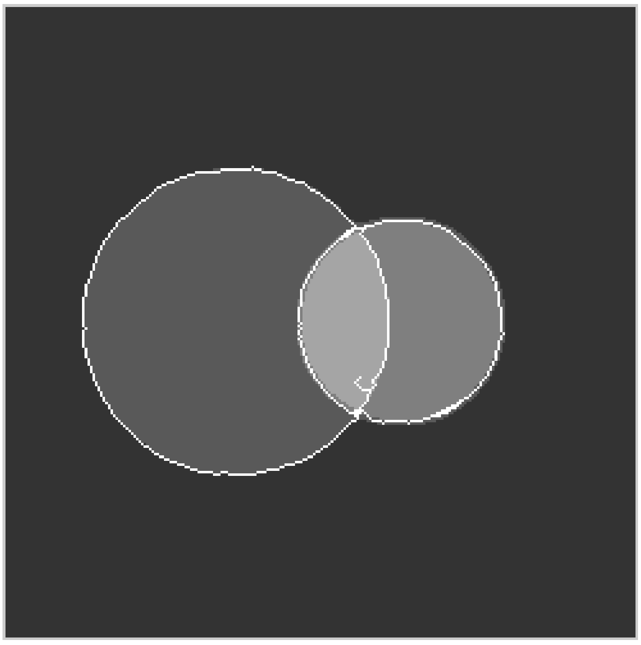
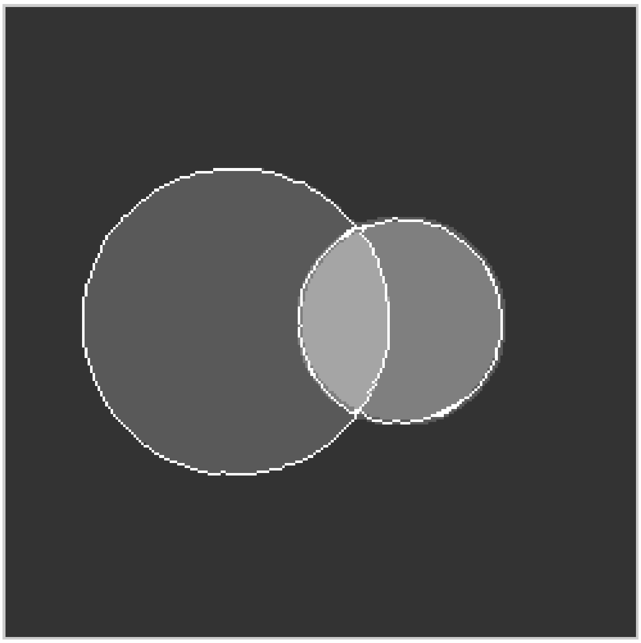


Figure 4.20: Combined region segmentation and edge detection, using a 9–point Gaussian filter (4 classes found during segmentation) [top], and a 9–point cubic spline filter (4 classes found during segmentation) [bottom].



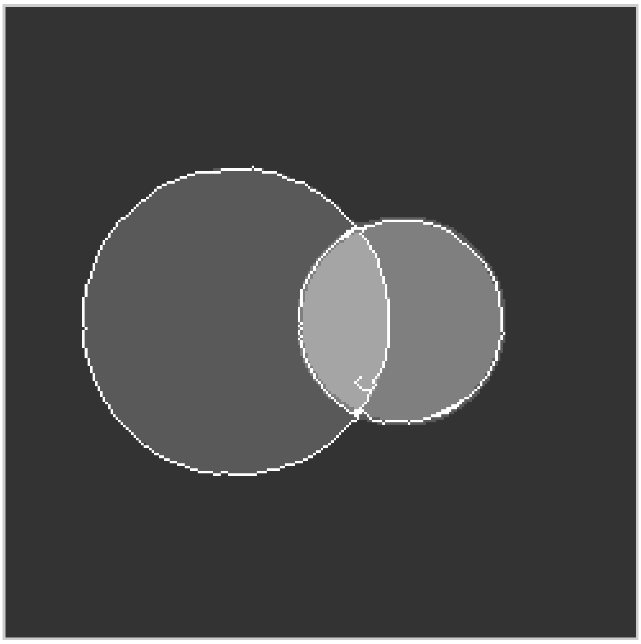
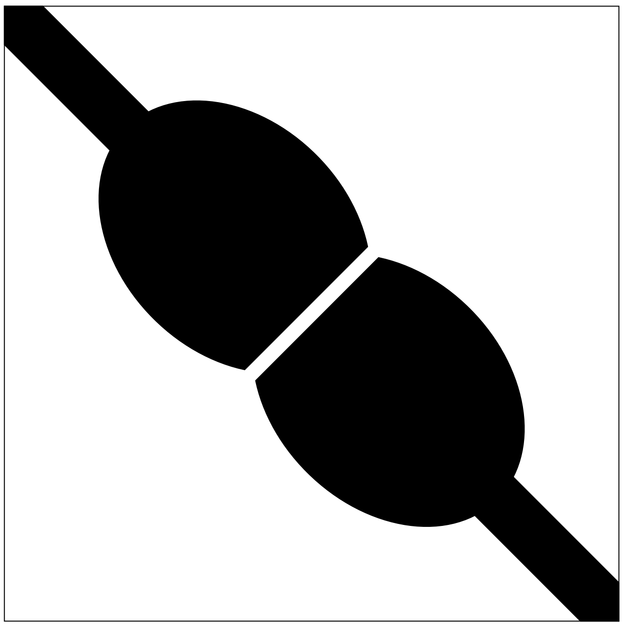


Figure 4.21: Combined region segmentation and edge detection, using a 9–point Petrou filter (4 classes found during segmentation) [top], and a 9–point Spacek filter (4 classes found during segmentation) [bottom].



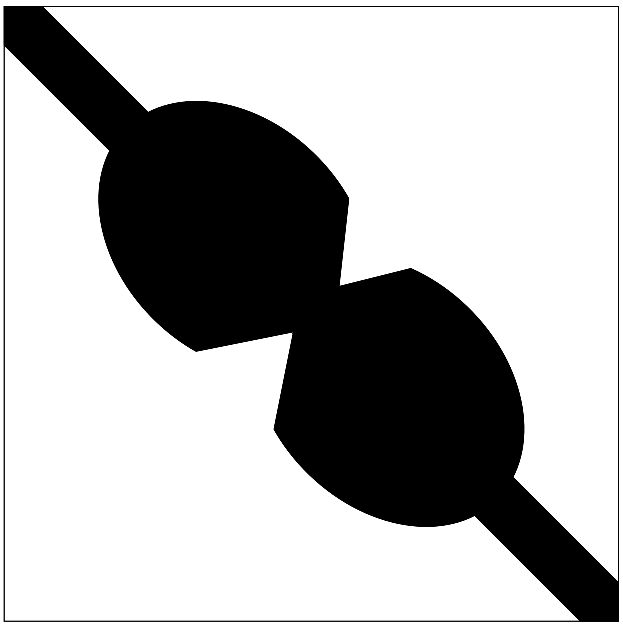


Figure 4.22: Band [top], and wedge [bottom] shaped high edge probability incursions between close clusters.

probability mapping function will be the wrong shape. This means that some of the pixels may receive the wrong edge probability. If the value is too low then edges may become broken, or be missing completely, if the value is too high then we will, again, get whole regions which appear to be edge-like, causing confusion for the HYSTER algorithm.

It would appear that the Petrou filter gives the best results but, for reasons of compatibility with old results, the Spacek filter will be used throughout the rest of this work. Note that the detected edges do not always match the region edges. This is because the blurring action of the filters does not affect the edge detection and segmentation in the same way. The edge detector will place the edge at the point of maximum slope on the edge ramp, whereas the segmentation algorithm will place the edge halfway up the edge ramp. The two do not necessarily coincide, especially as the ramp height increases.

4.6 Results using real images :

Figures 4.23 to 4.26 show the result of running the segmentation program on the full colour version of the BBC testcard data (see figure 4.4), using various different options.

Figure 4.23 is the segmentation which results when the image is not smoothed, and the lighting effects are not removed. 22 classes have been found, and the resulting regions are, mostly, very fragmented. This sort of segmentation would only be useful, for anything other than data compression, after a great deal of post-processing.

Figure 4.24 is the segmentation which results when the image **is** smoothed, with a five point Spacek filter, but the lighting effects are still not removed. This also conatins 22 classes. The regions are far less fragmented now, but it would still be difficult to use this as the basis for any attempt to analyse the image.

Figure 4.25 is the segmentation which results when the image is not smoothed, but the lighting effects **are** removed. 6 classes have been found, and, although the regions show a certain amount of fragmentation, this is a vast improvement on the previous two results.

Finally, figure 4.26 is the segmentation which results when the image is smoothed, with a five point Spacek filter, **and** the lighting effects are removed. This contains 6 classes, as with the previous result, but now the fragmentation is minimal. Arguably there should be a 'pink' class for the clown's face, but this is so pale that it is not surprising that it has not been distinguished from the 'colourless' objects. More information would need to be combined with this segmentation before an interpretation could be attempted – for instance the girl's hair would need to be distinguished from her skin, and united into a single region. This information may be able to be gleaned from the intensity image, which we have discarded, particularly by analysing the textures in it. But, as far as a segmentation on the basis of colour alone goes, this result is probably as good as it is possible to get.

Figure 4.27 shows the edge probability image which results from the last segmentation, and figure 4.28 shows the result of running Canny's HYSTER algorithm on it (with a high threshold of 0.9, and a low threshold of 0.1, as with the previous cases). The edge probability map contains some regions of high edge probability, as discussed in the previous section, and these have caused some problems for HYSTER (in particular on the girl's forehead). Also, several real edges have been removed by HYSTER. It would be possible to get these back by 'tuning' the thresholds, but since the main idea behind the segmentation algorithm is to have no user intervention, this would rather defeat the object. For this reason any further work on edge detection will have

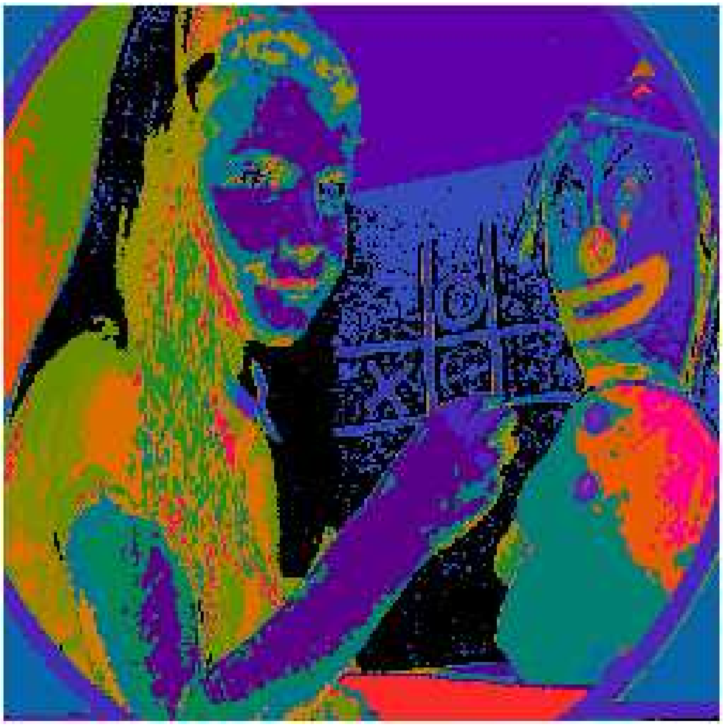


Figure 4.23: Segmentation of the colour BBC testcard image, produced without smoothing, or lighting effect removal.

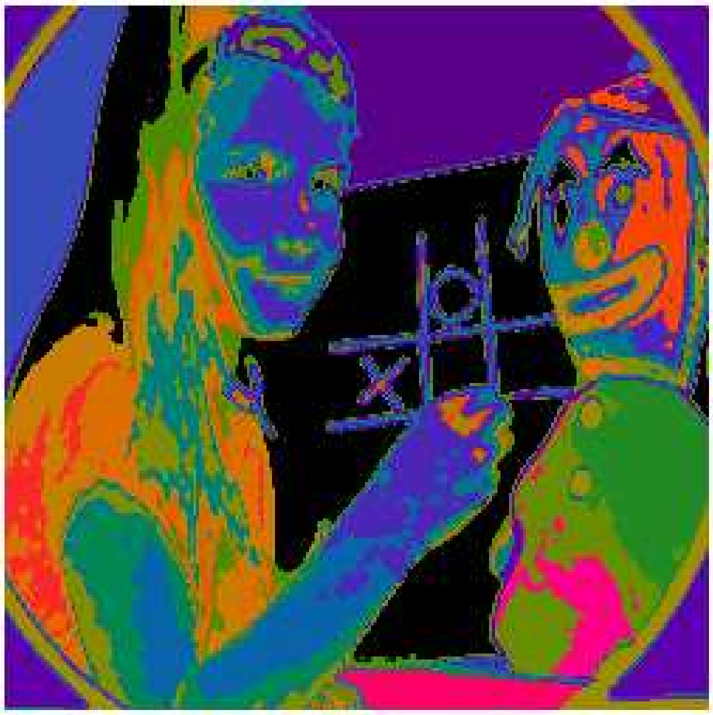


Figure 4.24: Segmentation of the colour BBC testcard image, produced with smoothing, but without lighting effect removal.

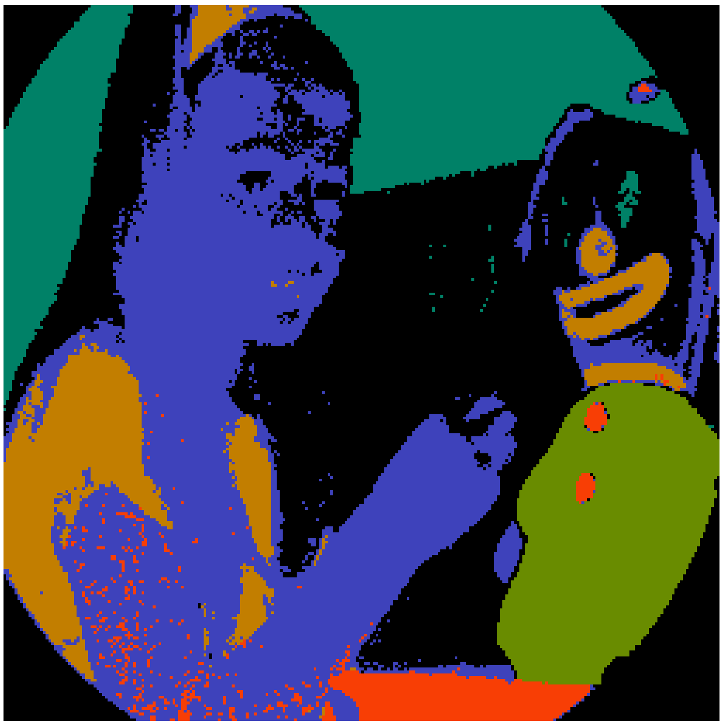


Figure 4.25: Segmentation of the colour BBC testcard image, produced without smoothing, but with lighting effect removal.

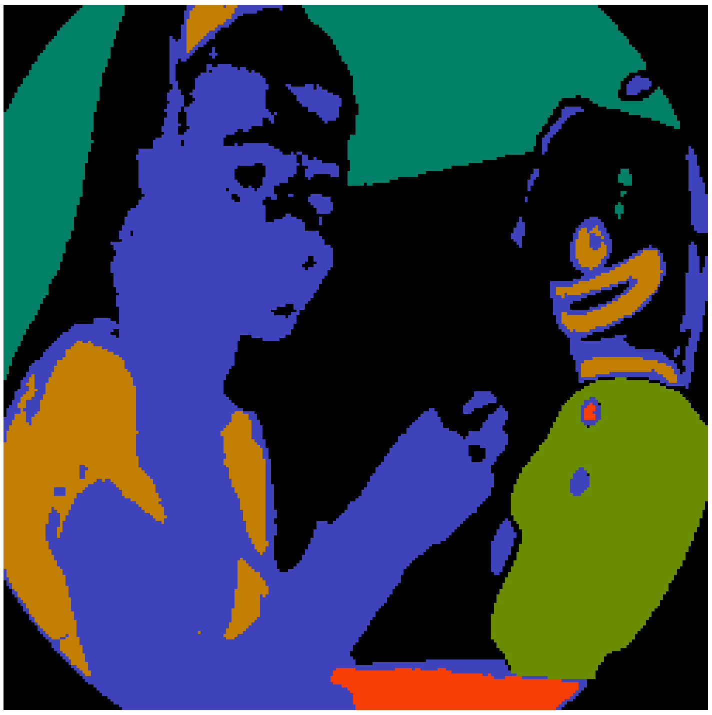


Figure 4.26: Segmentation of the colour BBC testcard image, produced with both smoothing and lighting effect removal.

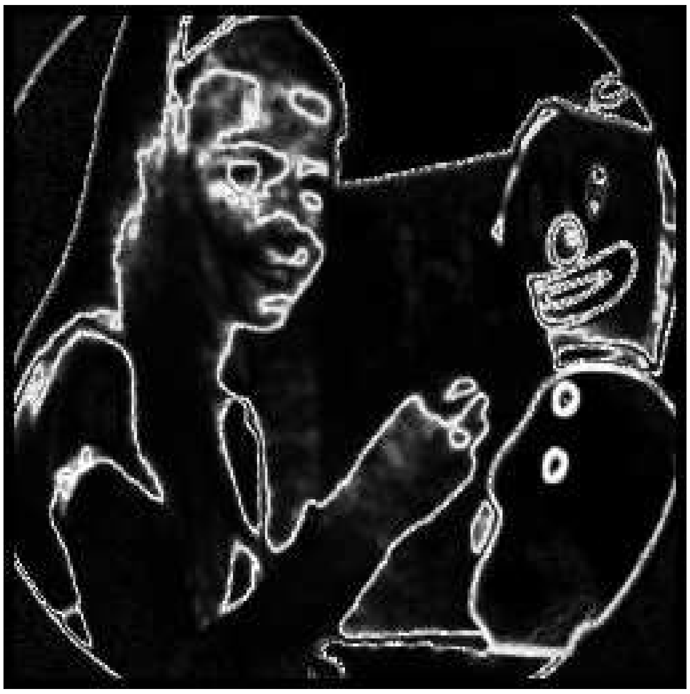


Figure 4.27: Edge probability image for the BBC testcard.

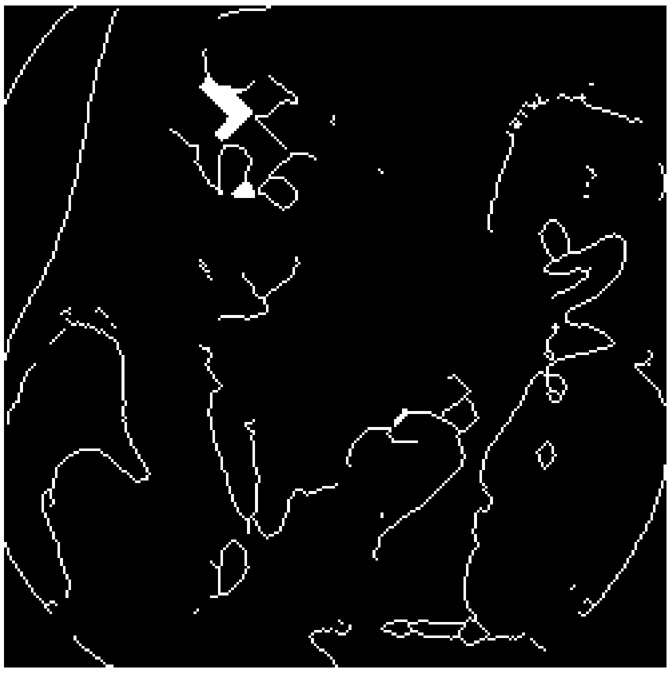


Figure 4.28: HYSTER edge map for the BBC testcard.

to be defered until after the problems with the edge probability mapping are sorted out.

4.7 The coding of the segmentation algorithm :

The following pseudocode is only intended to give an outline of the way in which the various algorithms were converted into a computer program. The full, `FORTRAN 77`, program listing is given in appendix A. This has been commented in as descriptive a manner as possible.

In the pseudocode, curly brackets have been used to indicate the parts which were coded as separate subprograms (with the name of the corresponding `FORTRAN` routine in square brackets). Comments are enclosed in hash marks.

It is clear that the program is not as modular as it might be. This is particularly true of the routines which perform the peak linkage (`FINDPK`) and the segmentation (`SEGMENT`). These two parts of the program were the most ‘under development’ and, unfortunately, there was no time available to tidy them up when they were completed.

The various parameters used, thresholds etc., were not ‘tuned’ to any great extent, so they may well not be optimal.

```
############################################################
# Pseudocode description of the multi-spectral image       #
# segmentation algorithm.  P.J.Naylor 25/5/94.             #
############################################################
[COOCSEG]{
Read in the data [DATAIN]{
  Get the dimensions of the image.
  Get the number of bands in the image.
  For each band :
    For each pixel :
      Read in the pixel value.
```

```
############################################################
# In the range -128 to 127, so that it can be stored as a  #
# single byte.                                             #
############################################################
}
If there is more than one band :
  Remove the lighting effects [NOLIGHT]{
    Define the forward transformation matrix.
    Invert the transformation matrix [MATRIX_INV].
    Modify the inverse matrix so that it uses a total
    intensity of zero.
    Form the product of the two matrices.
    For each pixel :
      Apply the final transformation matrix.
  }
Define the smoothing filter to use [FILTDEF]{
  Get the user's choice of filter type.
############################################################
# Canny (Gaussian), Cubic Spline, Petrou, or Spacek.       #
############################################################
  Get the user's choice of filter size.
  Define the appropriate filter.
}
If the filter size is greater than one :
  Convolve the image with the filter [CONVOLVE]{
    For each band :
      For each pixel which does not involve the filter going
      of the edge of the image :
        Apply the smoothing filter.
  }
Produce and analyse the cooccurrence matrices [GLCOOC]{
  Initialise the number of peaks found in each band to zero.
  For each band :
    Form the leading diagonal of the cooccurrence matrix
    [DIAG]{
      Initialise the elements of the diagonal to zero.
      For each pixel :
        For each cooccurrence direction :
          If the pair of pixels make a contribution to the
          diagonal :
            Add that contribution to the diagonal.
############################################################
# Contributions are actually included from just off of the #
# diagonal, so as to try and reduce  the amount of noise.  #
############################################################
    }
    Analyse the leading diagonal [HSTANAL]{
      Initialise the number of peaks found to zero.
      Set the thresholding and smoothing parameters.
############################################################
# The threshold is 0.5% for multiband images, and 1% for   #
# single band.  The smoothing filter length is 5.          #
############################################################
      Smooth the diagonal.
      Find the maximum in the diagonal, and record the
      value.
      While the current maximum is greater than the first
      maximum times the threshold factor :
        Calculate the standard deviation of the peak
        associated with the maximum [SIGMA]{
          Initially assume that the peak is very narrow.
############################################################
# A standard deviation of 1.0.                             #
############################################################
          While the standard deviation does not converge :
            If the difference between the area under the
            assumed peak and the area under the real peak is
            positive :
              Increase the standard deviation.
############################################################
# By 10%.                                                  #
############################################################
            If the difference between the area under the
            assumed peak and the area under the real peak is
            not positive :
              Decrease the standard deviation.
############################################################
# By 10%.                                                  #
############################################################
        }
        Subtract the found peak from the diagonal.
############################################################
# Actually, it may be necessary to subtract a but more     #
# than one peak's worth, in order to make sure that the    #
# whole of the 'wings' of the distribution are removed.    #
############################################################
        Record the details of the peak.
        Find the new maximum in the diagonal.
      Record the total number of peaks found in this band.
    }
    Sort the peaks into order [SORT].
    If more than one peak was found :
      For each peak :
        If the next peak in the list is very close :
```

```
#############################################################
# Less than 4 bins away.                                    #
#############################################################
          Merge the two peaks.
          Calculate the properties of the new peak.
          Move all subsequent peaks up the list one place.
          Reduce, by one, the record of the number of peaks
          found.
    Output the number of peaks found in this band, their
    widths, and locations.
  If there is more than one band :
    Do the peak linkage tests [FINDPK]{
      Set the lower limit on the size for the first test.
#############################################################
# 0.025% of the total number of pixels, or 200 pixels,      #
# whichever is the larger.                                  #
#############################################################
      Calculate the number of combinations of peaks
      For each group of 1000000 combinations (candidates) :
#############################################################
# This is because we can't necessarily cope with all the    #
# candidates in one go.                                     #
#############################################################
        Initialise the number of pixels associated with each
        candidate to zero.
        For each pixel :
          Find out which candidate is nearest in the feature
          space.
#############################################################
# Using hypercube decision boundaries, rather than          #
# hyperellipses.                                            #
#############################################################
          Add one to the number of pixels associated with
          that candidate.
#############################################################
# Only bother doing this if the number of pixels            #
# associated with this candidate hasn't reached the         #
# threshold yet.                                            #
#############################################################
        For each candidate :
          If the number of pixels associated with this
          candidate is greater than the lower limit :
            Add this candidate to the list of surviving
            candidates.
      Record, and output, the number of surviving
      candidates.
#############################################################
```

```
# Now try to improve the estimates of the class properties #
# a bit.                                                   #
############################################################
      For each surviving candidate :
        Initialise the associated class size, location, and
        widths to zero.
      For each pixel :
        If the pixel is close to one or more candidates :
############################################################
# ‘Close’ is defined as being within a Euclidean distance  #
# of 10.  This is to try and stop the calculation of       #
# cluster properties being corrupted by nearby parts of    #
# other clusters.                                          #
############################################################
          Classify it as belonging to the closest.
          Calculate it’s contribution to the size, location,
          and widths of the candidate with which it has been
          associated.
      Now we have a new set of candidate properties, for
      each candidate :
        Reset the class size to zero.
      For each pixel :
        If the pixel is close to one or more candidates :
          Classify it as belonging to the closest.
          Calculate it’s contribution to the size, location,
          widths, and covariance matrix of the candidate
          with which it has been associated.
      For each candidate :
        Invert the covariance matrix associated with it.
############################################################
# Using ‘Numerical Recipes’ routines.  This is done using  #
# eigenvectors/values in order to cope with the fact that  #
# the distributions have been flattened by the removal of  #
# the lighting effects.  Values which would otherwise be   #
# infinite are trapped and just made very large.  Any      #
# other inversion method would just fail.                  #
############################################################
        Sort the candidates based on the size of cluster
        with which they are associated, largest first
        [SORT4].
      Set the threshold which defines the ‘neighbourhood’ of
      a candidate.
############################################################
# This is a transformed divergence of 1.7.                 #
# The transformed divergence saturates at a value of 2.0   #
# when two vectors belong to different clusters.           #
############################################################
```

```
      For each candidate :
        Set the most maxima-like candidate to be ‘none’.
        For each candidate :
          Calculate the divergence between the current pair
          of candidates.
          Change this to the transformed divergence.
          If the transformed divergence is less than the
          threshold :
            Record which of the two candidates is most
            maxima-like.
        Overwrite the details of the current candidate with
        those of the most maxima-like candidate in its
        neighbourhood (this may be itself).
      Go through the list of candidate and remove any
      duplicate entries.
      Reset the number of classes found to the number of
      surviving candidates.
      For each class :
        Initialise the class size to zero.
############################################################
# Do a final recalculation of the cluster properties.      #
############################################################
      For each pixel :
        Classify it as belonging to the closest class mean
        vector (using a non-Euclidean distance measure).
############################################################
# The class (cluster) widths are used in the equation of   #
# an ellipse so that the distance measure equals 1.0 at    #
# the very edge of the volume which is taken to enclose    #
# the cluster.                                             #
############################################################
        Calculate its contribution to the widths of the
        class to which it has been assigned.
      Sort the details of the classes by size, largest first
      [SORT3].
        Output the number of classes found, their mean
        vectors and widths.
    }
  If there is only one band :
    For each peak :
      Make the peak details class details.
}
Segment the image on the basis of the classes found
[SEGMENT]{
  Set the number of dimensions, in cooccurrence space, to be
  twice the number of bands.
  Initialise the segmented image, and the edge probability
  image, to be blank.
  Find the minimum class width, over all the bands, for
  use as the off-diagonal width of the classes.
############################################################
# This assumes that the least elliptical cluster is        #
# circular.  This is not necessarily true though.  The     #
# off-diagonal distribution should really be analysed in   #
# order to get this value.                                 #
############################################################
  If there are more than ten classes :
    For each class :
      Find, and record, the ten classes which are most
      similar to this class.
############################################################
# This was originally done to cover situations where       #
# a hundred or more cluster centres still remained after   #
# the peak linkage stage.  It was pointless checking every #
# pixel against every class in every cooccurrence          #
# direction.                                               #
############################################################
  For each pixel :
    Initialise the non-Euclidean distance to the nearest
    cluster centre to one.
    For each cooccurrence direction :
############################################################
# ‘North’, ‘north-east’, ‘east’, and ‘south-east’.         #
############################################################
      Set the direction vector.
      Calculate the feature vector for the cooccurrence
      pair.
      Calculate the along-diagonal and off-diagonal
      components.
      If the pixel is currently unclassified :
        For all the classes :
          Calculate the distance between the feature vector
          and the class mean vector.
          If this is less than the currently recorded
          minimum distance :
            Record this as the new minimum distance.
            Record the old minimum distance as the second
            best.
            Classify the pixel as belonging to this class,
            and record the classification in the segmented
            image.
        Calculate the edge probability value, for this
        pixel, based on the distances from the feature
        vector to the nearest, and second nearest, cluster
        centres, and record it in the edge probability
        image.
############################################################
# This formula is currently very empirical.  Look in       #
# appendix A, if you have to.                              #
############################################################
      If the pixel is already classified :
        For the ten classes closest to the one that the
        pixel is already in (or all of them, if there are
        less than ten) :
          Calculate the distance between the feature vector
          and the class mean vector.
          If this is less than the currently recorded
          minimum distance :
            Record this as the new minimum distance.
            Record the old minimum distance as second best.
            Classify the pixel as belonging to this class,
            and record the classification in the segmented
            image.
        Calculate the edge probability value, for this
        pixel, based on the distances from the feature
        vector to the nearest, and second nearest,  cluster
        centres.
        If the edge probability value is greater than the
        currently recorded one, for this pixel :
          Record the edge probability in the edge
          probability image.
  Write out the edge probability image [DATAOUT]{
############################################################
# The edge probabilities are actually stored as real       #
# numbers, so they have to be scaled and converted to      #
# bytes before they are output.                            #
############################################################
    For each pixel :
      Write out the pixel value.
  }
Write out the segmented image [DATAOUT]{
  For each pixel :
    Write out the pixel value.
}
}
```

CHAPTER 5
Analysis of Remotely Sensed Imagery

The use of remotely sensed imagery in geological/mineralogical exploration, and environmental monitoring is reviewed. The image segmentation program based on the algorithm described in chapters 2, 3, & 4 is then applied to the sort of images which are of interest in these fields.

5.1 Geological & mineralogical surveying applications :

5.1.1 The requirements :

The chief purpose of carrying out a geological, or mineralogical, survey is to locate deposits of useful (or otherwise valuable) materials, in order to be able to extract them. Such materials could include precious metals [Bedell, et al., 1990], base metals [Banninger, 1990], petrochemicals [Stavtser & Karasev, 1990], and construction materials (e.g. limestone, clays, and aggregates) [Stone, 1989].

5.1.2 The role of remotely sensed imagery :

The use of remotely sensed imagery, obtained from either airborne, or spaceborne, platforms, allows surveying to be performed in areas in which it would be expensive, difficult, or even impossible, to carry out field surveys.

It also allows a much wider view to be taken. A single LANDSAT image covers an area roughly 185 km square, whereas a field survey will only give details local to the sites visited.

Having located an area of interest one can always then (if practical) make a field survey of the region, in order to test the validity of the original analysis, and provide more information.

5.1.3 The problems associated with the use of remotely sensed imagery :

There are two problems associated with using remotely sensed imagery, in this context. The first is that the objects of interest are quite often obscured by soil and/or vegetation. Or, if they are not obscured, then their surfaces may have been altered by weathering. (Vegetation and weathering can give clues to underlying structures though, see §5.1.4.)

The second problem is that one has to be careful when deciding what resolution to work at. At too low a resolution, each pixel will contain a mixture of different types of material, this will require the application of 'mixture modelling' techniques [Foody & Cox, 1991]. At too high a resolution, a segmentation of the image will begin to fragment, due to 'unwanted' inhomogeneity in the data (see §5.2.3 for an example of this).

5.1.4 The information contained in the images :

Mineral deposits are, in general, small and buried. Thus, their detection relies on the identification of the larger scale consequences of their existence, or on the location of the sort of geomorphology which is required for (but does not necessarily guarantee) their formation. For example: the presence of metal ores will result in the alteration of the chemical composition of the rocks around them; and a combination of permiable rocks overlain by impermiable ones, and faulting, will result in regions where petrochemicals may become trapped.

In arid areas, where there is little vegetation, the identification of surface geological units can be made directly from their spectral responses. Where there is weathering the responses will be modified from those which are obtained from laboratory samples, but different rocks should still be distinguishable. Also, weathering imparts a texture to the surface which may be characteristic of the rock type (in general, sedimentary rocks will be smooth, and igneous rocks rough).

In non-arid, and vegetated, areas problems arise due to absorption features being introduced into the spectra by the presence of water, and due to the geological units being obscured by plant life. The presence of water in a rock (and the quantity) can, however, be an important identifying feature, and vegetation can itself play a role in the location of mineral deposits. Geobotany, as the study of the effect on plant life of minerals is known, provides information on two fronts. Firstly, particular plant species are known to be, relatively, more abundant around deposits of particular minerals, since they are more tolerant of high concentrations of those minerals in the soil and groundwater. Secondly, plants may be only partially 'poisoned' by the presence of the minerals, so that they display a spectral signature which differs from that of a 'healthy' plant [Xu, et al., 1990].

Further information about surface, and sub-surface, geology can be obtained from thermal infra-red (TIR) images, as a result of the physical characteristics of rocks and soils. Figure 5.1 shows how warm and cool areas in a pre-dawn TIR image can be related to subterranean structure [Loughlin, 1990]. Also, the difference between day-time and night-time TIR images of the same scene can be used to distinguish between rock types on the basis of their thermal inertia [Sabins, 1987].

Faulting will express itself in remotely sensed imagery as linear features,

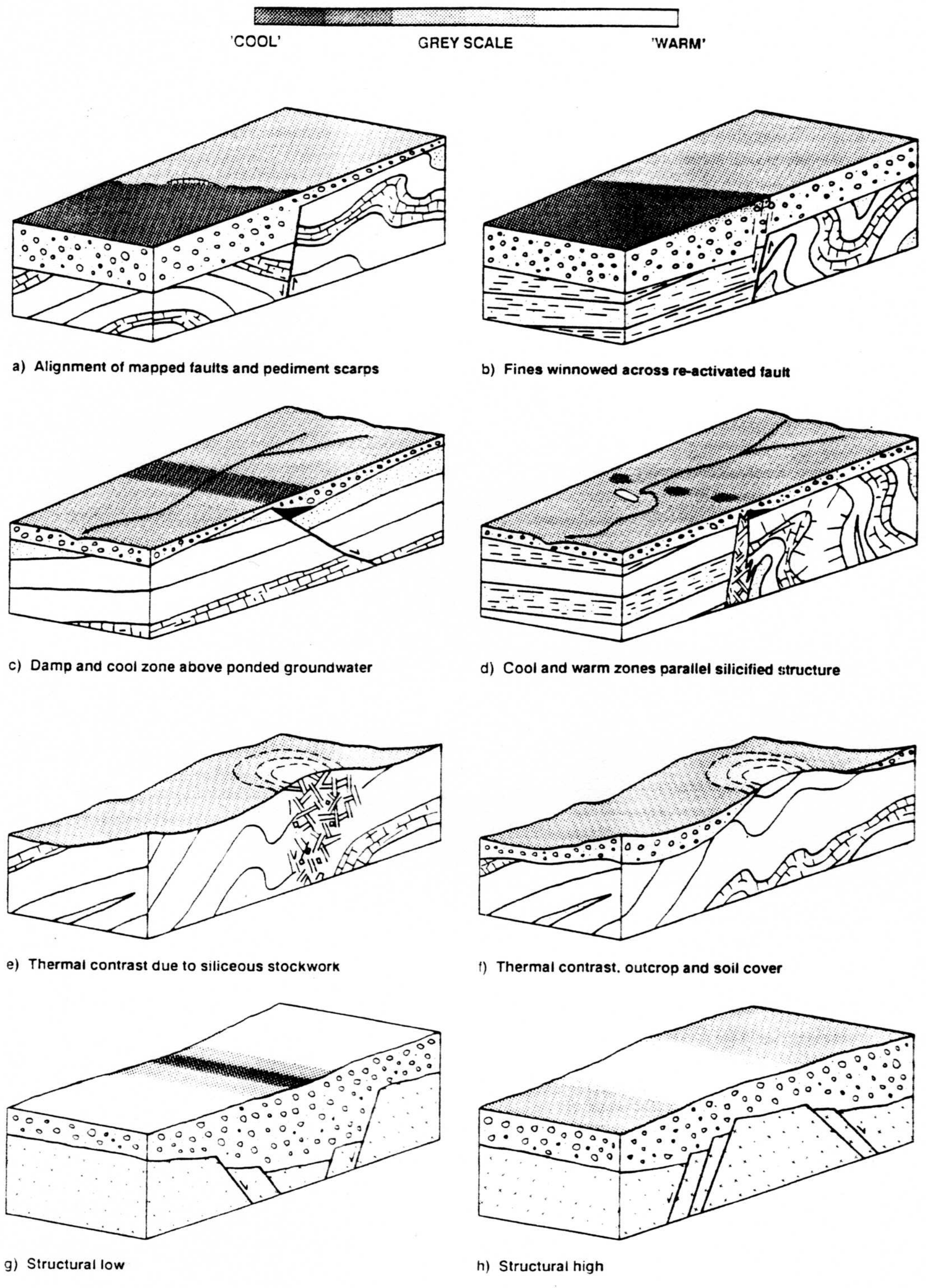

'COOL'
GREY SCALE
'WARM'
a) Alignment of mapped faults and pediment scarps
b) Fines winnowed across re-activated fault
c) Damp and cool zone above ponded groundwater
d) Cool and warm zones parallel silicified structure
e) Thermal contrast due to siliceous stockwork
f) Thermal contrast. outcrop and soil cover
g) Structural low
h) Structural high

Figure 5.1: Pre-dawn thermal infra-red image interpretation [Loughlin, 1990].

with lengths varying from a few metres to hundreds of kilometres. Apart from their importance in causing petrochemical traps (as mentioned above), they also provide sites at which mineralisation can occur [Stefouli & Osmaston, 1984]. Not all linear features in a remotely sensed image are going to be faults though. One must take into account other natural line-like objects, such as streams, and man-made objects, such as roads and pipelines.

5.1.5 The role of automated analysis :

The features described above can be very subtle and cover, relatively, small areas. A single (185km $\times$ 185km) LANDSAT scene plotted at 150 dpi (6 pixels/mm) will result in an image roughly 1 metre square. The analysis, by eye, of such an image requires specialist training, patience, and concentration. Also, as mentioned before, only three spectral bands can be presented to a human interpreter at any one time.

An automatic analysis program, provided that it is written in conjunction with someone who has the necessary analytical know-how, suffers from none of the above limitations. Therefore, it presents a cheap, and efficient, tool for isolating areas of interest, to be further investigated by (human) eye. Segmented images, with overlain edge images, can be used as geological maps of an area (provided that they are used with care). And, finally, with the attachment of a suitable 'expert system', there is the, future, possibility of a full-blown 'X marks the spot' exploration technique.

5.2 Land use & environmental monitoring applications :

5.2.1 The requirements :

Land use applications include census type analysis, e.g. the determination of the percentage of a particular region given over to particular land uses, or crop types [Barnsley, et al., 1991; Sharman, 1989; Bocchi, et al., 1989]; and the identification of areas of a given single land use, e.g. the location of illegal crops [Image Processing magazine, 1992].

Environmental monitoring applications would usually involve looking for changes in land use from one epoch to another. This could be in order to check that environmentally sensitive areas are not being damaged [Singh, 1984; Cross, 1989], or in order to ensure that environmental commitments, are being successfully met [Smith & Vaughan, 1991].

5.2.2 The role of remotely sensed imagery :

The expense, accessibility and large scale coverage factors are the same as for geological/mineralogical applications. There is also the factor that, whilst geologically/mineralogically interesting objects change on geological time scales, the objects of interest in land use/environmental monitoring applications vary on much shorter time scales. For example: urban development will cause changes on the scale of years/months; forest fires/deforestation and crop cycles will cause changes on the scale of months/weeks; and the spread of an oil slick will cause changes on the scale of weeks/days. Therefore, whilst it would be feasible to carry out a one-off field survey for a geological application, such is not the case with land use and environmental monitoring. Regular imaging of the same area is one of the features of spacebourne remote sensing systems: LANDSAT 4 & 5 have a repeat cycle of 16 days; and SPOT (which has a normal repeat cycle of 26 days) has the ability to look sideways, giving a possible repeat cycle of 2 or 3 days, depending on latitude. Airbourne systems may be flown as, and when, necessary (funds permitting).

5.2.3 The problems associated with the use of remotely sensed imagery :

The main problem is that of choosing the most appropriate resolution to use for a given application. Too low a resolution and the boundaries between different land use types cannot be accurately determined. Too high a resolution and one begins to see 'unimportant' variations in land use, e.g. trees or small areas of grass in urban areas, or patches of bare earth in cultivated fields (although the latter would, of course, be important if the application was a calculation of farming efficiency [Malthus, et al., 1990]).

5.2.4 The information contained in the images :

Unlike in geological/mineralogical applications, the objects of interest in land use/environmental monitoring applications are directly recognisable. Different materials and different crop types will have different spectral responses [Richards, 1986]. Variations from the ideal response, for crops, will indicate disease, or some other source of stress (drought, infertile soil, etc.) [Ali & Aggarwal, 1977; Danson, et al., 1990].

Temporal variations in spectral response allow the tracking of the spread of disease in crops, or they can be used to chart the ripening process and, thus, determine the best time for harvesting [Kauth & Thomas, 1976].

Fires will, of course, be well defined in thermal infra-red imagery [Muirhead & Cracknell, 1984].

5.2.5 The role of automated analysis :

As with geological/mineralogical applications, the limitations of human interpretation, and the need for map production are good reasons for using an automated approach. Also, in particular in land use applications, one may require precise numerical information, such as: the percentage of a crop affected by disease; or the area of a town which is given over to residential use compared with the area occupied by green spaces. Such details are readily obtained from a suitably segmented image.

5.3 Results of image segmentations :

Figure 5.2 is a false colour composite of the first three principal components of part of a LANDSAT TM image of a mountainous region in the Middle East. Figure 5.3 shows the result of segmenting this image using the techniques outlined in chapters 2–4. Bands 1–5, & 7 of the image were used (since the thermal infra-red band is at a different resolution). The data was smoothed with a seven point diameter circular Spacek filter prior to segmentation.

Unfortunately, the geological map of this area was not available for inclusion in this work. However, since the segmentation is only done on the basis of colour, it should really only be compared with the actual data, rather than another interpretation of the scene which has been produced using far more information.

Six classes have been found in the data. The only two which are readily identifiable with known features are the regions of deep shadow (coloured greeny-blue in the segmentation), and the river (coloured tan). There are a large number of shadow regions due to a combination of the rugged nature of the terrain and the Sun being low in the sky. The other classes correspond roughly to the variations in the underlying rock type, but the segmentation is rather more fragmented than one would want, if it was to be used as a geological map. However, it would appear to be a fairly faithful representation of the colour variations in figure 5.2.

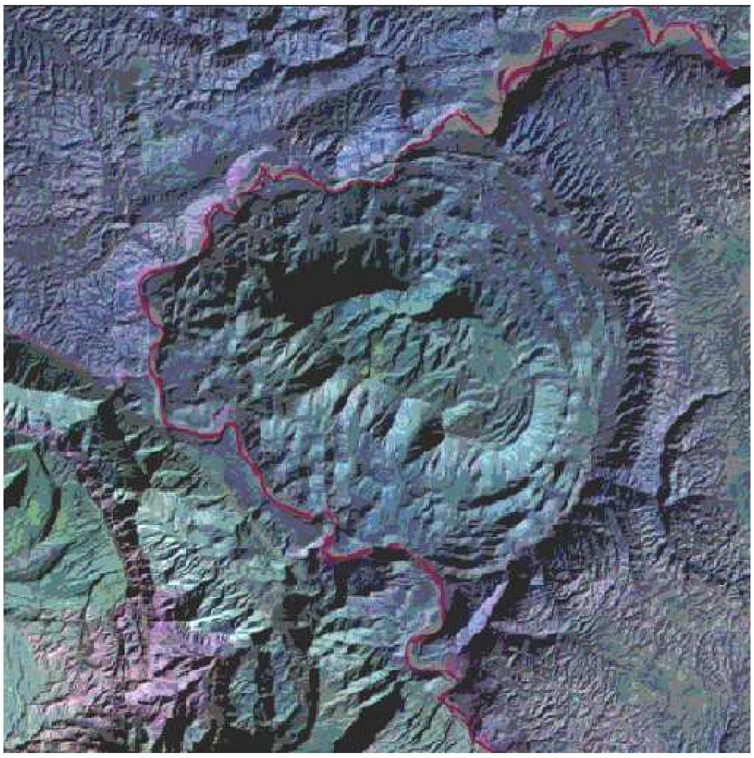


Figure 5.2: False colour composite of LANDSAT TM scene.

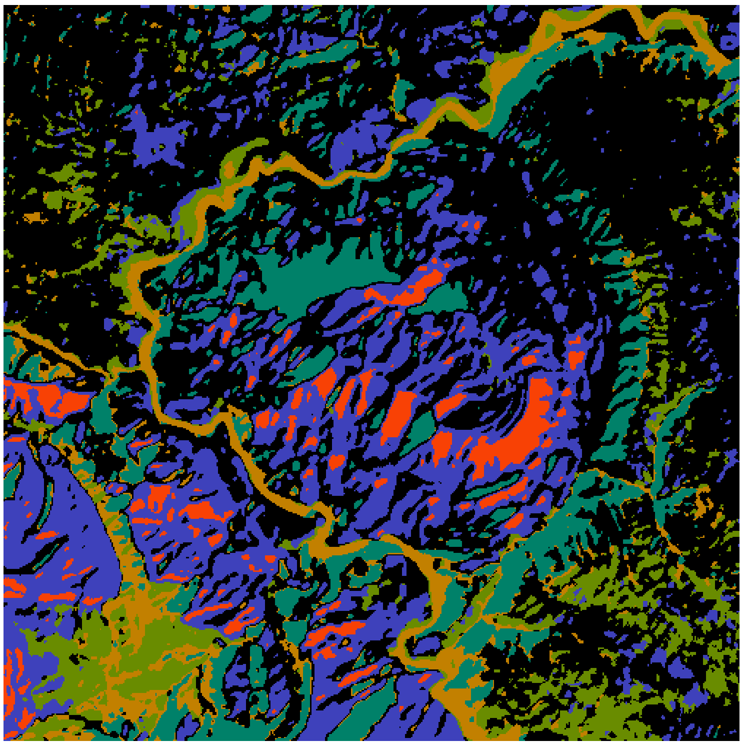


Figure 5.3: Segmentation of LANDSAT TM scene.

This is as much as can realistically be asked of such a low level algorithm. Any attempt to produce at actual map (including identification of rock types) would require the addition of some sort of artificial intelligence (or knowledge based) stage.

Figure 5.4 is a false colour composite of the first three principal components of the last seven spectral bands of part of an airborne thematic mapper (ATM) scene showing the town of Blewbury, in Oxfordshire (there are eleven spectral bands, and the spatial resolution is 5m). Figure 5.5 is the segmentation (using the last eight bands - the first three contain atmospheric distortions) produced using the same method as for the previous data set. Figure 5.6, and table 5.1, show the land use types in the area. Figure 5.7 is another part of the same scene showing Churn Farm. Figure 5.8 is its segmentation (using all eleven bands), and figure 5.9, and table 5.2, show the land use. Note that the colours used for displaying the segmentations and reference images are not intended to match.

Nine classes have been found in the Blewbury image, and thirteen in the Churn Farm one. There are three sources of differences between the segmentations and the reference images. Firstly, there are regions which are classified differently in the reference images but the same in the segmentations. These are either due to unnatural divisions in the reference images (e.g. differentiating grassy areas on the basis of what they are used for), or to there being no, discernable, spectral difference between the different land covers (e.g. different cereal crops, or an unripe cereal crop and grass). This can be seen in figure 5.5 where most of the grass, and some of the barley, has been put in one class (coloured aquamarine), and where most of the fields on the righthand side of the image have been put in the same class (mauve) regardless of use.

Secondly, there are regions which are classified the same in the reference images but differently in the segmentations. These are either due to different varieties of the same crop being sown in different fields, or to their being at different stages of development. This is particularly obvious, in both images, where the winter wheat is concerned. In figure 5.5 this crop is split between two classes (black and mauve) with some fields being wholly in one class, or the other, and some being split between the two (the top left quadrant of the figure). In figure 5.8 the central field is split between three classes. It would be interesting to know whether the fields were all sown at the same time, whether or not they were all treated in roughly the same way, and where the date when the image was taken fits into the plant's development cycle. It could well be that some fields contain ripe wheat (the black class in figure 5.5), some contain unripe wheat – which is being confused with the grass (the mauve class in figure 5.5), and some contain plants which are in the process of ripening (the inhomogeneously segmented fields).

Thirdly, there are regions which are homogeneously classified in the reference images but not in the segmentations. These are either due to the regions not actually being homogeneous (e.g. the 'urban', and 'farm building' classes), or to there being some hidden variation within the region (e.g. soil quality, or different types of tree in wooded areas). This point can be seen by looking at the variation in colour, within the fields, in the false colour composites.

All of these differences really just go to reinforce the point that it is far too simplistic a view to assume that colour, alone, can be used to characterise an object, or region. Also, that the 'idiot savant' segmentation algorithm is of little use for extracting information from a scene, without the addition of some sort of 'intelligent' (artificial, or otherwise) knowledge based system.

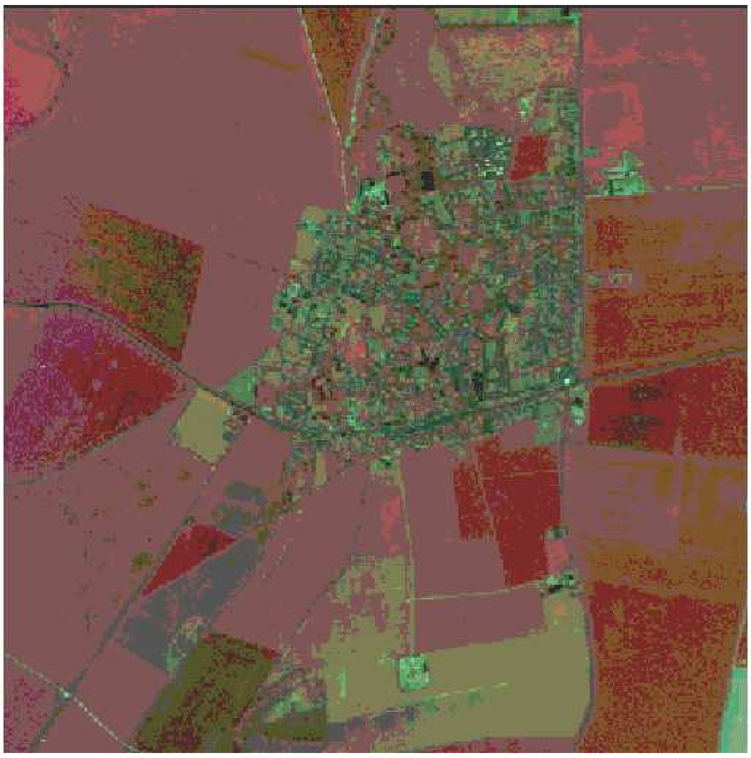


Figure 5.4: False colour composite of Blewbury ATM scene.

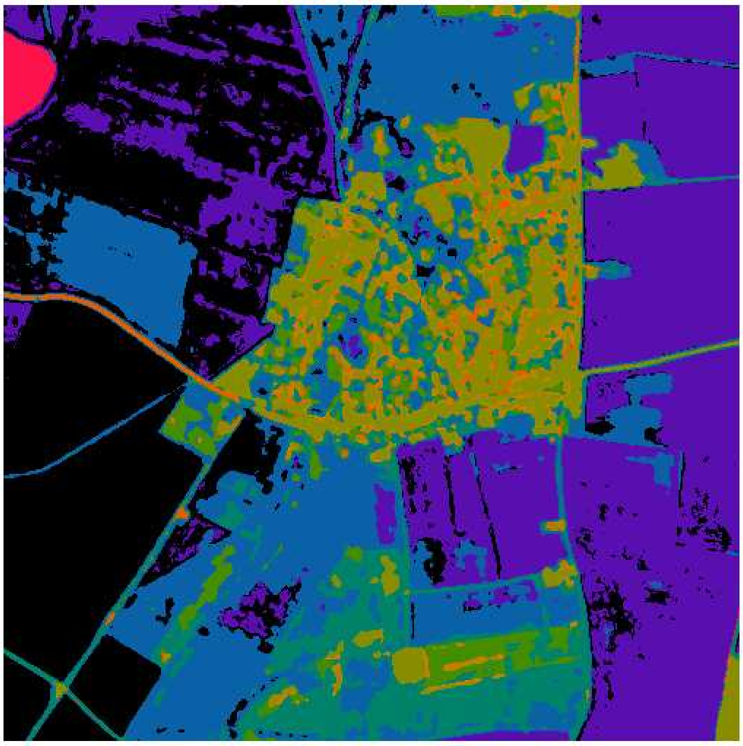


Figure 5.5: Segmentation of Blewbury ATM scene.

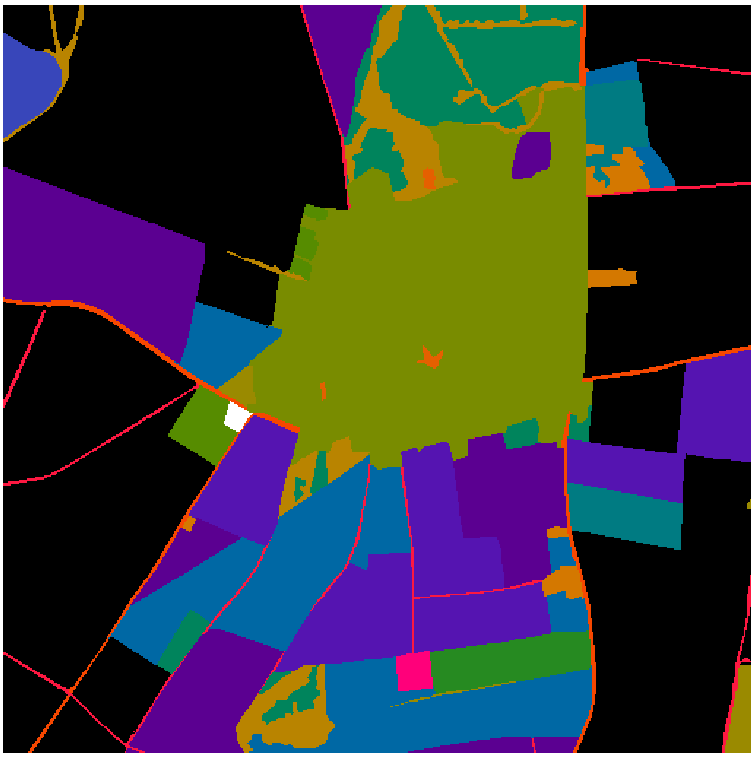


Figure 5.6: Blewbury ATM scene, reference data.

⬛	Winter Wheat
🟪	Winter Barley
🟪	Spring Barley
🟦	Oil Seed Rape
🟦	Pasture
🟦	Ley Grass
🟩	Other Grass
🟩	Peculiar Grass (no pesticides used)
🟩	Recreational Grass
🟩	Residential (built-up)
🟨	Bare Ground
🟫	Trees
🟧	Farm Buildings
🟧	Water
🟥	Roads (metalled)
🟥	Tracks and Paths
🟥	Water Pumping Station
⬜	Cemetery

Table 5.1: Land use types in Blewbury ATM scene.

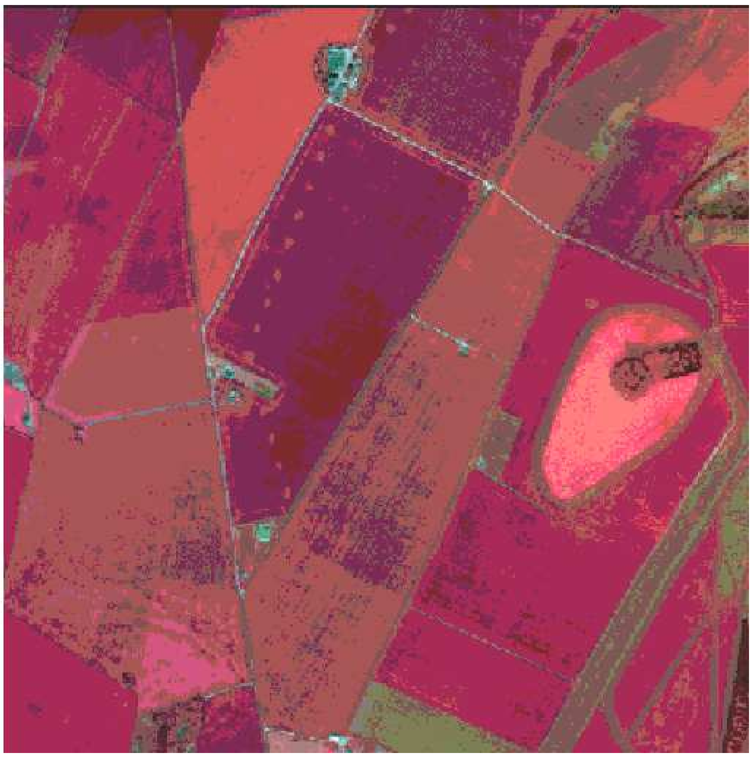


Figure 5.7: False colour composite of Churn Farm ATM scene.

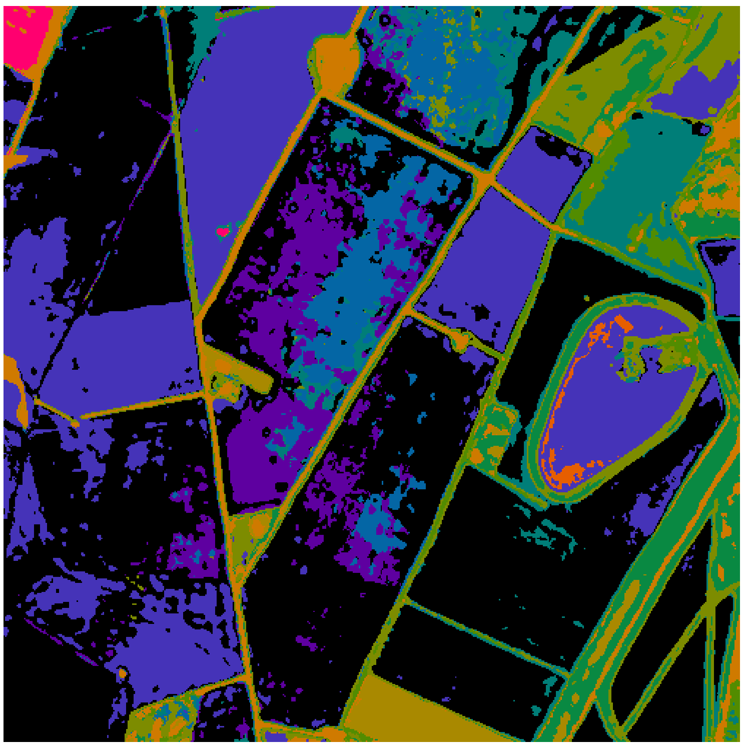


Figure 5.8: Segmentation of Churn Farm ATM scene.

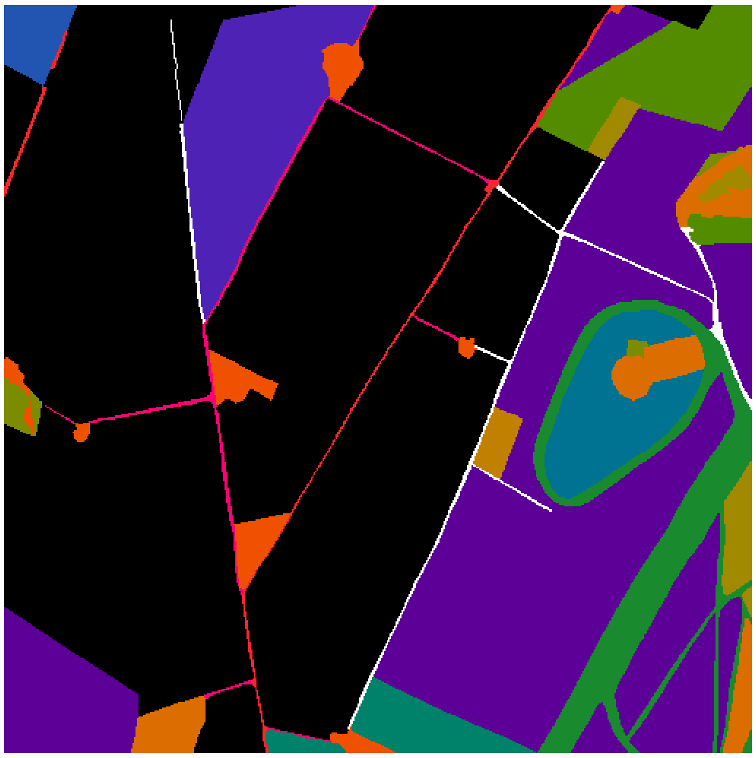


Figure 5.9: Churn Farm ATM scene, reference data.

⬛	Winter Wheat
🟪	Winter Barley
🟪	Winter Beans
🟦	Peas
🟦	Lucerne
🟩	Ley Grass
🟩	Gallops
🟩	Pasture
🟩	Other Grass
🟨	Scrub
🟫	Plantation
🟧	Trees
🟧	Buildings
🟥	Metalled Roads
🟥	Concrete Tracks
⬜	Tracks and Paths

Table 5.2: Land use types in Churn Farm ATM scene.

When taking a more general view of the quality of the segmentations, three points should be noted. Firstly, the combination of the removal of the lighting effects, plus the Spacek filtering, plus the use of directional information from the cooccurrence matrices, means that the regions are, in general, large and unbroken. This is not usually the case with class based segmentations. It is, therefore, unnecessary for any post-processing to be carried out to improve the local homogeneity of the segmentation (e.g. the relaxation labelling mentioned earlier, or Besag smoothing [Besag, 1986]). This can represent a considerable time saving.

Secondly, although the Spacek filtering has reduced the effects of the noise, it has caused narrow regions to become enlarged (e.g. the river in figure 5.3), and neighbouring colours to 'bleed' into one another (as was seen with the segmentation of the testcard image in chapter 4). This second effect results in a spurious region appearing between any two real ones which are of very different colour. This is particularly noticeable around the edges of the 'gallops' in the Churn Farm segmentation. These problems will occur no matter which of the 'proveably optimal' filters are used. They are all, at heart, just weighted mean filters. It should be possible to obtain much better results using an adaptive filter, such as the modified trimmed mean (MTM) filter [Lee & Kassam, 1985; Lee & Tantaratana, 1990], which will smooth the regions whilst leaving the edges sharp. Unfortunately, there has not been sufficient time for such a filter to be implemented in this work.

Thirdly, there is the problem of the regions which appear to be of quite different colour in the colour composites, but which the algorithm puts in the same class. The green and red field halfway up the lefthand side of figure 5.5 is one of the more obvious examples of this. The simple explanation of this phenomenon is that it is merely a case of misclassification. However, there is a situation which would give rise to exactly the same result even if the algorithm was working perfectly. Consider a cluster which has such a large variance, in one, or more, dimensions, that the data points which correspond to one end of the cluster are of a distinctly different colour to those at the other end. A human interpreter, looking at the data points in image space, will conclude that there are two, or more, distinct sets of data points in the image. A cluster analysis algorithm, looking at the same data points in feature space, will conclude that there is only one set of data points. Which of the two interpretations is correct ?

As far as the identification of objects in the imaged scene is concerned, it is the cluster analysis algorithm which has arrived at the correct conclusion. There **is** only one class of object in the scene, albeit one with a large degree of variance in it. The human interpreter has been "fooled" by his reliance on colour for the analysis of the image. We could reformulate the segmentation problem strictly in terms of identifying regions of different (humanly) distinguishable colours, in which case the human interpreter would be correct, and the algorithm incorrect. What we cannot do, unfortunately, is reconcile the two views – the human interpreter cannot cope with visualising the high dimensional feature space involved, and the algorithm cannot be given any concept of "colour". The two views are constrained to always give differing interpretations of the same data set. It would be nice to be able to produce

quantitative values for the quality of the segmentations presented in this chapter. However, as we have seen, it is extremely tricky comparing a segmentation with an analysis produced by other means, either objectively (as in the case of the land use maps), or subjectively (as in the case of the human interpretation of the colour composites). In the final analysis the only measure of the quality of a segmentation is how useful it is in solving the problem which you are addressing. In the case of these segmentations, they are not especially useful when it comes to land use identification (since they are far too fragmented), but they might prove interesting to someone working on an analysis of crop disease, or soil fertilty.

This may appear to be something of an anticlimax, but it seems foolish to attempt to claim to have comprehensively solved the problem of multi-spectral image segmentation when the precise nature of the problem is itself in some doubt.

CHAPTER 6
Image Compression

Various image compression techniques are reviewed. The advantages to be gained from using the segmentation technique, which has been developed in this work, as part of one of these techniques is discussed, and some results are presented.

6.1 The requirements :

Sometimes it is necessary to greatly reduce the size of the data set, associated with an image, without any (or with a minimum) loss of information, or image quality. This may be because the data has to be transfered very quickly over narrow bandwidth communications channels (e.g. high definition television (HDTV), or video telephones); or because there are extremely large, and growing, quantities of data which need to be permanently archived (e.g. data from remote sensing satellites).

There are eight basic classes of data compression, which are characterised by whether they provide perfect (lossless) or visual quality reconstruction of the data, and by whether the encoding, and decoding, processes are fast (video rates to seconds) or slow (minutes to hours). Different classes are appropriate to different applications. For example, HDTV, and video telephones, require visual quality, fast encoding, and fast decoding. Archival storage, on the other hand, might be better served by lossless, slow encoding, and fast decoding (depending on the exact specifications of the application).

6.2 Existing techniques :

The basis of most techniques is 'run length encoding' [Gonzalez & Wintz, 1987]. This involves replacing a sequence of identical data elements with a token, representing the data value(s), and the number of occurrences. There also needs to be a 'look-up table' to allow the tokens to be mapped back to data values.

For example, figure 6.1 shows the pixel values in a small (8 by 8 pixels), single band, 'image'. This has been represented using 66 bytes – two to record the dimensions of the image (in the 'header') and one for each of the pixel values (in the 'body'). This is a rather basic scheme, since it only allows us to represent single band images with a dynamic range of no more than 256 grey levels, and a maximum size of 256 by 256 pixels. More general schemes are possible, obviously, but these require far more complex headers.

```
The header:     8  8
The body:      16 16 16 16 16 16 16 16
               16  8  8  8  8 16 64 16
               16  8  8  8  8 16 16 16
               16  8  8 32 32 32 32 16
               16  8  8 32 32 32 32 16
               16 16 16 32 32 32 32 16
               16 64 16 16 16 16 16 16
               16 16 16 16 16 16 16 16
```

Figure 6.1: A simple 8 by 8 pixel, single band, 'image'.

Figure 6.2 shows the result of run length encoding this image. The header now contains not only the dimensions of the image but the look-up table, preceeded by a byte indicating the number of entries in the table (4). The table indicates that token 1 maps to a pixel value of 16, token 2 to a value of 8, etc.. Note that, in the case of single band images we can actually do without the look-up table, since the mapping is one to one, i.e. we could use the actual pixel values as tokens. The body of the run length encoded image consists of a series run length/token pairs, with the runs going from left to right along each row of pixels, starting at the top left corner. Note that, since we know the dimensions of the image, we can consider runs which 'wrap around', from the end of one row to the beginning of the next, as single runs, rather than as multiple runs which terminate at the end of each row. So, for example, the run length/token pairs start off with `9 1 4 2` rather than `8 1 1 1 4 2`.

```
The header:    8  8  4 16  8 64 32
The body:      9  1  4  2  1  1  1  3  2  1  4  2  4  1
               2  2  4  4  2  1  2  2  4  4  4  1  4  4
               2  1  1  3 14  1
```

Figure 6.2: Run length encoded version of the image in figure 6.1.

Some parts of the image now take more bytes to encode than they did originally – the two single pixels, with value 64, now require two bytes each rather than just one, for example. Some take the same, for example the two pixel runs, with value 64, down the edges of the image. However, because the long runs of pixels now take far fewer bytes to encode, we have, overall, managed to compress the amount of image data from 66 bytes to 49, a compression ratio of 1.3:1. If we do without the look-up table (so the run length/token pairs become `9 16 4 8 1 6 1 64 ...`), this reduces further, to 44 bytes, a compression ratio of 1.5:1. Note that, absolutely no information has been lost due to the encoding process, the original data can be reconstructed perfectly from the compressed data. This is done by simply expanding each of the runs, and substituting the appropriate data value(s) for each of the tokens.

Doing a run length based encoding of the BBC testcard image (see figure 6.9) results in a compression ratio of 1:1.02 – i.e. the file actually gets larger. This is because real images tend not to contain particularly long runs of pixels which are all the same.

Run length encoded data can be further compressed, still allowing perfect reconstruction, using 'Huffman coding' [Gonzalez & Wintz, 1987]. In its ideal form this involves determining the frequency of each occuring run length/token pair, then replacing the most frequently occuring ones with tokens which require only a small amount of storage (less than a byte), and replacing those which occur infrequently with larger tokens.

Table 6.1 shows the frequency of occurrence of each of the run length/ token pairs in the body of the run length encoded image in figure 6.2.

Run length/ token pair	Frequency	Run length/ token pair	Frequency
1 1	1	4 2	2
9 1	1	1 3	2
14 1	1	2 1	3
4 1	2	4 4	3
2 2	2		

Table 6.1: Frequency of run length/token pairs in figure 6.2.

Figure 6.3 shows how the Huffman codes for the pairs are generated. The first stage involves ordering the pairs according to their frequency of occurrence, and then, two at a time, combining the pairs which occur least frequently, until only two pairs remain. At each step, the frequency used is the total frequency of the previously combined pairs, and the ordering of pairs which occur with the same frequency is immaterial. In the second stage, the tree structure, which was created in the first stage, is traversed in the opposite direction (right to left, in the figure) and every time the tree branches a `0` is appended to the bit pattern of the Huffman code on one branch, and a `1` is appended to the code on the other. The lefthand side of the diagram shows

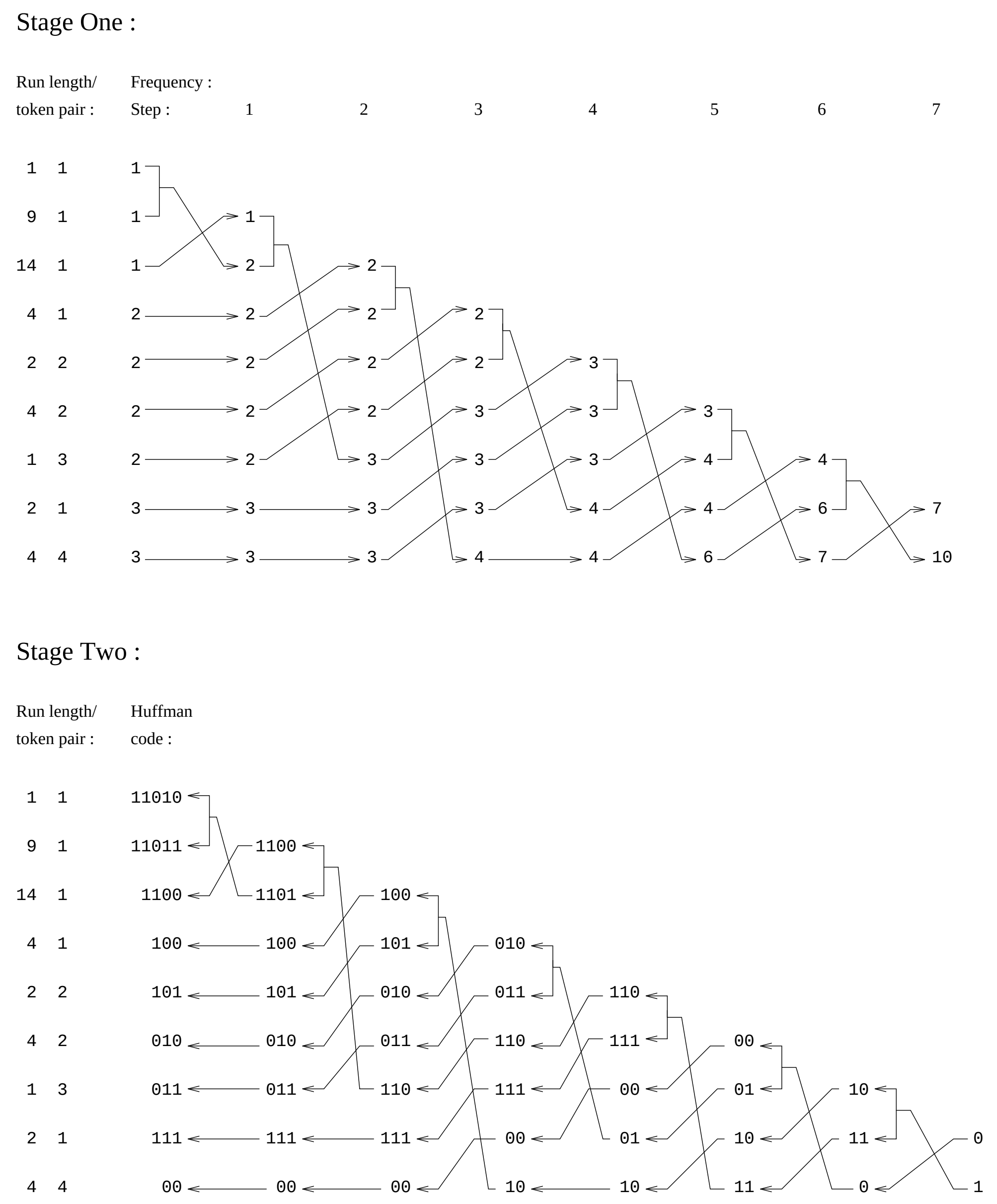


Figure 6.3: Producing the Huffman codes for the run length/token pairs in figure 6.2.

the run length/token pairs and their corresponding Huffman codes. There are three things to note about these codes. Firstly, they are only one of several possible ways of Huffman coding the run length/token pairs. We could have generated a completely different set of codes by combining different sets of pairs in the first stage (whenever the frequencies were the same), or by assigning the `0`'s and `1`'s to the branches in a different order in the second stage. Secondly, as stated earlier, the longest Huffman codes have been assigned to the least frequently occurring run length/token pairs, and the shortest codes to the most frequently occurring. Finally, and probably most importantly, there is no pair of codes, of length i and j ($j \geq i$) for which the first i bits of the two codes match. Without this property it would be impossible to separate the, variable length, Huffman codes from each other, when it comes to decoding the data.

Encoding the body of our example image, using the Huffman codes that we have just generated, reduces its size to 53 bits, a mere 6.625 bytes – giving a compression ratio, for the body alone, of 9.7:1. However, we will have to add a second look-up table to the header in order to facilitate the mapping between the Huffman codes and the run length/token pairs. This will require 28 bytes – one to give the length of the table, followed by a triplet of bytes for each of the nine table entrys (one for the Huffman code, and two for the associated run length/token pair). Together with the two bytes for the image size, and five for the token to pixel value look-up table, this gives a total size of 41.625 bytes. This will need to be stored as 42 bytes, giving a compression ratio of 1.6:1. This is not much better than that achieved by run length coding alone. The dramatic decrease in the size of the body has almost been made up for by the increase in the size of the header.

For this reason it is usual to have several pre-defined Huffman coding schemes, for different distributions of run length/token pairs. One then encodes the data using the scheme associated with the distribution which most closely matches that of the actual run length/token pairs, and a single token is placed in the header to denote which of the schemes should be used for decoding. The chosen set of codes are unlikely to encode the body quite as efficiently as the proper ones would, but this will be much more than made up for by the absence of the Huffman code to run length/token pair look-up table. If we assume that our sample image would require 8 bytes to encode the body using a suboptimal scheme (rather than 6.625), the total size of the data describing the image would now be reduced to 16 bytes (the header will contain 8 bytes – two for the image size, five for the token to pixel value look-up table, and one for the Huffman scheme token). This gives a compression ratio of 4.1:1.

Doing a Huffman based encoding of the run length encoded version of the BBC testcard image gives a compression ratio of 1.2:1. Again, this is not as good as the simple example because of the relatively small number of long runs which occur in real images.

With dedicated hardware, there is no reason why run-length encoding, and Huffman coding, cannot be implemented at video rates.

In order to reduce the size of the data set further one needs to preprocess the image so that it becomes more locally homogeneous, so that the average run length is increased. This, inevitably, leads to a loss of information, so the resulting reconstruction of the image is imperfect.

One way of doing this is the discrete cosine transform (DCT) method [ISO/JTC1/SC2/WG8 N800]. In this technique the image is split up into small square patches (say 8 by 8 pixels), and each is then replaced by its two dimensional discrete cosine transform [Press, et al., 1986]. Up to this point we have lost no information, however, now the pixel values in the resulting 'image' are scaled and quantised so that they all lie in the range 0 to 255, and cover as much of that range as possible – this results in the loss of some information about the original image. Next, some of the high frequency components in the transformed patches are set to zero, this reduces, quite considerably, the amount of information contained in the data. The 'image' is then run length encoded, with the runs going diagonally across it, so as to maximise the length of the runs with values of zero. Finally the whole thing can be Huffman coded. As before, the last two steps involve no further loss of information.

The image is reconstructed by reversing the Huffman coding, and expanding the runs to create the, modified, transformed patches. The pixels which were not set to zero are scaled back to their original values, and the inverse discrete cosine transform is applied to each of the patches. The quality of the resulting reconstruction will depend on two factors, the patch size used (this will determine how 'blocky' the reconstruction is), and how much of the high frequency information was discarded (this will determine how much fine detail is lost within each patch).

Figures 6.4 to 6.8 show some of the steps involved in applying this technique to the simple image in figure 6.1. Firstly the image data is split into 4 by 4 patches, and the patch size is recorded in the header. Secondly, each patch is replaced by its discrete cosine transform (n.b. the zero frequency component is in the top lefthand corner of each patch). Next, the new pixel values are scaled and quantised, and the original maximum and minimum values are stored in the header (n.b. they are real numbers, so they will require four bytes each to store). The fourth stage is the discarding of high frequency components from the patches. In this example we shall discard all but the two lowest frequency components in each direction, i.e. all of the pixels

```
The header:    8  8  4
The body:     16 16 16 16 | 16 16 16 16
              16  8  8  8 |  8 16 64 16
              16  8  8  8 |  8 16 16 16
              16  8  8 32 | 32 32 32 16
              -------------------------
              16  8  8 32 | 32 32 32 16
              16 16 16 32 | 32 32 32 16
              16 64 16 16 | 16 16 16 16
              16 16 16 16 | 16 16 16 16
```

Figure 6.4: DCT encoding, stage one - split the image into patches.

```
The header:  8   8   4
The body:   208.0  47.0  24.0  81.0 | 336.0  75.3 -64.0  52.7
             47.0  28.0   0.0   4.0 |  58.3  -9.0 -39.6   7.0
             24.0   0.0  -8.0   0.0 |   8.0   8.0   8.0   8.0
             81.0   4.0   0.0  28.0 |  69.7  41.0  39.6  25.0
            -------------------------------------------------
            320.0  70.0   8.0  58.3 | 352.0 118.6   0.0  73.4
             75.3  -9.0   8.0  41.0 | 146.0  62.6   0.0  24.0
            -48.0 -50.9   8.0  50.9 |  48.0  27.3   0.0   4.7
             52.7   7.0   8.0  25.0 |  78.1  24.0   0.0  17.4
```

Figure 6.5: DCT encoding, stage two - form the discrete cosine transform of the patches.

```
The header:      8   8   4 352.00 -64.00
The body:      167  68  52  89 | 247  86   0  72
                68  57  39  42 |  75  34  15  44
                54  39  35  39 |  44  44  44  44
                89  42  39  57 |  82  65  64  55
              ---------------------------------
               237  82  44  75 | 255 113  39  85
                86  34  44  65 | 130  78  39  54
                10   8  44  71 |  69  56  39  42
                72  44  44  55 |  88  54  39  50
```

Figure 6.6: DCT encoding, stage three - scale the values and quantise them.

```
The header:     8   8   4 352.00 -64.00
The body:     167  68   0   0 | 247  86   0   0
               68  57   0   0 |  75  34   0   0
                0   0   0   0 |   0   0   0   0
                0   0   0   0 |   0   0   0   0
              ---------------------------------
              237  82   0   0 | 255 113   0   0
               86  34   0   0 | 130  78   0   0
                0   0   0   0 |   0   0   0   0
                0   0   0   0 |   0   0   0   0
```

Figure 6.7: DCT encoding, stage four - discard high frequency information.

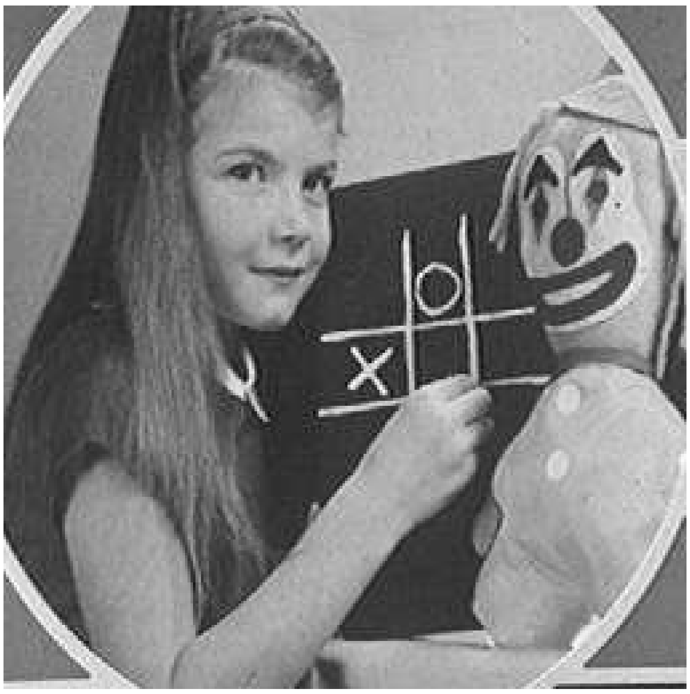
```
The header:  8   8   4 352.00 -64.00
            14 167 68 0 57 247 237 86 75 82 34 255 113 130 78
The body:    1   1   2   2   1   3   1   4   5   3   1   5   3   3
             1   6   1   7   1   8   2   3   1   9   1   7   1   3
             1  10   3   3   1  10  12   3   1  11   5   3   1  12
             1  13   4   3   1  14  12   3
```

Figure 6.8: DCT encoding, stage five - run length encode the data, diagonally.

outside of the top left quadrant of each patch are set to zero. Note that, if we had discarded all but the zero frequency component, the reconstructed image would simply be the original reduced to a resolution equal to the patch size. Finally, the 'image' is run length encoded, with the runs going diagonally from upper right to lower left, starting at the top lefthand corner.

At this stage we are using 76 bytes to encode the data, and by Huffman coding we might expect to reduce this to about 15 bytes – a compression ratio of 4.4:1. This is slightly better than was acheived by using run length encoding and Huffman encoding alone, but not worth the loss in quality of the image – in fact the encoded data that we have left produces a reconstruction which is unrecognisable as the original.

In the case of a real image the runs of zeroes would be longer, even if less high frequency information was discarded, and larger patch sizes can be



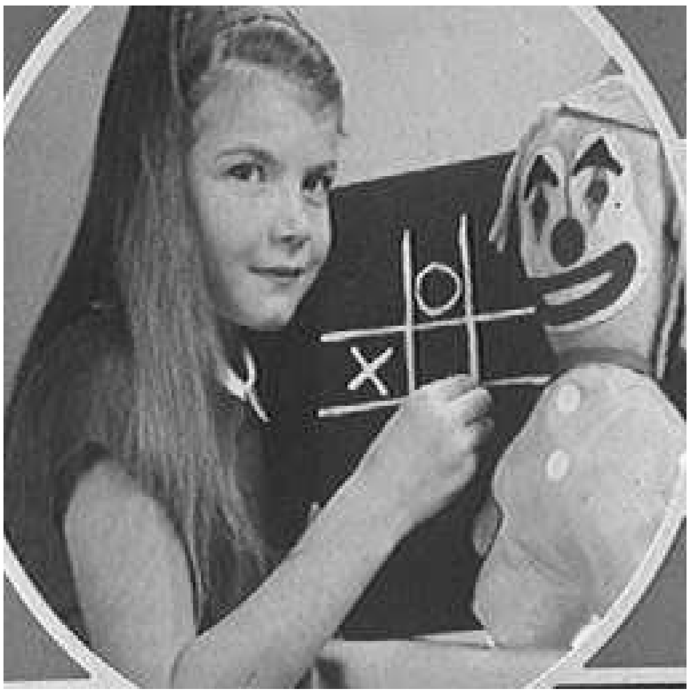


Figure 6.9: BBC testcard image – original [top], reconstruction from DCT based compression [bottom].

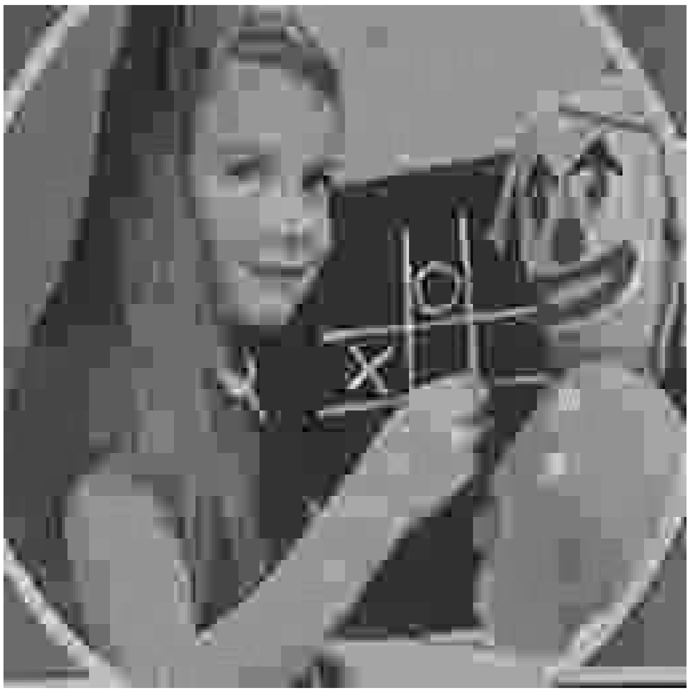


Figure 6.10: BBC testcard image reconstructed from a DCT based compression, with a compression ratio of 100:1.

used. This results in a good quality reconstruction from even higher compression ratios. Figure 6.9 shows the single band version of the BBC testcard image, both in its original form and as reconstructed from a DCT based compression. The compression factor achieved is 6.2:1, and the reduction in image quality is unnoticeable. Forcing a higher compression factor, by using larger patches and discarding more frequency components, would result in quite obvious imperfections appearing. Figure 6.10 shows the, rather extreme, case of forcing a compression factor of 100:1.

Again, with specialised hardware, DCT encoding and decoding can be achieved at video rates.

Another way of increasing the local homogeneity of an image is to segment it. A well segmented image should contain fairly large homogeneous regions which will run length encode efficiently. If the segmentation does not constitute a sufficiently good representation of the image, then a second image, composed of correction terms, can be encoded separately using one of the previously discussed methods. Most of this correction image should be zero, or close to zero, so it will encode reasonably efficiently. The two images together will need to occupy less space than a straight compression of the original image in order for the approach to be worthwhile.

Since segmentation does produce such large and, very, homogeneous regions, we can use an encoding technique which is even more efficient, in this sort of case, than run length encoding. This is boundary following, or contour, encoding [Gonzalez & Wintz, 1987]. This makes use of the fact that, for sufficiently large regions, with smooth outlines, the number of pixels around the perimeter of the region will be considerably less than the number of pixels in the region, and the shape of the region will be completely described by the shape of its outline.

The major complication with boundary following encoding is figuring out what order the boundaries should be encoded in. Consider the example image in figure 6.1 (it is sufficiently homogeneous for us to take it as already having been segmented). We can either imagine this to be a jigsaw, with the regions slotting into one another, or a montage, with the regions lying one on top of another. If we take the former view the order in which the regions are dealt with is immaterial, but we will end up encoding the outlines of the four interior regions twice each – once as the shape of the boundaries of those regions, and once as the shape of the 'holes' into which they slot. This redundancy of information will result in a reduction in the efficiency of the compression. Also, since we do not have any way of predicting how many holes there may be in a region we will either have to encode the number of boundaries that each region possesses, or insert markers between the sets of boundaries which describe each region. Both of these approaches will, again, reduce the efficiency of the compression. However, if we take the montage view we have only to descibe the outer boundary of each region. Then, during the reconstruction of the image the 'foreground' regions will just be 'drawn' on top of the 'background' ones. There is now only one boundary description per region so there will be no need to place markers between them, since the end of each description can be determined by the fact that it must return to the starting point. With the montage view it is necessary, however, to ensure that the regions are encoded going from the background to the foreground, otherwise, during reconstruction, foreground regions may be overdrawn by background ones.

Figure 6.11 shows the result of doing a simple boundary following encoding of the image in figure 6.1. In this example the header is the same as for run length encoding. The first byte of the body indicates that the background intensity is to be set to that specified by the first entry in the pixel value look-up table. Note that, there is no point in specifying the shape of the background region, since it is the same shape as the image. The next two bytes give the coordinates of the top lefthand vertex of the first pixel in the first region (the coordinates are relative to the top lefthand corner of the image). There then follow a series of pairs of numbers which define the boundary of the region – the first of each specifies how long the boundary segment is, the second its direction ($0 \Rightarrow$ right, $1 \Rightarrow$ down, $2 \Rightarrow$ left, $3 \Rightarrow$ up). These two values can be stored in a single byte, six bits for the length and two for the direction. After six segments we have returned to the starting point, and the next byte indicates what intensity to fill the region with (by reference to the look-up table). The process is then repeated for the other three regions.

```
The header:  8  8  4 16  8 64 32
The body:    1
             1  1  4 0  2 1  2 2  2 1  2 2  4 3
             2
             6  1  1 0  1 1  1 2  1 3
             3
             3  3  4 0  3 1  4 2  3 3
             4
             1  6  1 0  1 1  1 2  1 3
             3
```

Figure 6.11: Boundary following encoding of the image in figure 6.1.

The total encoded size is 38 bytes, giving a compression ratio of 1.7:1. This is slightly better than that achieved by simple run length encoding. However, although the regions in this image are large compared with the image size, they are, in absolute terms, very small. If the image was scaled up by a factor (α) in each direction, the original image size would go up as α^2 (ignoring the header), the run length encoded verion would go up, roughly, as α (there would be about α times as many runs, each α times longer than before), but the size of the boundary following encoded version would remain the same (there would be the same number of regions, each with the same number of boundary segments, the segments would just be α times longer). This means that the compression ratio for run length encoding will go up as α as the image size increases, but for boundary following encoding it will go up as α^2. Doubling the size of the image would give compression ratios of about 2.7:1 for run length encoding, and 6.9:1 for boundary following. Quadrupling the size gives ratios of about 5.4:1 and 13.8:1 respectively, a significant difference. In fact the compression ratio for the boundary following is getting to the sort of values associated with DCT encoding, and this is before doing any Huffman coding of the segment length/direction pairs.

Unfortunately the author did not have time to code up any implementations of boundary following encoding techniques, so compression factors for real images are not available.

An example of encoding using segmentation and boundary following is Moran & Morris's 'region and texture' (RAT) method [1989]. In this technique the image is segmented using a region growing algorithm, and the segmented (region) image is then encoded using a boundary following technique due to Biggar & Constantinides [1987]. The difference between the segmented image and the original (in Moran & Morris's terms, the texture) is DCT encoded.

The RAT method is intended for use in the field of HDTV, and it is interesting to see how Moran & Morris extend their algorithm to work with multi-spectral images. Rather than encoding each of the three (red, green, and blue) bands separately, they form the total intensity image ($\frac{1}{3}$[red+green+blue]) and use this to produce a single pair of region and texture images to be encoded. The only colour information that is recorded is the average colour of each of the regions. The rationale behind this is that, since the application only requires good visual quality reconstruction, we can make use of the fact that the human eye/brain pays more attention to intensity variations, when analysing an image, than it does to colour variations. This means that it is not absolutely necessary for the region image to describe every subtle variation in colour, nor is it necessary for colour error terms to be calculated.

This approach means that segmentation based compression of multi-spectral images has a built in headstart on any other technique. That is, no matter how many bands there are in the original image, we only ever need to encode two (the region and error images). So, whereas with non-segmentation based compression algorithms the size of the compressed data goes up roughly with the number of spectral bands, and the compression ratio remains constant, with a segmentation based approach the size of the compressed data remains constant, and the compression ratio goes up with the number of bands. With a seven band LANDSAT data set we would be looking at a compression ratio of about 3.5:1 even if the region and error images received no further compression.

The disadvantage with segmentation based techniques is that they only provide slow encoding – the region growing technique used by Moran & Morris takes the best part of an afternoon to segment a single television type image (in software). Whether or not a very parallel implementation, in hardware, can achieve segmentation at video rates remains to be seen.

However, reconstruction would be quick and potentially very high compression is possible. This makes segmentation based encoding suitable for some applications involving the archiving of imagery, particularly remote sensing imagery where large numbers of spectral bands are common. It will not be suitable for those archiving applications where the reconstructed data is to be analysed quantitatively, though, because of the large amount of information lost in the segmentation process.

6.3 Use of the new segmentation technique :

The segmentation technique described in this work offers two possible advantages over that used by Moran & Morris. Firstly, it is definitely quicker – taking something like ten minutes to segment the sort of images that Moran & Morris are dealing with. Secondly, it could enable higher compression ratios to be attained. There are two reasons for this. In the first place, the new segmentation technique is class based rather than region based. This means that we only need to record colour information for each of the classes in the image (typically ten or twenty in number), rather than for each of the regions (over a thousand in Moran & Morris's segmentations). In the second place, the removal of the lighting effects from the image goes a long way towards guaranteeing a segmentation composed of a small number of large regions, which is what is needed for boundary following encoding to work at its most efficient.

Consider figure 6.12. In this idealised case (regular square regions) we can see that the total number of boundary segments in the image is proportional to the number of regions, and the total length of all the boundaries is proportional to the square root of the number of regions. Now, the size of an image compressed using a boundary following technique is roughly proportional to the total number of boundary segments in the image. So, if the number of boundary segments required to define the shape of a region is independent of the size of the region (as it is in figure 6.12) we would expect the compression ratio for this approach to be inversely proportional to the number of regions found. This is not likely to be the case though, since the larger a region is the more scope there is for it to have a highly convoluted boundary. In the worst case the total number of boundary elements will equal the total length of all of the boundarys, and we have just seen (from the idealised example) that we can expect the total length of all the boundaries to be roughly proportional to the square root of the number of regions. This means that, if the new segmentation technique can reduce the number of regions in the segmentation by a factor of ten or twenty, say, we could increase the compression ratio by a factor of about three or four.

The new segmentation technique cannot just be dropped straight into the RAT method though. Since the intensity information was removed from the image prior to segmentation, the segmented image will contain only information on the colours in the image. Therefore, the texture image needs to be replaced by an intensity image. This is unlikely to DCT encode quite as

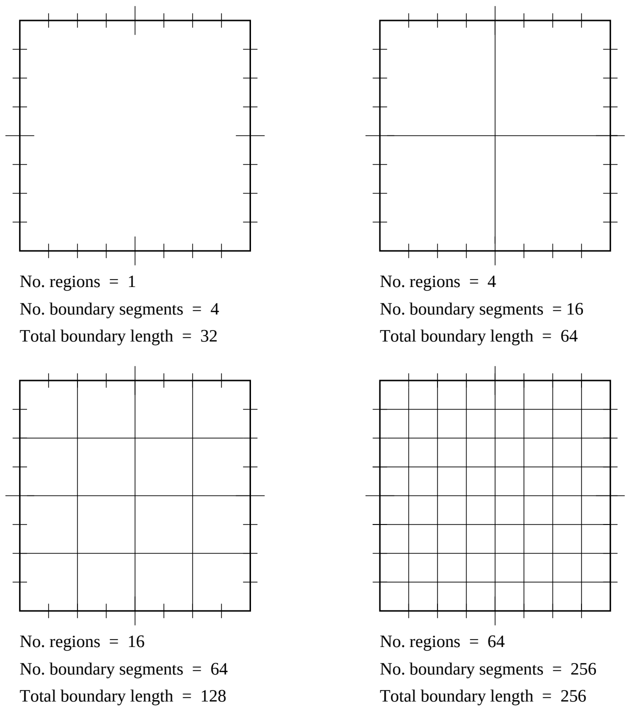

No. regions = 1
No. boundary segments = 4
Total boundary length = 32
No. regions = 4
No. boundary segments = 16
Total boundary length = 64
No. regions = 16
No. boundary segments = 64
Total boundary length = 128
No. regions = 64
No. boundary segments = 256
Total boundary length = 256

Figure 6.12: Variation of boundary quantities with the number of regions.

efficiently as Moran & Morris's error image, but, hopefully, it will not use up all of the gains made on the region coding side.

6.4 Results :

Figure 6.13 shows the full, three band, version of the BBC testcard image. Figure 6.14 is the result of segmenting it using the new technique. Six classes have been found, and the image is divided into only about fifty regions. The regions have each been filled with the average colour of the class to which they belong. Figure 6.15 is the intensity component of figure 6.13, which has been compressed and uncompressed using a DCT based algorithm. When the intensity image is applied to the, coloured, segmented image in HIS space (see chapter 4), figure 6.16 is the result. There is a large amount of bleaching in the faces (the clown's has lost all colour information) – this is the reverse of the deep shadow problem seen in chapter 5. Other than that the result is reasonably acceptable. It may be possible to reduce the bleaching problem by deliberately splitting any 'colourless' cluster into pale versions of the six primary and secondary colours (red, green, blue, cyan, magenta, and yellow), but that is only going to be worth doing if the problem exhibits itself in more than this one image.

Figure 6.17 shows the full colour version of the common image processing test image 'Lenna'. Figure 6.18 is the reconstruction from the segmentation based compression. This is actually quite good.

These reconstructions, even that of 'Lenna', whilst of acceptable visual quality, are far from good enough for use in the field of HDTV. This is because we have discarded a lot of information in the segmentation process. It is of the sort of quality which would be alright for use with video telephones, or some archiving applications, though.

Unfortunately, the lack of an implementation of a boundary following encoding algorithm, and the unavailability of a detailed account of Moran & Morris's method, has meant that it has not been possible to make any quantitative comparisons between the two segmentation approaches.

6.5 Future possibilities :

Moran & Morris's idea of splitting an image into a combination of a set of coloured regions and a greylevel image containing nothing but texture information opens up an avenue of investigation which could lead to even higher compression ratios.

For a long time it has been possible to segment an image on the basis of the textures that it conatins, usually by splitting the image into small patches and comparing the Haralick texture measures obtained from each of them

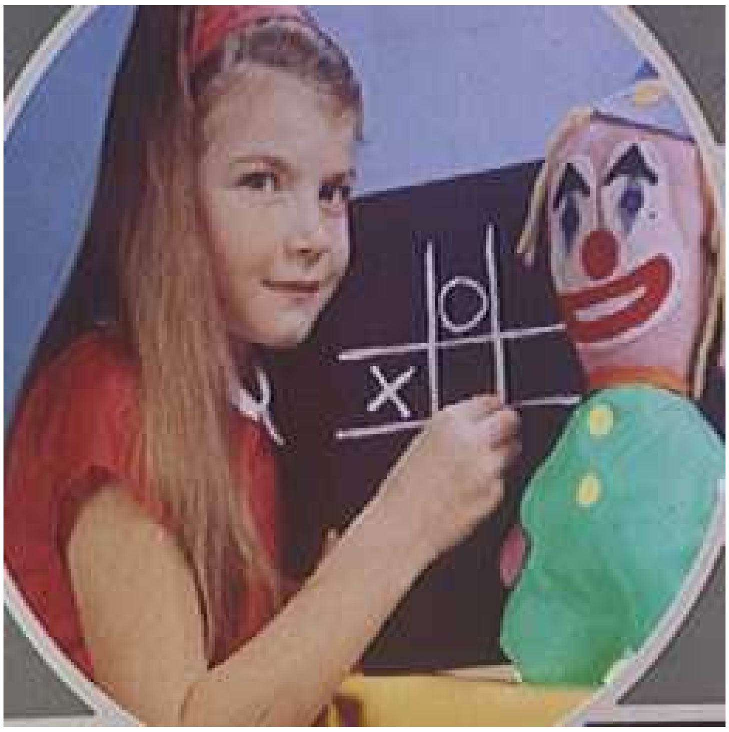


Figure 6.13: BBC testcard, original image.

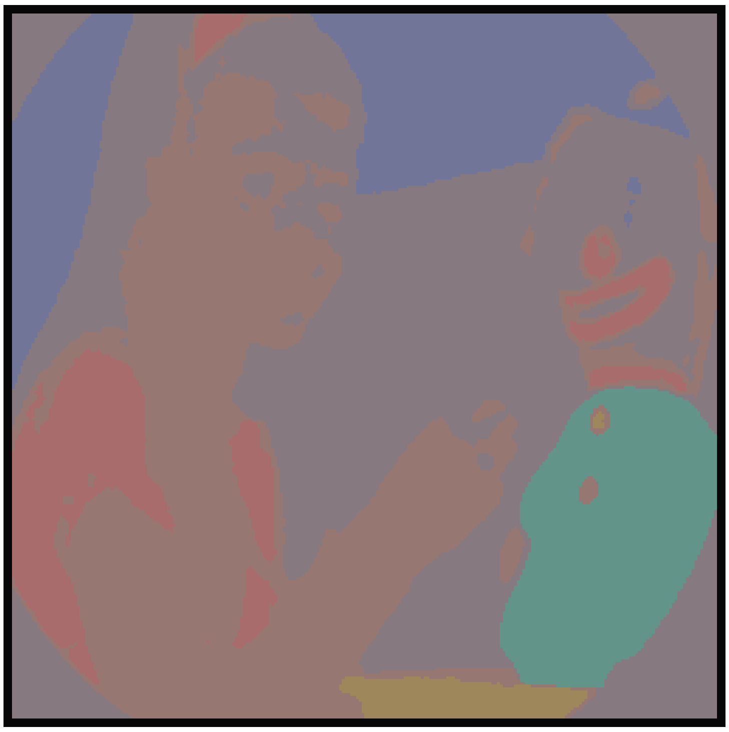


Figure 6.14: BBC testcard, segmented image.

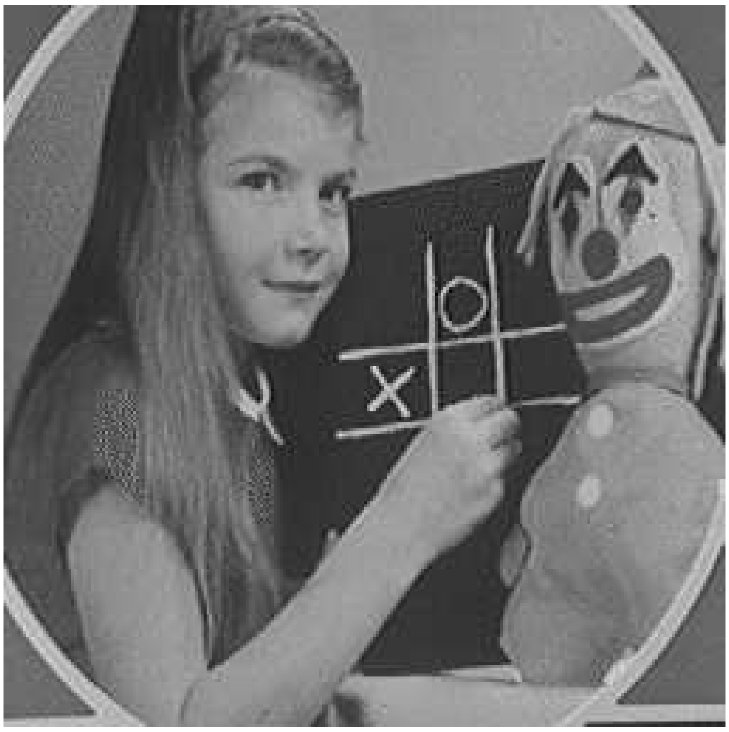


Figure 6.15: BBC testcard, intensity image.

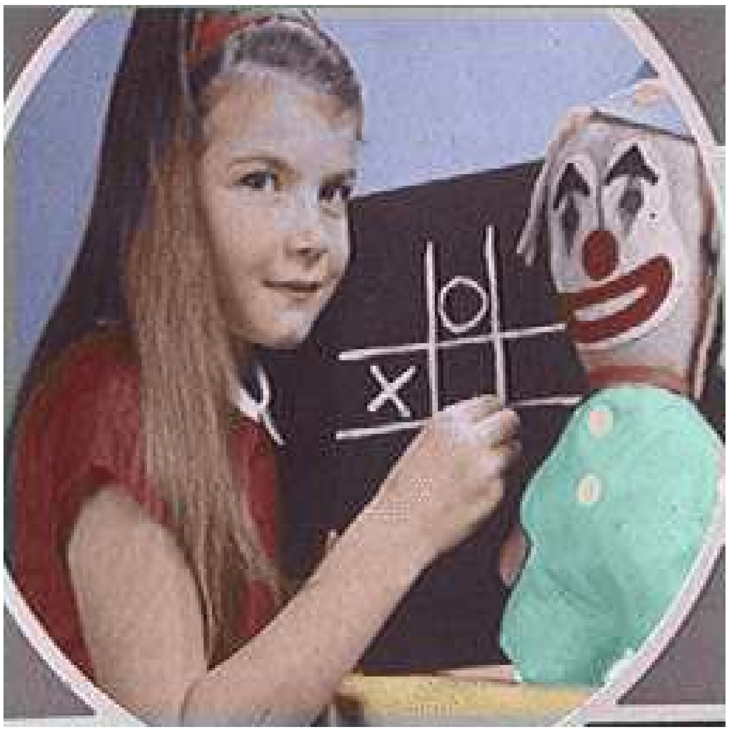


Figure 6.16: BBC testcard, reconstructed image.

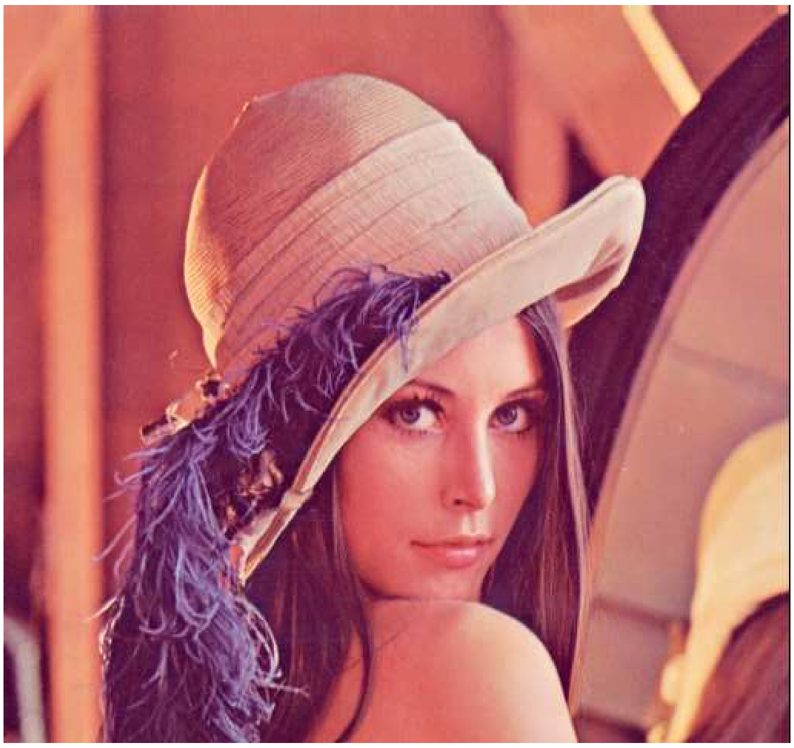


Figure 6.17: Lenna, original image.

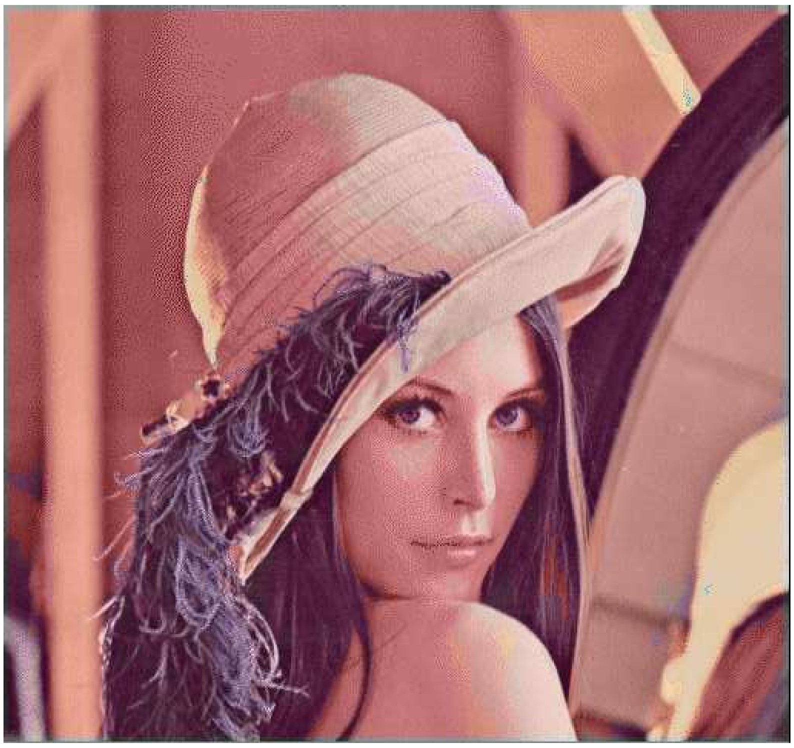


Figure 6.18: Lenna, reconstructed image.

[Haralick, et al., 1973; and Carbon & Ebel, 1988]. If the texture image, that we are dealing with in the original RAT method, was to be segmented, it too could be encoded using the boundary following method, rather than DCT.

Unfortunately, the Haralick texture measures are non-invertible – that is, although we can calculate them for each texture region, and store them in the compressed image, they cannot be used to reconstruct the texture image when it comes to doing the uncompression. However, Rabe [1991] has been working on a representation of texture which is based on the Ising model, from statistical mechanics. This not only allows the texture in an image patch to be parameterised, and hence segmented, but the parameters can be used to regenerate a swatch of texture which looks the same as the original.

Sadly, commitments to other work have meant that it has not been possible for this potentially profitable area to be explored.

CHAPTER 7
Comparison with existing techniques

The performance of the proposed cluster analysis technique is compared with that of two well established techniques. The comparison is made on the basis of their relative speeds, and the quality of their results. It is shown that, whilst looking promising, the proposed technique still requires some development.

In order that they could be compared with the proposed technique, Forgy's method and MacQueen's k-means were implemented in `FORTRAN 77` (see appendix B). The implementation of the proposed method used for the comparisons was a slimmed down version of that used to produce the segmentations in the preceeding chapters – it only performed cluster location, with none of the image processing specific steps. Neither of the hierarchical techniques, discussed in chapter 1, were included in these comparisons because they are unsuitable for use with large numbers of data points (see §1.4), and the proposed technique is unsuitable for use with small numbers of data points (because of the problem of noise on the one dimensional histograms, see §§ 1.1 & 2.2).

The stopping criterion used for Forgy's method was that the mean squared error of the classification should have ceased to decrease. This is not necessarily the best choice, but it does remove the possibility of rogue data causing the algorithm to iterate indefinitely.

For the data sets used in these comparisons, a set of cluster attributes

– the cluster position, the cluster width, and the probability of a data point belonging to a particular cluster – were selected at random (with some constraints, so as to prevent too wide a range of positions, or unreasonably large values for the cluster widths). The data points themselves were then selected at random so that, en mass, they produced Gaussian density distributions of the required position, standard deviation, and height. The details of the various distributions used are given in tables 7.1 to 7.9.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.16	216.07	11.75
Ω_2	0.28	212.48	10.36
Ω_3	0.26	109.89	5.32
Ω_4	0.31	230.20	11.28

Table 7.1: Details of 1 dimensional; 4 class; 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.13	(216.07, 133.46)	(11.75,11.02)
Ω_2	0.29	(118.88, 53.01)	(7.35, 9.17)
Ω_3	0.33	(219.81, 131.19)	(7.13, 9.07)
Ω_4	0.25	(36.28, 70.15)	(4.33, 5.69)

Table 7.2: Details of 2 dimensional; 4 class; 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	1.00	(216.07, 133.46, 194.10, 109.89)	(11.75, 11.02, 7.67, 5.32)

Table 7.3: Details of 4 dimensional; 1 class; 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.34	(216.07, 133.46, 194.10, 109.89)	(11.75, 11.02, 7.67, 5.32)
Ω_2	0.66	(230.20, 103.74, 157.99, 36.28)	(11.27, 8.11, 8.80, 4.33)

Table 7.4: Details of 4 dimensional; 2 class; 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.17	(216.07, 133.46, 194.10, 109.89)	(11.75, 11.02, 7.67, 5.32)
Ω_2	0.33	(230.20, 103.74, 157.99, 36.28)	(11.27, 8.11, 8.80, 4.33)
Ω_3	0.21	(63.46, 20.42, 200.64, 36.27)	(8.88, 6.14, 9.45, 7.78)
Ω_4	0.30	(200.82, 52.24, 89.47, 83.19)	(6.88, 6.12, 7.82, 11.63)

Table 7.5: Details of 4 dimensional; 4 class; 8000, 16000, 32000, 64000, 128000, & 256000 sample data sets.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.07	(216.07, 133.46, 194.10, 109.89)	(11.75, 11.02, 7.67, 5.3)
Ω_2	0.14	(230.20, 103.74, 157.99, 36.28)	(11.28, 8.11, 8.80, 4.33)
Ω_3	0.09	(63.46, 20.42, 200.64, 36.27)	(8.88, 6.14, 9.45, 7.78)
Ω_4	0.13	(200.82, 52.24, 89.47, 83.19)	(6.88, 6.12, 7.82, 11.63)
Ω_5	0.10	(123.29, 154.97, 112.03, 218.04)	(6.57, 10.62, 6.07, 11.14)
Ω_6	0.15	(137.47, 158.77, 144.32, 154.72)	(11.46, 6.08, 11.36, 10.03)
Ω_7	0.18	(148.82, 155.24, 119.22, 206.08)	(8.68, 11.22, 11.99, 7.46)
Ω_8	0.13	(16.74, 24.01, 87.59, 40.76)	(7.39, 11.50, 11.63, 8.87)

Table 7.6: Details of 4 dimensional; 8 class; 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.04	(216.07, 133.46, 194.10, 109.89)	(11.75, 11.02, 7.67 5.32)
Ω_2	0.07	(230.20, 103.74, 157.99, 36.28)	(11.28, 8.11, 8.80, 4.33)
Ω_3	0.05	(63.46, 20.42, 200.64, 36.27)	(8.88, 6.14, 9.45, 7.78)
Ω_4	0.06	(200.82, 52.24, 89.47, 83.19)	(6.88, 6.12, 7.82, 11.63)
Ω_5	0.05	(123.29, 154.97, 112.03, 218.04)	(6.57, 10.62, 6.07, 11.14)
Ω_6	0.07	(137.47, 158.77, 144.32, 154.72)	(11.46, 6.08, 11.36, 10.03)
Ω_7	0.09	(148.82, 155.24, 119.22, 206.08)	(8.68, 11.22, 11.99, 7.46)
Ω_8	0.06	(16.75, 24.01, 87.59, 40.76)	(7.39, 11.51, 11.63, 8.87)
Ω_9	0.05	(117.61, 228.48, 44.93, 53.33)	(8.03, 5.23, 8.91, 11.96)
Ω_{10}	0.07	(49.02, 92.06, 103.92, 45.83)	(9.48, 7.19, 10.41, 5.76)
Ω_{11}	0.07	(135.68, 89.93, 82.97, 31.45)	(7.21, 11.43, 5.65, 10.30)
Ω_{12}	0.08	(107.33, 215.22, 120.36, 54.61)	(6.49, 10.12, 7.31, 4.12)
Ω_{13}	0.05	(129.72, 227.54, 174.62, 110.28)	(5.75, 9.50, 5.55, 7.58)
Ω_{14}	0.06	(53.85, 99.59, 116.98, 55.13)	(9.85, 7.31, 4.02, 9.21)
Ω_{15}	0.05	(207.40, 110.44, 168.86, 31.40)	(7.11, 11.20, 4.10, 5.64)
Ω_{16}	0.08	(25.50, 235.90, 221.79, 175.91)	(5.38, 4.90, 6.96, 9.65)

Table 7.7: Details of 4 dimensional, 16 class, 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.18	(216.07, 133.46, 194.10, 109.89, 160.68, 219.81, 131.19, 150.50)	(11.75, 11.02, 7.67, 5.32, 11.64, 7.13, 9.07, 4.72)
Ω_2	0.16	(70.15, 152.54, 75.85, 168.46, 121.70, 200.82, 52.24, 89.47)	(5.69, 4.16, 10.59, 4.72, 8.35, 6.88, 6.12, 7.82)
Ω_3	0.24	(229.66, 123.29, 154.97, 112.03, 218.04, 167.59, 224.99, 74.23)	(6.13, 6.57, 10.62, 6.08, 11.14, 8.34, 9.10, 8.58)
Ω_4	0.42	(154.72, 227.32, 147.17, 218.30, 239.76, 112.88, 16.75, 24.01)	(10.03, 8.74, 8.97, 7.69, 10.79, 8.26, 7.39, 11.51)

Table 7.8: Details of 8 dimensional, 4 class, 64000 sample data set.

	$p(\Omega)$	λ_2 (mean)	λ_3 (standard deviation)
Ω_1	0.17	(216.07, 133.46, 194.10, 109.89, 160.68, 219.81, 131.19, 150.50, 25.29, 63.46, 20.42, 200.64, 36.27, 137.75, 96.61, 75.29)	(11.75, 11.02, 7.67, 5.32, 11.65, 7.13, 9.07, 4.72, 5.93, 8.88, 6.14, 9.44, 7.78, 10.60, 5.29, 6.62)
Ω_2	0.28	(83.19, 75.75, 87.97, 201.25, 74.04, 215.78, 137.47, 158.77, 144.32, 154.72, 227.32, 147.17, 218.30, 239.76, 112.88, 16.75)	(11.63, 7.83, 8.96, 7.43, 11.22, 9.41, 11.46, 6.08, 11.36, 10.03, 8.74, 8.97, 7.69, 10.79, 8.26, 7.39)
Ω_3	0.15	(226.15, 229.58, 152.28, 117.61, 228.48, 44.93, 53.33, 167.07, 169.34, 105.21, 195.62, 65.22, 135.68, 89.93, 82.97, 31.45)	(6.56, 4.88, 6.43, 8.03, 5.23, 8.91, 11.96, 5.18, 6.72, 7.14, 5.07, 8.90, 7.21, 11.43, 5.65, 10.30)
Ω_4	0.39	(107.33, 215.22, 120.36, 54.61, 75.11, 65.08, 170.02, 59.42, 116.14, 53.85, 99.59, 116.98, 55.13, 75.01, 103.14, 217.72)	(6.49, 10.11, 7.31, 4.12, 8.06, 11.55, 9.67, 7.37, 8.05, 9.85, 7.31, 4.02, 9.21, 10.84, 7.37, 9.46)

Table 7.9: Details of 16 dimensional, 4 class, 64000 sample data set.

7.1 Comparison with respect to speed :

All of the timings in the following graphs are the total amount of CPU time taken for the whole of the execution of the programs – including reading in the data, and writing out the results.

In these first comparisons, Forgy's method and MacQueen's k-means were both given the correct number of clusters to look for, the proposed method was left to find as many as it liked.

Figure 7.1 shows the variation in CPU time taken to locate the clusters in data sets which all contain the same number of clusters ($k = 4$), and which all occupy the same number of dimensions ($N = 4$), but which each contain different numbers of data points (M = 16000,32000,64000,128000, & 256000).

It would seem that all of the techniques are linearly dependent on M, with a zero offset (zero data points would take zero time). This is as would be expected, since all the data points are processed independently of each other. Forgy's method and MacQueen's k-means are both roughly twice as fast as the proposed method.

Figure 7.2 shows the variation in CPU time taken to locate the clusters in data sets which all contain the same number of data points ($M = 64000$), and which all occupy the same number of dimensions ($N = 4$), but which each contain different numbers of clusters ($k =$ 1,2,4,8, & 16).

It would appear that the proposed method and MacQueen's k-means are both roughly linearly dependent on k – although the proposed method would require quite large error bars. Forgy's method, on the other hand, has become somewhat erratic. This is likely to be the result of particular juxtapositionings of clusters creating conditions which require either an abnormally large, or an abnormally small, number of iterations to be performed before they can be seperated.

MacQueen's k-means is between two and five times faster than the proposed method, in these cases, and Forgy's method is between two and three times faster.

Figure 7.3 shows the variation in CPU time taken to locate the clusters in data sets which all contain the same number of data points ($M = 64000$), and which all contain the same number of clusters ($k = 4$), but which each occupy different numbers of dimensions ($N =$ 1,2,4,8, & 16).

Forgy's method and MacQueen's k-means both show a very linear dependence on N. This is as would be expected since, for these algorithms, the number of dimensions just determines the number of times particular loops are executed. The proposed method, in contrast, shows a very high, non-linear,

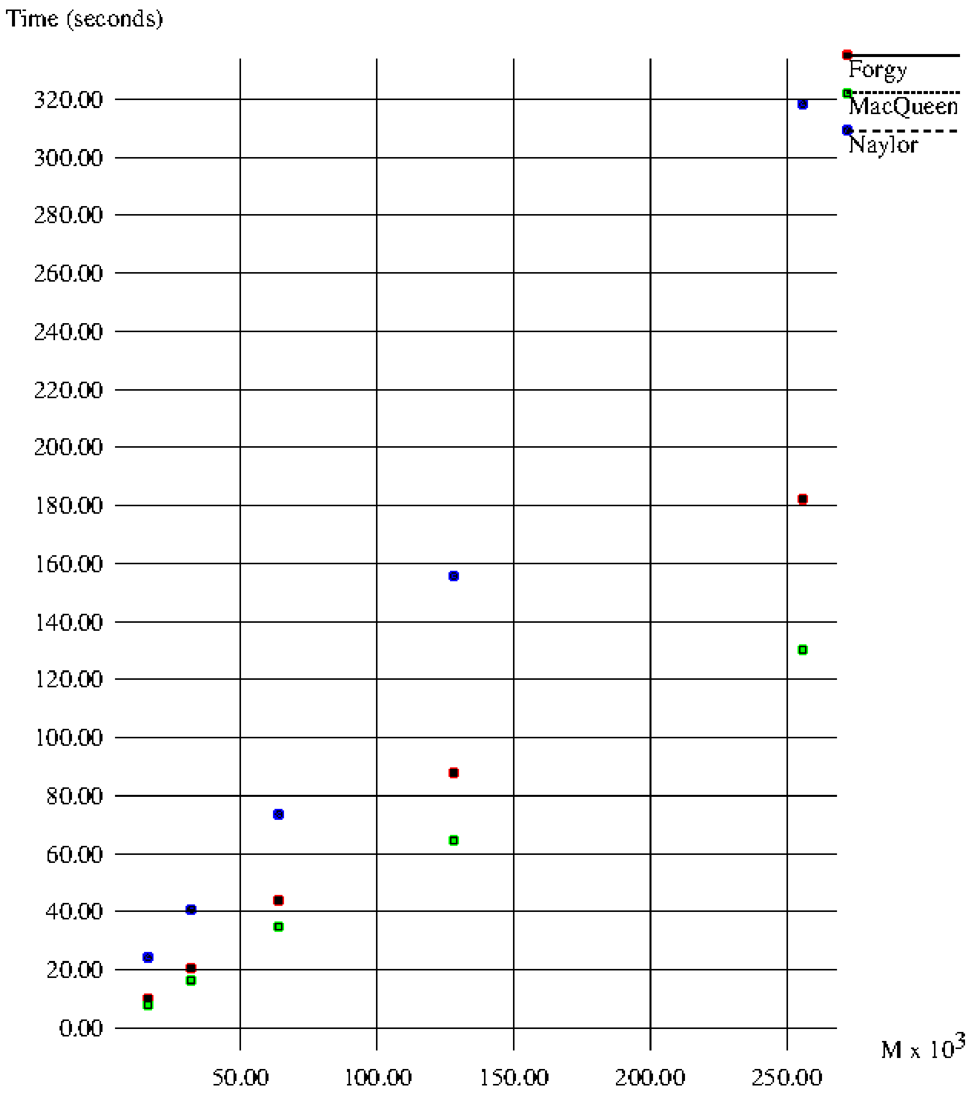


Figure 7.1: Variation in CPU time used with respect to data set size.

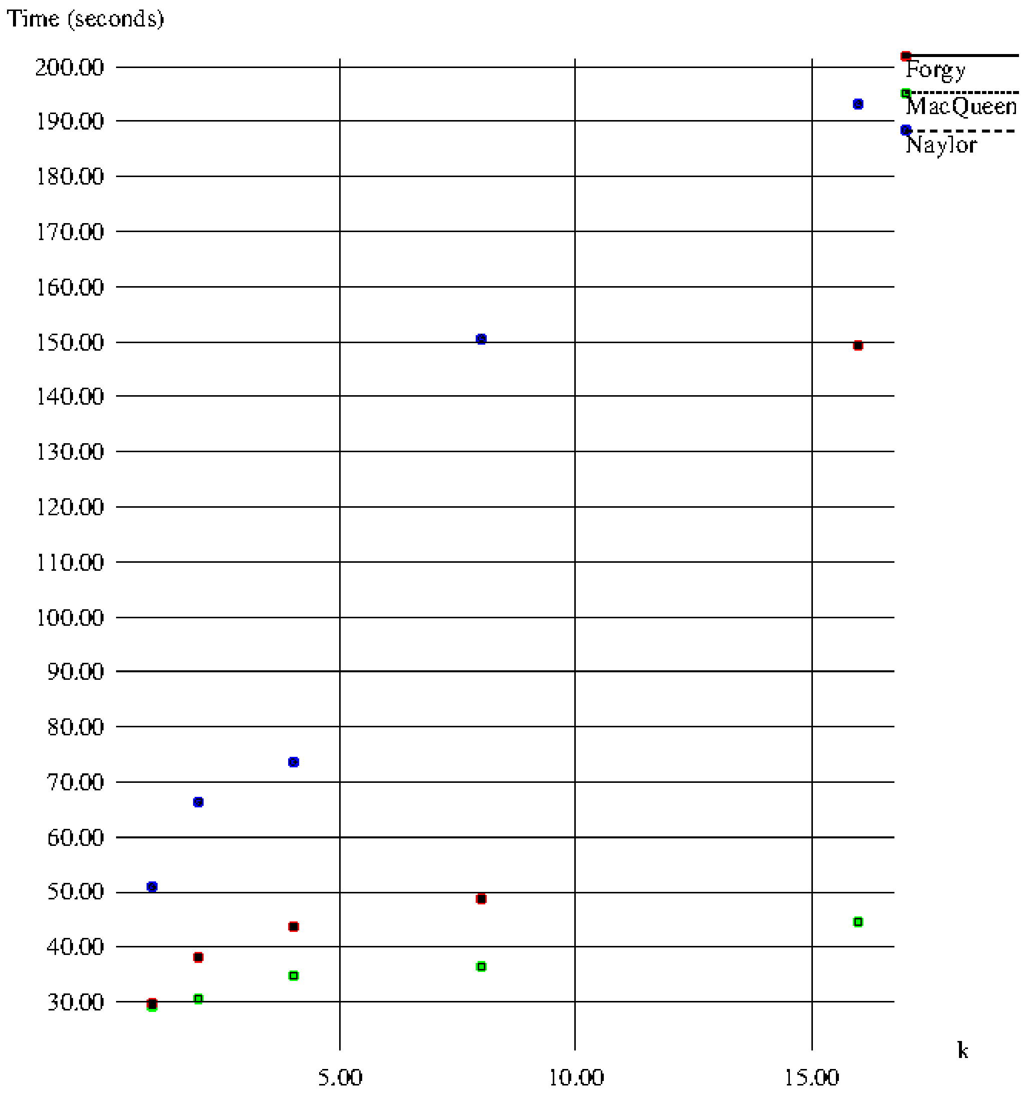


Figure 7.2: Variation in CPU time used with respect to the number of classes.

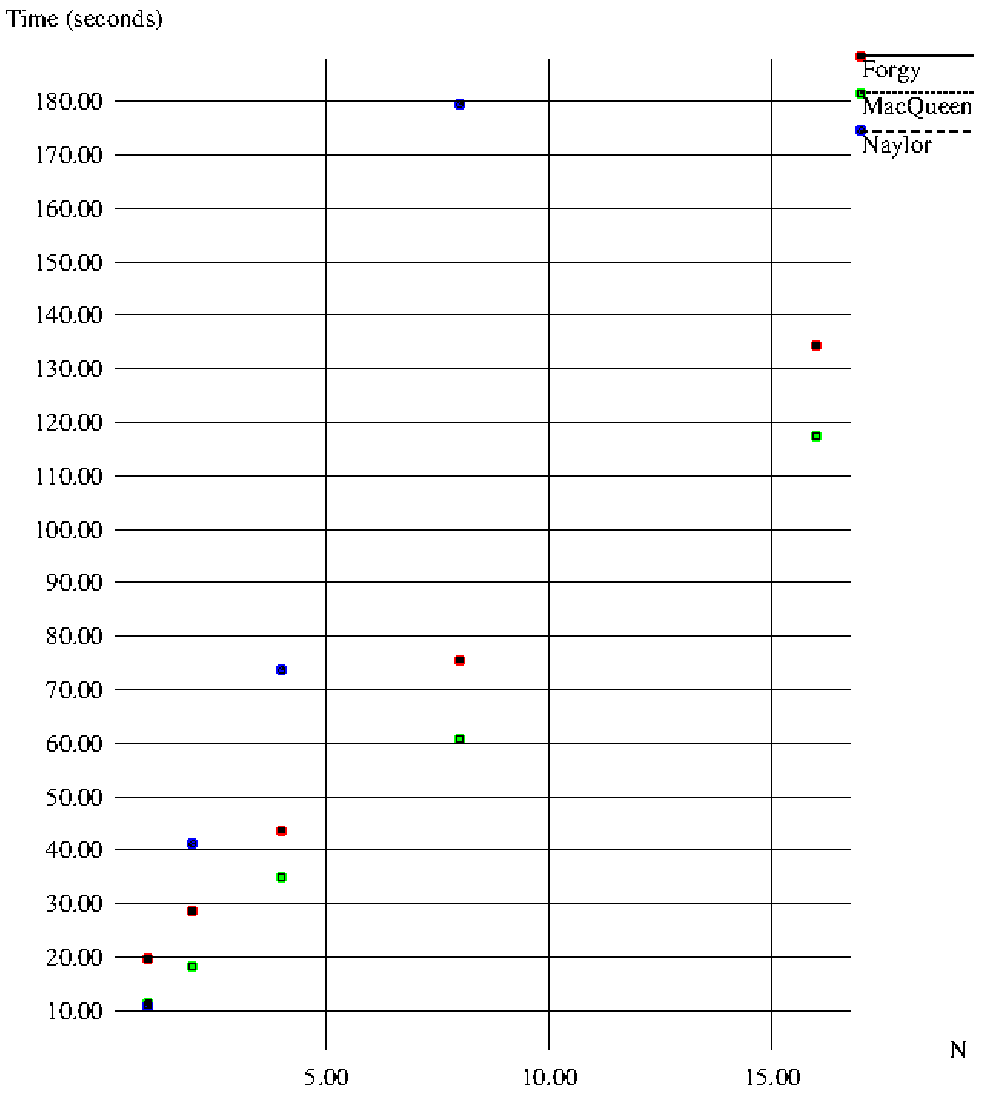


Figure 7.3: Variation in CPU time used with respect to the number of dimensions.

dependence on N. The last point (for 16 dimensions) has not been plotted because the program had still not finished after more than 5400 seconds (an hour and a half) of CPU time.

A possible explanation for this great sensitivity to the number of dimensions is that the number of candidate clusters which have to undergo the first test in the peak linkage process is going to be of the order of k^N. This is because the number of peaks found in each of the one dimensional histograms (n_s) is going to be of order k, and every possible combination of peaks (n_s^N of them) have to be tested. However, if this was all that was involved, the proposed method should also show a strong, non-linear, dependence on k. This is not apparent on figure 7.2, unless the fifth point ($k = 16$) is spuriously low, for some reason.

Given the proposed method's high dependence on N, it is only possible to say that both Forgy's method and MacQueen's k-means are both, for the most part, faster.

The conclusion so far, then, is that MacQueen's k-means is the fastest of the three methods under consideration, and the proposed method is the slowest.

However, what about situations where it is impractical, or undesirable, to have to supply the cluster analysis algorithm with the number of clusters to look for ? Unless it is possible to provide an automated method for estimating k, Forgy's method and MacQueen's k-means will need to be applied to the data for all values of k' from $k' = 1$ upwards. At each iteration the quality of the results would need to be measured, and this value should be higher for $k' = k$ than it is for $k' = k - 1$ or $k' = k + 1$. Thus, a total of $k + 1$ iterations will be required to find k and do the cluster analysis.

To simulate this situation an extra loop was added to the programs implementing Forgy's method and MacQueen's k-means, so that the cluster location part was executed (with the appropriate value of k) the required number of times. No attempt, however, was made to actually provide a test of the quality of the analysis.

Figure 7.4 shows the new variation in CPU time taken to locate the clusters, with respect to data set size.

MacQueen's k-means retains a linear dependence on M. Forgy's method, on the other hand, now shows a non-linear dependence. This is because, when the wrong value of k is supplied, there is no way of knowing how many iterations (within Forgy's method) will be required for what is a nonsensical description of the data to result in a minimum in the mean squared error of classification.

The proposed method is now faster than Forgy's method, for $M \geq 25000$ or so, but MacQueen's k-means is still about one and a half times faster than the proposed method.

Figure 7.5 shows the new variation in CPU time taken to locate the clusters, with respect to the number of clusters in the data set.

With the iteration from $k' = 1$ to $k' = k + 1$ we would expect both Forgy's method and MacQueen's k-means to exhibit a non-linear dependence on k, and this is indeed the case. The severity of the effect on Forgy's method would, perhaps, be surprising if we had not already seen the high non-linearity of the previous graph.

The proposed method is faster than Forgy's method, for $k \geq 3$ or so, and, this time, MacQueen's k-means is only faster than the proposed method for $k \leq 14$, or thereabouts.

Figure 7.6 shows the new variation in CPU time taken to locate the clusters, with respect to the number of dimensions which the data set occupies.

Forgy's method amd MacQueen's both retain their very linear dependence on N. This might not have been the case for Forgy's method if the two fixed values ($M = 64000$ & $k = 4$) had not been within the range where it is still behaving fairly linearly in figures 7.4 & 7.5.

We now have a series of changes in the ordering of the algorithms as regards speed. In the one dimensional case the proposed method is the fastest, followed by MacQueen's k-means. Between two dimensions and about five dimensions MacQueen's k-means is the fastest, followed by the proposed method. Finally, above about five dimensions MacQueen's k-means is still the fastest, but now Forgy's method is in second place.

What conclusions can we draw, then, about the situation in which the proposed method is designed to work ?

Well, provided that there are less than about fourteen clusters in the data, the fastest method will always be MacQueen's k-means. Second and third place are determined as follows : for less than about three clusters, more than about 25000 data points, or a data set that occupies more than about five dimensions, Forgy's method is second best; in all other cases the proposed method is.

If there are more than about fourteen clusters in the data, then the proposed method will be the fastest, provided that the data set occupies less than about five dimensions. Second place would go to MacQueen's k-means, with Forgy's method third.

How does all this fit in with the sort of values that we would expect for real data sets, derived from multi-spectral imagery ?

A television quality image will comprise of three spectral bands

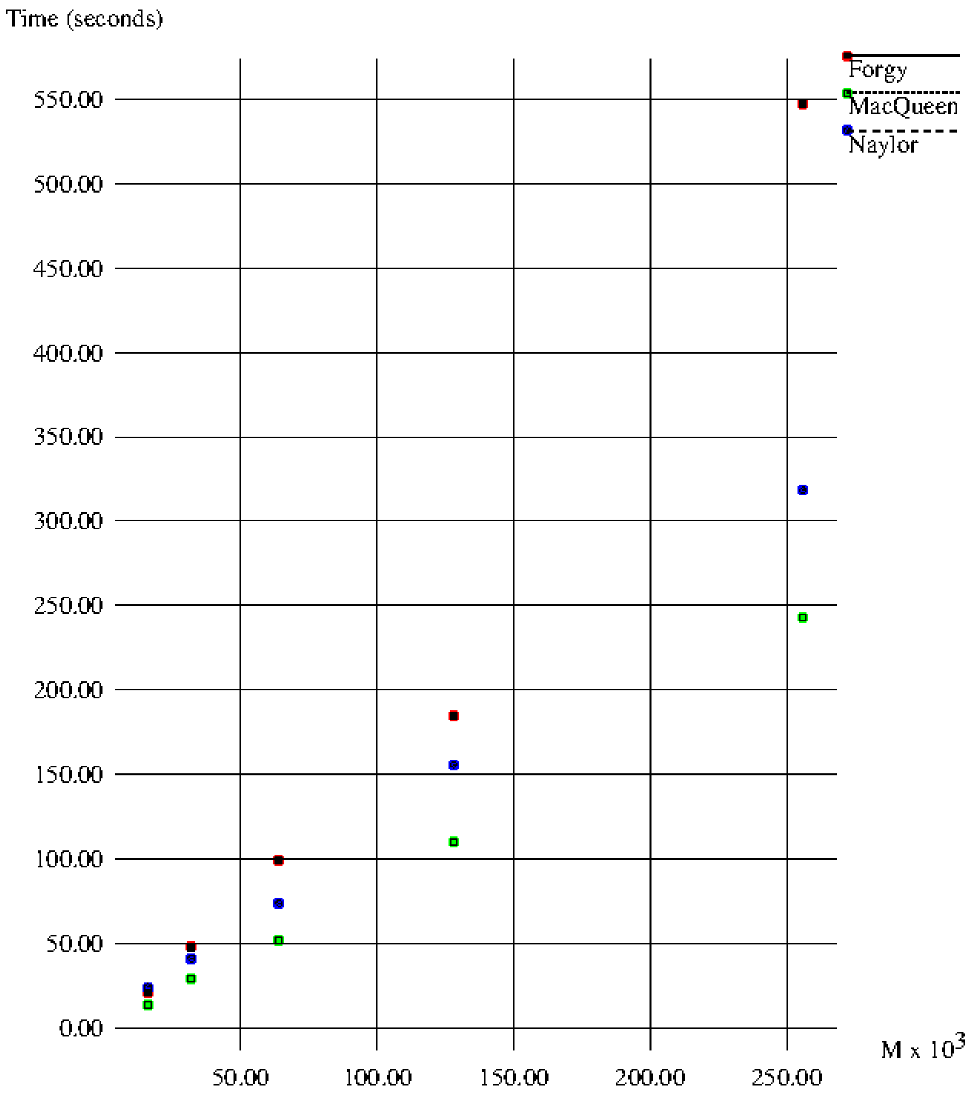


Figure 7.4: Variation in CPU time used with respect to data set size, for multiple iterations of Forgy's method & MacQueen's k-means.

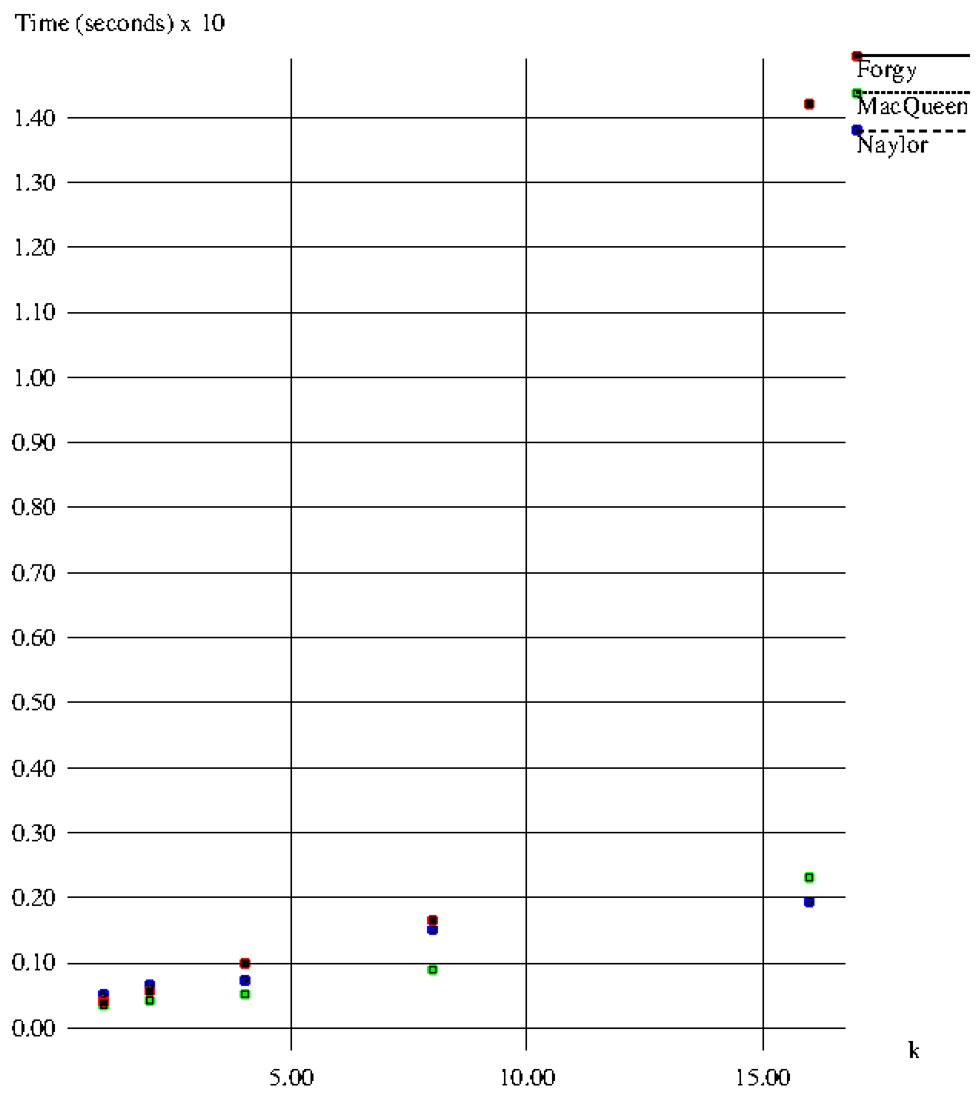


Figure 7.5: Variation in CPU time used with respect to the number of classes, for multiple iterations of Forgy's method & MacQueen's k-means.

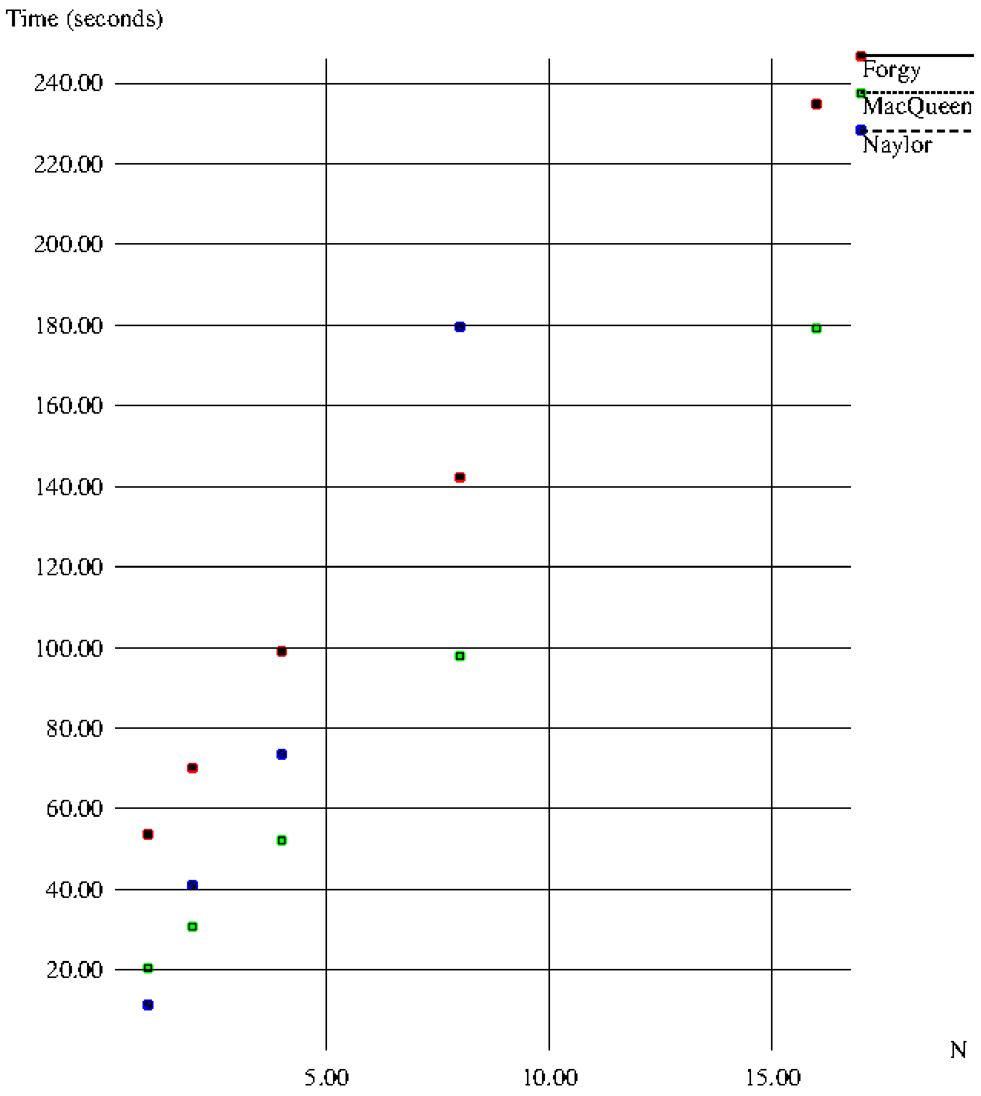


Figure 7.6: Variation in CPU time used with respect to the number of dimensions, for multiple iterations of Forgy's method & MacQueen's k-means.

($N = 3$), 512 by 512 pixels ($M = 262144$), and might contain anywhere between about six and sixteen classes ($6 \leq k \leq 16$). Figures 7.2 and 7.5 give us timings for $N = 4$, $k = 8$, and $M = 64000$, for $M = 262144$ we would expect all of the various methods to take at least four times longer than this. So, for a television image we can expect cluster location times of roughly 2 minutes for MacQueen's k-means, 3 minutes for Forgy's method, and 10 minutes for the proposed method – if k is known in advance. If k is unknown, then we can expect times of roughly 6 minutes for MacQueen's k-means, 10 minutes (still) for the proposed method, and 11 minutes for Forgy's method.

Taking k to the other end of the range ($k = 16$) and using figures 7.2 and 7.5 to determine appropriate factors for the change from $k = 8$, for each method, we get times of roughly 3 minutes for MacQueen's k-means, 9 minutes for Forgy's method, and 13 minutes for the proposed method – if k is known in advance. If k is unknown, then these become roughly 8 minutes for MacQueen's k-means, 13 minutes (still) for the proposed method, and 33 minutes for Forgy's method.

A full LANDSAT scene comprises seven spectral bands ($N = 7$), 7000 by 6000 pixels ($M = 4.2 \times 10^7$), and might be expected, again, to contain anywhere between about six and sixteen classes ($6 \leq k \leq 16$). Figures 7.3 and 7.6 give us timings for $N = 8$, $k = 4$, and $M = 64000$, for $M = 4.2 \times 10^7$ we would expect all of the various methods to take at least 660 times longer than this. Using $k = 8$, again, as an example of the lower bound of values for k we can get factors for the change from $k = 4$ from figures 7.2 and 7.5, for each method. This all results in cluster location times, for a full LANDSAT scene, of roughly 12 hours for MacQueen's k-means, 15 hours for Forgy's method, and 66 hours for the proposed method – if k is known in advance. If k is not known, then these times become roughly 36 hours for MacQueen's k-means, 39 hours for Forgy's method, and 66 hours (still) for the proposed method. If we go, as before, to $k = 16$, then we get times of roughly 15 hours for MacQueen's k-means, 48 hours for Forgy's method, and 83 hours for the proposed method – if k is known – and roughly 83 hours for both MacQueen's k-means and the proposed method, and 359 hours (nearly 15 days) for Forgy's method – if k is unknown.

7.2 Comparison with respect to quality :

Tables 7.10 to 7.22 show where the clusters were located, by each of the methods, in each of the data sets used for the speed comparisons. Ideally one would want to quantify the quality of the results by measuring the average error in the location of the positions of the clusters. This would require each cluster centre found by each method to be identified with a real cluster centre. Unfortunately, this is not possible – partly because the proposed method consistently over estimates the number of clusters in the data sets, and partly because when MacQueen's k-means goes wrong it goes so badly wrong that it is not possible to identify which clusters it thinks that it has found.

One way in which we can get a quantitative analysis of the quality of the cluster location is to define a permissible margin of error, and then see how many of the clusters locations determined by the various methods lie within this distance of a real cluster centre. If an algorithm finds the same cluster several times then only the first occasion counts as a 'hit', all the rest are 'misses'. The proposed method will automatically score four misses for the case where the program failed to terminate in a reasonable length of time.

Tables 7.23 and 7.24 show the number of hits and misses, respectively, for each method under consideration, and for a range of permissible margins of error. For ease of calculation the margins of error apply to the values of

Forgy	MacQueen	Naylor
		104
109.42	109.42	110
208.70	204.57	
	215.78	215
221.68		
234.82	231.26	232
		241

Table 7.10: Cluster centres found in 1 dimensional, 4 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(35.79, 67.07)		(36, 70)
(35.75, 73.35)		
	(80.48, 60.40)	
(118.46, 52.54)		(118, 52)
	(212.00, 126.76)	
	(216.56, 139.07)	
(218.19, 131.38)	(222.42, 129.27)	(219, 131)

Table 7.11: Cluster centres found in 2 dimensional, 4 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(215.59, 132.92, 193.61, 109.38)	(215.59, 132.92, 193.61, 109.38)	(216, 132, 194, 109)

Table 7.12: Cluster centres found in 4 dimensional, 1 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(215.54, 133.08, 193.61, 109.41)	(215.54, 133.08, 193.61, 109.41)	(213, 133, 194, 109)
(229.57, 103.22, 157.50, 35.78)	(229.57, 103.22, 157.50, 35.78)	(229, 103, 158, 36)

Table 7.13: Cluster centres found in 4 dimensional, 2 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(63.13, 20.06, 200.14, 35.79)	(63.13, 20.06, 200.14, 35.79)	(63, 20, 198, 36)
(200.34, 51.86, 89.09, 82.61)		(200, 52, 90, 82)
	(208.40, 131.17, 193.97, 109.36)	
	(215.70, 78.76, 124.92, 58.10)	
(215.37, 133.08, 193.61, 109.33)	(220.78, 134.57, 193.33, 109.31)	(223, 134, 195, 109)
(229.68, 103.24, 157.52, 35.79)		(227, 103, 157, 36)

Table 7.14: Cluster centres found in 4 dimensional, 4 class, 16000 sample data set.

Forgy	MacQueen	Naylor
(62.86, 19.98, 200.10, 35.80)	(62.86, 19.98, 200.10, 35.80)	(64, 20, 198, 36)
(200.28, 51.72, 88.94, 82.77)		(200, 52, 89, 82)
		(201, 52, 89, 105)
	(208.43, 131.38, 193.87, 109.34)	
(215.36, 133.11, 193.63, 109.37)	(220.85, 134.48, 193.45, 109.40)	(214, 134, 195, 109)
	(215.71, 78.72, 124.81, 58.20)	
(229.75, 103.30, 157.47, 35.84)		(229, 103, 158, 36)

Table 7.15: Cluster centres found in 4 dimensional, 4 class, 32000 sample data set.

Forgy	MacQueen	Naylor
(62.92, 20.00, 200.08, 35.78)	(62.92, 20.00, 200.08, 35.78)	(63, 20, 198, 36)
(200.27, 51.73, 88.94, 82.70)		(201, 52, 89, 83)
		(201, 52, 89, 105)
(215.56, 133.10, 193.64, 109.39)	(211.62, 127.86, 193.77, 109.40)	(214, 133, 195, 109)
	(215.63, 78.68, 124.82, 58.14)	
	(219.09, 137.79, 193.53, 109.37)	
(229.62, 103.22, 157.50, 35.78)		(229, 103, 158, 36)

Table 7.16: Cluster centres found in 4 dimensional, 4 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(62.96, 19.88, 200.15, 35.72)	(62.96, 19.88, 200.15, 35.72)	(63, 20, 198, 36)
(200.31, 51.74, 88.91, 82.64)		(200, 52, 89, 83)
		(200, 52, 89, 105)
	(215.62, 78.68, 124.81, 58.12)	
(215.54, 133.12, 193.61, 109.40)	(209.29, 130.78, 193.60, 109.46)	(214, 133, 195, 109)
	(221.16, 135.23, 193.62, 109.34)	
		(214, 107, 194, 109)
(229.57, 103.21, 157.51, 35.78)		(229, 103, 158, 36)

Table 7.17: Cluster centres found in 4 dimensional, 4 class, 128000 sample data set.

Forgy	MacQueen	Naylor
(62.96, 19.92, 200.18, 35.77)	(62.96, 19.92, 200.18, 35.77)	(63, 20, 198, 36)
(200.30, 51.73, 88.97, 82.71)		(201, 52, 89, 82)
		(201, 52, 89, 105)
	(215.66, 78.67, 124.82, 58.14)	
(215.56, 133.00, 193.62, 109.38)	(209.38, 130.72, 193.60, 109.41)	(214, 133, 195, 109)
	(221.25, 135.11, 193.63, 109.36)	
		(214, 107, 195, 109)
(229.64, 103.20, 157.46, 35.77)		(229, 103, 157, 36)

Table 7.18: Cluster centres found in 4 dimensional, 4 class, 256000 sample data set.

Forgy	MacQueen	Naylor
(16.27, 23.61, 86.96, 40.20)		(17, 22, 89, 38)
(23.64, 217.48, 126.35, 85.03)		
	(35.83, 22.03, 134.32, 38.33)	
(63.06, 19.85, 200.23, 35.73)		(63, 20, 198, 36)
(112.58, 161.30, 190.85, 68.30)		
		(123, 157, 112, 219)
	(136.30, 158.44, 136.98, 154.87)	(134, 158, 140, 153)
	(137.50, 158.30, 149.16, 153.77)	
(138.58, 155.92, 125.78, 190.43)		
	(139.44, 154.62, 116.24, 209.69)	
		(147, 157, 115, 206)
		(147, 157, 136, 206)
(200.33, 51.67, 88.99, 82.70)	(200.33, 51.67, 88.99, 82.70)	(200, 51, 89, 84)
(215.67, 132.82, 193.56, 109.44)	(215.67, 132.82, 193.56, 109.44)	(213, 133, 195, 109)
(229.71, 103.32, 157.49, 35.83)	(225.43, 103.14, 160.00, 35.85)	(229, 103, 157, 36)
	(235.29, 103.56, 154.22, 35.81)	

Table 7.19: Cluster centres found in 4 dimensional, 8 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(16.33, 23.61, 87.04, 40.26)	(16.33, 23.61, 87.04, 40.26)	(14, 21, 85, 38)
(24.91, 235.41, 221.18, 175.51)	(24.91, 235.41, 221.18, 175.51)	(25, 235, 221, 175)
(50.84, 94.96, 109.65, 49.61)	(50.84, 94.96, 109.65, 49.61)	(51, 93, 103, 46)
		(53, 97, 116, 54)
	(59.41, 20.11, 194.23, 34.09)	
(62.91, 20.04, 200.21, 35.78)	(61.89, 20.50, 207.57, 33.28)	(65, 20, 197, 36)
	(61.09, 17.85, 202.15, 41.51)	
	(69.58, 21.22, 198.96, 35.72)	
(74.07, 243.98, 90.62, 60.50)		
(106.82, 214.82, 120.01, 54.11)		(107, 217, 119, 54)
	(114.82, 219.16, 139.33, 73.99)	
(117.11, 227.91, 44.40, 52.88)	(117.11, 227.91, 44.40, 52.88)	(117, 230, 44, 54)
(122.63, 152.24, 111.51, 222.57)		(120, 157, 114, 219)
(123.74, 157.35, 111.60, 211.31)		
(129.15, 226.94, 173.93, 109.60)		(130, 232, 172, 110)
(135.21, 89.29, 82.53, 30.92)	(134.86, 89.51, 82.55, 30.77)	(133, 93, 83, 34)
(136.87, 158.35, 143.74, 154.30)	(136.87, 158.36, 143.74, 154.30)	(133, 158, 139, 154)
	(139.48, 154.63, 116.01, 209.77)	
(148.74, 154.62, 118.54, 205.51)		(149, 157, 117, 206)
	(163.07, 72.37, 84.69, 45.29)	
(200.34, 51.77, 89.25, 82.78)	(200.41, 51.73, 89.25, 83.10)	(202, 52, 87, 84)
		(205, 130, 168, 32)
		(206, 98, 169, 33)
(215.34, 132.86, 193.53, 109.41)	(215.34, 132.86, 193.53, 109.41)	(223, 134, 194, 110)
(220.59, 105.95, 161.84, 33.80)	(220.59, 105.95, 161.84, 33.80)	(227,99,155, 36)

Table 7.20: Cluster centres found in 4 dimensional, 16 class, 64000 sample data set.

Forgy	MacQueen	Naylor
		(70, 152, 76, 168, 121, 200, 52, 90)
(130.87, 206.17, 126.89, 204.04, 206.28, 136.67, 26.07, 41.63)	(130.87, 206.17, 126.89, 204.04, 206.28, 136.67, 26.07, 41.63)	
		(154, 227, 148, 218, 221, 113, 16, 23)
		(154, 227, 148, 218, 240, 113, 16, 23)
(215.64, 133.08, 193.68, 109.38, 160.14, 219.33, 130.72, 149.98)	(212.32, 135.42, 193.59, 109.40, 156.35, 219.41, 130.76, 149.95)	(214, 136, 194, 110, 159, 219, 130, 150)
	(220.04, 129.98, 193.80, 109.35, 165.15, 219.21, 130.66, 150.03)	
(219.38, 58.90, 190.14, 208.32, 23.64, 217.48, 126.35, 85.03)		
(229.11, 122.76, 154.43, 111.45, 217.49, 167.08, 224.47, 73.72)	(229.11, 122.76, 154.43, 111.45, 217.49, 167.08, 224.47, 73.72)	(229, 123, 150, 111, 218, 167, 224, 74)
		(229, 124, 161, 110, 218, 167, 224, 87)
		(229, 123, 151, 111, 237, 167, 224, 73)

Table 7.21: Cluster centres found in 8 dimensional, 4 class, 64000 sample data set.

Forgy	MacQueen	Naylor
(74.89, 109.42, 108.53, 145.95, 150.63, 204.06, 250.14, 217.97, 34.77, 73.58, 168.70, 126.05, 220.36, 59.75, 9.10, 52.65)		
	(82.57, 75.16, 87.47, 200.74, 73.53, 215.24, 137.01, 158.21, 143.79, 154.16, 226.81, 146.61, 217.77, 238.84, 112.30, 16.30)	
(106.80, 214.71, 119.87, 54.10, 74.59, 64.64, 169.53, 58.93, 115.68, 53.33, 99.06, 116.49, 54.54, 74.52, 102.64, 217.24)		
(133.25, 129.64, 110.24, 171.15, 128.22, 154.79, 107.21, 161.21, 152.64, 136.65, 215.57, 117.58, 188.55, 185.93, 101.75, 21.46)		
	(140.27, 218.76, 128.86, 71.84, 117.76, 58.96, 136.69, 89.25, 130.62, 67.78, 126.08, 101.89, 77.24, 78.70, 96.96, 164.80)	
(215.58, 133.10, 193.61, 109.40, 160.12, 219.35, 130.71, 150.03, 24.82, 62.96, 19.90, 200.01, 35.79, 137.29, 96.12, 74.81)	(211.46, 133.46, 193.69, 109.48, 163.70, 219.19, 130.38, 150.07, 24.86, 63.11, 19.98, 199.84, 35.63, 137.11, 96.11, 74.87)	
	(221.16, 132.60, 193.49, 109.31, 155.28, 219.56, 131.15, 149.97, 24.77, 62.77, 19.79, 200.24, 36.00, 137.54, 96.14, 74.73)	

Table 7.22: Cluster centres found in 16 dimensional, 4 class, 64000 sample data set.

each coordinate, rather than the Euclidean distance between the locations – that is they define (hyper-)cubes around the cluster locations, rather than (hyper-)spheres. The results are expressed as a percentage of the total number of clusters in the data sets. The results for the proposed method add up to more than 100% because of its over estimation of the number of clusters.

Margin of error	+/− 1	+/− 2	+/− 4	+/− 8
Forgy's method	73%	73%	81%	84%
MacQueen's k-means	33%	36%	43%	57%
Naylor's method	32%	63%	82%	90%

Table 7.23: Hits scored by the various methods, at different permissible margins of error.

Margin of error	+/− 1	+/− 2	+/− 4	+/− 8
Forgy's method	27%	27%	19%	16%
MacQueen's k-means	67%	64%	57%	43%
Naylor's method	86%	55%	36%	28%

Table 7.24: Misses scored by the various methods, at different permissible margins of error.

None of the methods gives particularly impressive results, even at the most generous margin of error.

MacQueen's k-means gives consistently poor results, this is because, in its single pass through the data, it has no opportunity to correct for poor initial estimates of the cluster locations. It may well be the fastest of the three methods, but the quality of its results is so poor as to make it virtually unusable.

The way in which the quality of the results for Forgy's method slowly increases, as the permissible error increases, indicates that whilst the majority (73%) of its cluster locations are 'spot on', the ones which it does get wrong (presumably also because of poor initial estimates) are so far wrong as to be useless. This, combined with the erraticness of its execution time, goes to make Forgy's method so unpredictable as to make its results just as untrustworthy as those of MacQueen's k-means.

The proposed method is just as bad as MacQueen's k-means at scoring 'direct hits', however, as the permissible error increases, its score continually rises towards a respectable level. This tends to suggest that, whilst it may be producing rather sloppy results, it is not getting things as wrong as Forgy's method or MacQueen's k-means do. The addition of some sort of 'fine tuning' stage, at the end of the algorithm, might result in a useful ($\sim$ 95%) score, even at a low permissible error. The numebr of misses remains high, for all margins of error, because of the continual over estimation of the number of classes. This needs to be dealt with too, before the method can be considered truly useful. The results for the one dimensional feature space are particularly poor. This is because there is no peak linkage stage to weed out false peaks. The algorithm should probably carry a warning that it should only be used in two, or more, dimensions.

Finally, a more visual representation of the quality of the results produced by the various methods is provided in figures 7.7 to 7.13. These segmentations were performed using the versions of the programs used for the first set of speed tests. The data set is the BBC testcard image, after having all of the lighting effects removed (see figure 4.4).

Figure 7.7 shows the result of segmenting the image using the proposed method. Six classes have been found, and, apart from a scattering of 'flesh' pixels having been classified as 'yellow', the result is quite acceptable.

Figures 7.8 to 7.10 show the result of segmenting the image using Forgy's method, with k specified as 5, 6, & 7 respectively. The 'yellow' cluster (the table cloth) has not been found as a distinct entity in any of the segmentations. This is the smallest cluster in the data set, but it is large enough that one would expect it to be found. Also, for $k = 5$ & 6, only four classes have resulted in the segmentation, and, for $k = 7$, there are only five classes. This means that Forgy's method is getting some of the cluster locations so far wrong that they do not actually relate to any of the data points.

Figures 7.11 to 7.13 show the result of segmenting the image using MacQueen's k-means, again with k specified as 5, 6, & 7. Again, the 'yellow' cluster is never found as a distinct entity. Also, k clusters seem to be obtained by just splitting the 'colourless' class (the blackboard and borders) over more and more classes.

This really all just goes to reinforce what we have already seen about how well the various methods actually cope with locating clusters.

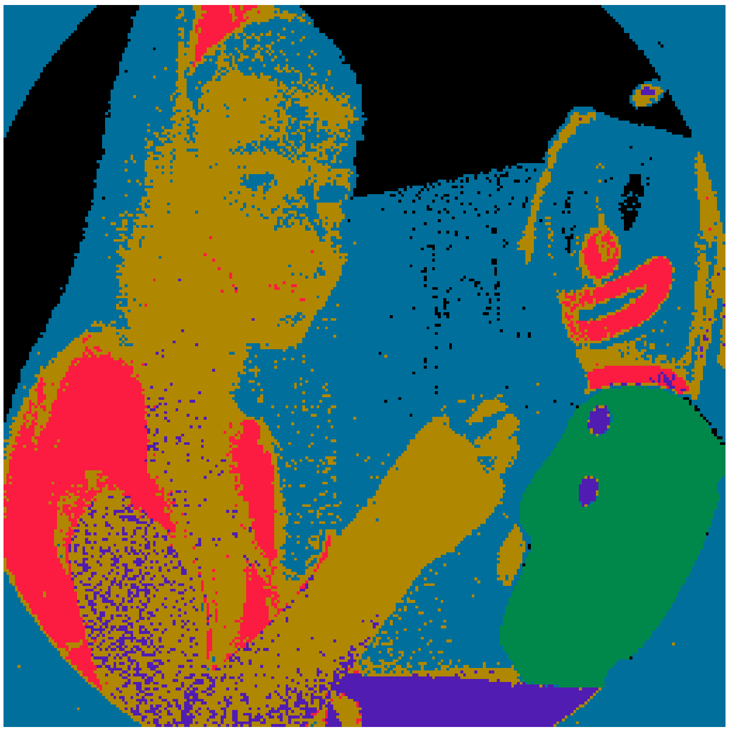


Figure 7.7: Result of segmenting the BBC testcard using Naylor's method (6 classes found).

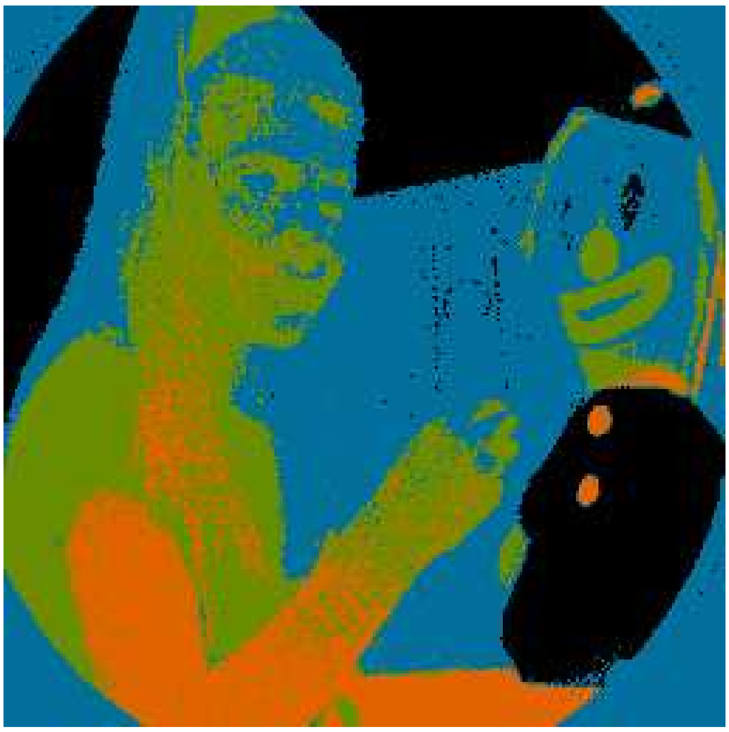


Figure 7.8: Result of segmenting the BBC testcard using Forgy's method ($k = 5$).

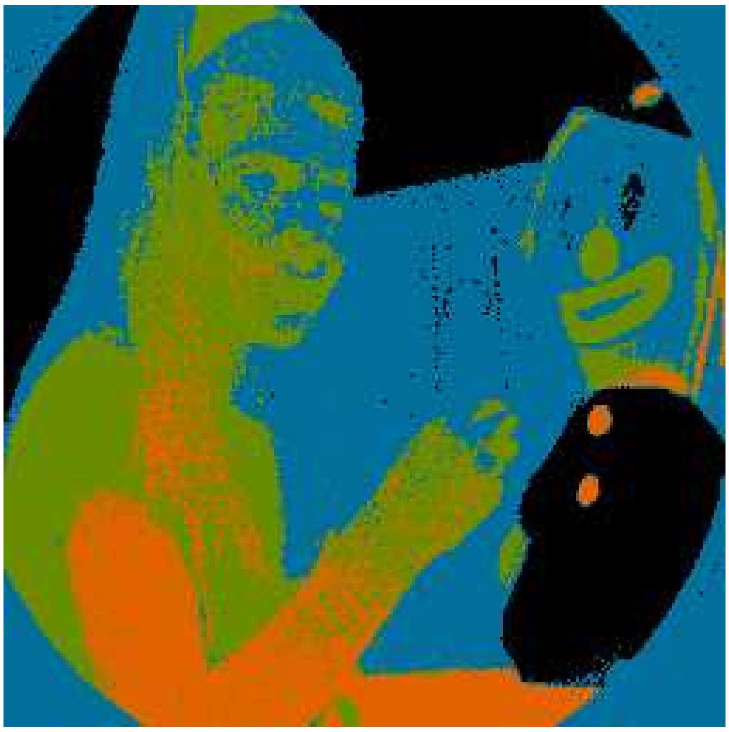


Figure 7.9: Result of segmenting the BBC testcard using Forgy's method ($k = 6$).

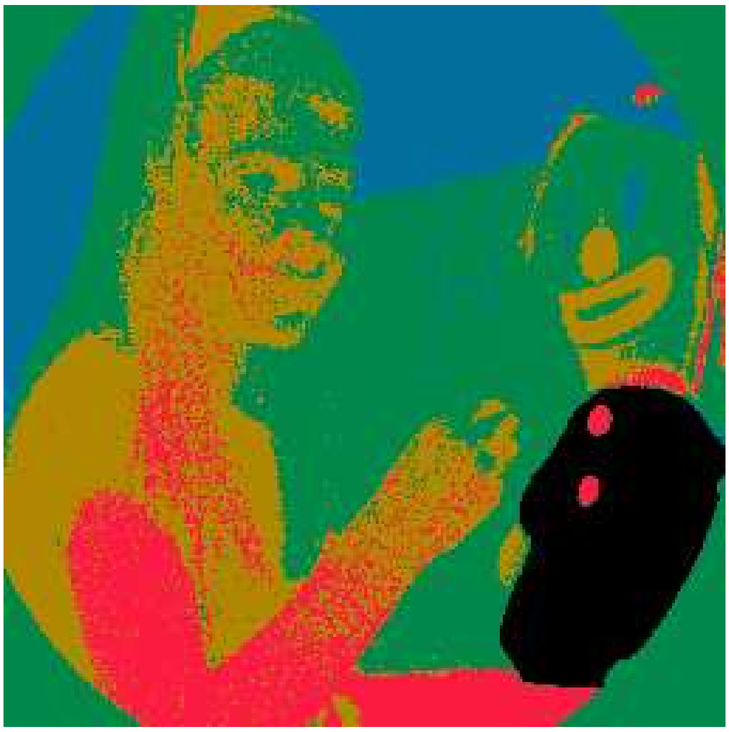


Figure 7.10: Result of segmenting the BBC testcard using Forgy's method ($k = 7$).

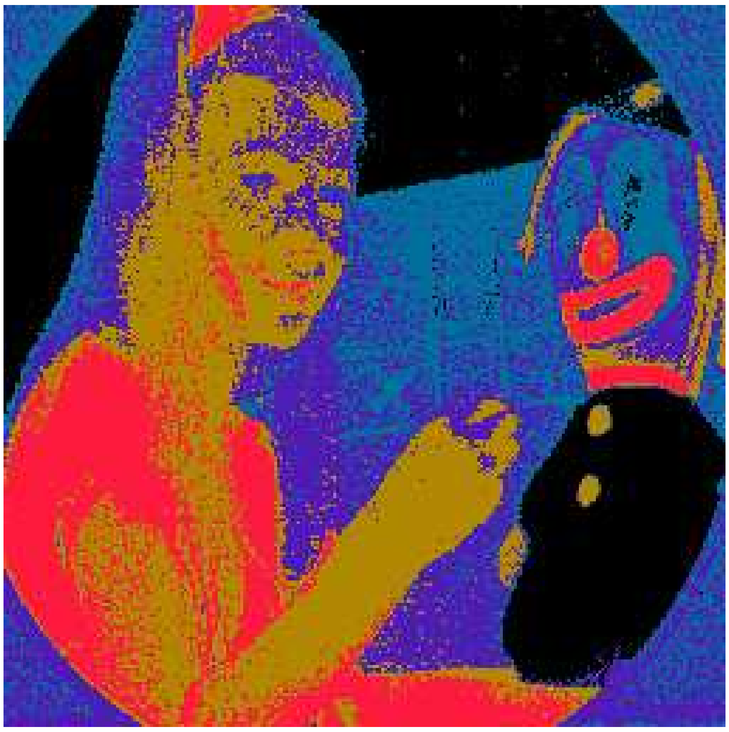


Figure 7.11: Result of segmenting the BBC testcard using MacQueen's k-means ($k = 5$).

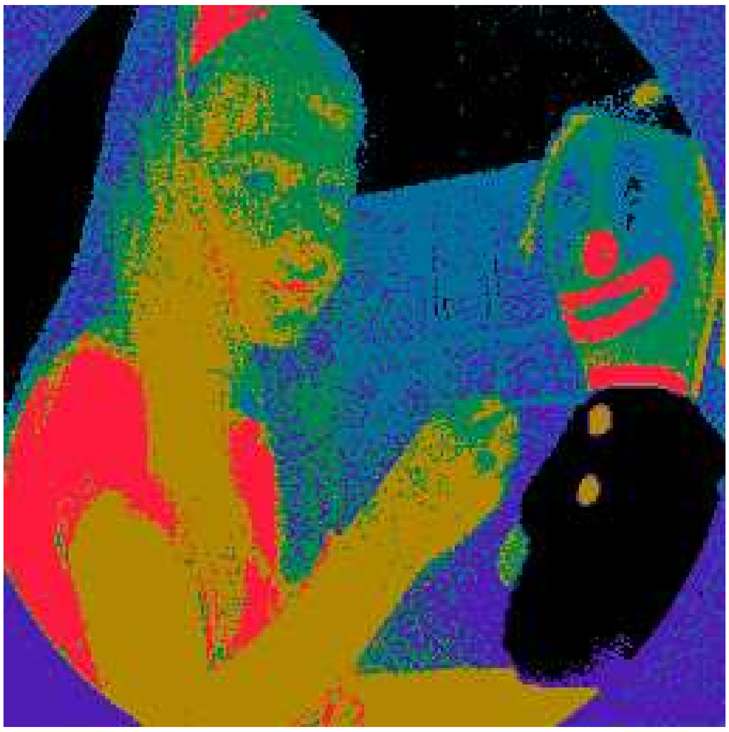


Figure 7.12: Result of segmenting the BBC testcard using MacQueen's k-means ($k = 6$).

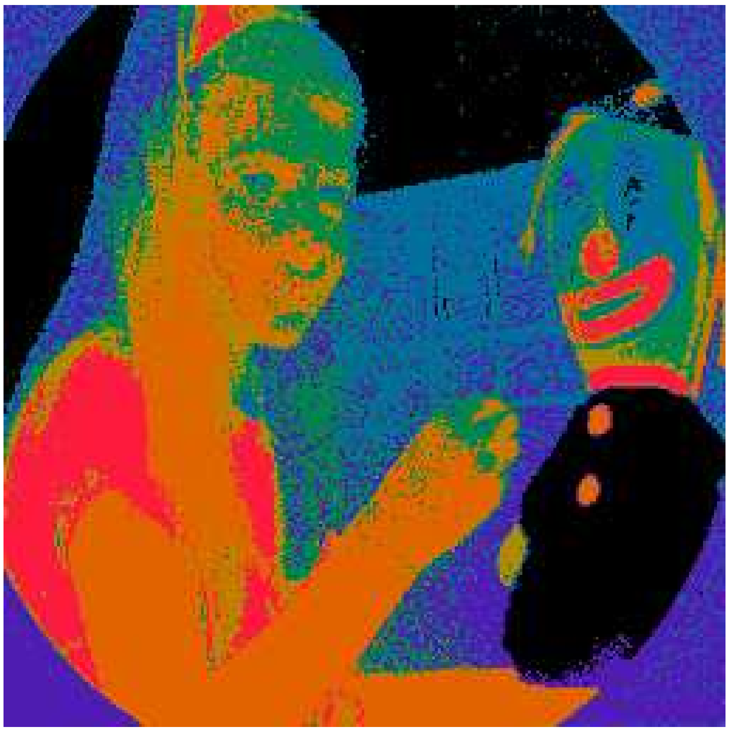


Figure 7.13: Result of segmenting the BBC testcard using MacQueen's k-means ($k = 7$).

CHAPTER 8
Conclusions and Future Work

The results obtained using this new method of cluster location, particularly those presented in the preceeding chapter, seem to indicate that, whilst the underlying principle is sound, the ways in which various parts of the algorithm have actually been implemented leave room for improvement. The weak points can be summarised as follows :

- the location of peaks in one dimension could do with being able to cope better with merged peaks – no amount of modification of the later stages of the algorithm can correct for information which is substantially wrong to start with.
- the peak linkage stage needs to be done in a manner which has a time dependence somewhat less than k^N – this is necessary if the method is to be used in the high dimensional feature spaces for which it was intended.
- the peak linkage stage, again, needs to be improved so that no false peaks at all survive – otherwise it would be premature to claim that the cluster validation problem had been solved.
- the accuracy of the final peak locations needs to be greatly improved, by some sort of fine tuning – otherwise the results will be no good for anything other than very rough segmentations.

The first of these problems could be greatly reduced by using a fuller implementation of the CLEAN algorithm. Removing only small portions of the peaks on each iteration, and allowing for negative peaks – caused by accumulated errors – should result in a much more accurate picture of the peak locations being obtained. However, fragmenting the peaks will make it difficult to accurately recover details of their widths and heights. This would mean that the results were no longer suitable for use with cooccurrence segmentation.

A possible solution to the second problem would be to do the peak linkage one dimension at a time. That is, the peaks found in two one dimensional histograms could be analysed to see how they correspond to peaks in two dimensions. The surviving, two dimensional, candidates could then be linked with the peaks from a third histogram to produce a set of three dimensional candidates, and so on. Since the number of surviving candidates at each stage, and the number of peaks found in each histogram, will be of order k, this should result in an execution time which is dependent on Nk^2, rather than k^N. For large values of N and K this would imply a very considerable speed up.

The third problem is probably the most difficult one associated with the use of this algorithm. In theory there are plenty of solutions, as detailed in chapter 3, but, in practice, the solutions are unusable either because they are too slow when employed in a high dimensional feature space (particularly one which cannot be stored in the computer's memory), or because they are very sensitive to any noise on the distributions being analysed. No obviously workable solution to this problem springs to mind, so finding one should probably be considered the most urgent task for any future work.

The final problem has a very simple solution. This would be to use the cluster locations determined by the new method, and the value for the number of clusters, as the starting point for a classification using Forgy's method (or something similar). This would make use of Forgy's method's ability to produce accurate values for the cluster centre locations, whilst removing the problems of its needing to know how many clusters to look for, and of its occasionally going awry when given poor initial estimates of the cluster locations.

The cooccurrence segmentation of multi-spectral imagery has produced some interesting results, but, if the author was doing this work again, he would probably omit it. The cluster location/image segmentation problem is hard enough, as has been seen, without the added constraint of having to estimate peak widths, and the added problems of having to do the classification in a $2N$ dimensional feature space, whilst also producing an edge strength map. Admittedly, not using coocurrence information would result in slightly more fragmented segmentations – compare figure 7.7, which does not use cooccurrence, with figure 4.25, which does – but there are post processing solutions to this problem, for example 'Besag smoothing' [Besag, 1986]. The benefits just do not seem to outweigh the difficulties. Anyway, it would appear, from the results in chapter 4, that the removal of the lighting effects from a multi-spectral image plays a far more important part in ensuring a good looking segmentation.

Also, trying to produce an algorithm which would work in an arbitrarily high number of dimensions was probably being rather overenthusiastic. It would have been better to have kept the problem simple, at least to start with, for example the segmentation of three band images television type images. Results could still have been achieved for the remote sensing work, either by selecting the three bands which contain information pertinent to the required classification, or by using the first three principal components. This would have had the added bonus of making the analysis of the remote sensing results easier – since all the spectral information that was being used by the algorithm could have been visualised as a colour composite.

Over the lifetime of this project the execution time of the program has decreased dramatically, largely due to the availability of increasingly high performance computers. The very first version of the program would take several days to segment a 512 pixel square, 7 band, image on a DEC VAX 11/780. A subsequent version would run in a day on a DEC VAX 8500, and the final version runs in a couple of hours on a Sun SPARCstation II. All of these machines have sequential architectures, the structure of the algorithm, though, is inherently parallel. The production and analysis of the one dimensional histograms could all be performed on a multiple instruction/multiple data (MIMD) architecture machine, e.g. something based on transputers, with one processor per histogram. Similarly all the pixel manipulation processes – smoothing the image, removing lighting effects, and labelling the pixels – are ideal for implementation on a single instruction/multiple data (SIMD) architecture, e.g. a distributed array processor (DAP), where you can have one processor per pixel.

In fact, a small part of the labelling process was modified to make use of a DAP attached to a Sun SPARCstation at BP. This resulted in an, overall, fourfold increase in speed. This may not seem too impressive for something containing 1024 processors, but only a very small amount of memory was available on each processor. This meant that the data had to be read onto, and from, the DAP in several pieces, and it is this, data transfer, process which is the main limiting factor on such machines. A larger number of processors, or a larger amount of memory on each processor, would hopefully have secure a much more significant increase in speed.

Given the unlikelihood of a combined MIMD/SIMD computer being made available to the author, perhaps the most practical solution to speeding up the program (from the hardware side of things) would be to rewrite it to work across a collection of networked workstations. The smoothing and lighting effect removal could be spread across all the available machines in a network by giving each of them a part of the image to work on. Inevitably one will finish before all the rest, and that machine would be given the job of analysing the histogram of one of the bands (assuming that there is a band which has finished being smoothed, etc.). As other machines complete their tasks they too can start work on histogram analysis. Once all the histograms have been assigned to machines all the rest will be left idle – unfortunately there is nothing else which they can be getting on with at this stage. However, the next stage will keep all the machines busy as the list of candidate cluster centres is divided between them for the tests of local maximality. Once the final set of, real, cluster centres has been determined the image would again be split up across all of the machines for the classification stage.

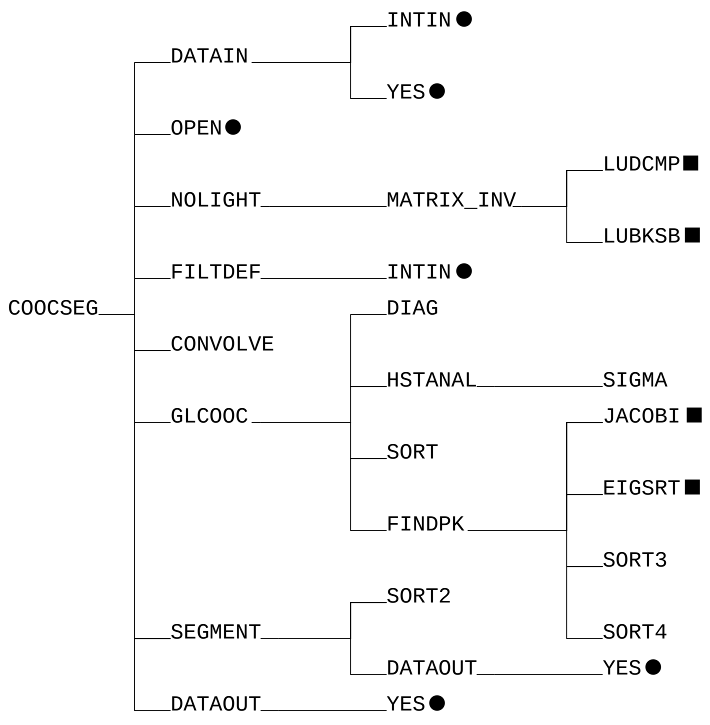
Obviously the sort of speed increase which would be possible using this method would depend on the number of workstations available. However, with a couple of dozen it should be possible to get to the point where, for television type images at least, the speed is limited by such things as synchronisation problems, interprocessor communication times, and the time taken to read the data in and write the results out, rather than by the actual time taken to process the data. This is still unlikely to mean that the program will run in 'real time' (i.e. at video rates). For this one would really require, specially designed, dedicated hardware.

In conclusion then, it is probably safe to claim a limited success for this new technique. Fortunately, it is clear what needs doing to improve things, even if it is not always obvious how that can actually be achieved. Given time, and the opportunity, the author would certainly like to pursue these hopeful beginings through to a more positive outcome.

Appendix A

The FORTRAN source code for the image segmentation program.

The structure of the program is as follows :



● - "ERIC" routines.

■ - "Numerical Recipes" routines.

COOCSEG is the main program. DATAIN reads ‘raw’ (single byte per pixel) data from a file. NOLIGHT removes the lighting effects from an image. MATRIX_INV inverts matrices. FILTDEF defines a smoothing (noise reduction) filter. CONVOLVE convolves the image with the filter. GLCOOC forms and analyses cooccurrence matrices. DIAG constructs the leading diagonal of the cooccurrence matrix of an image. HSTANAL analyses a one dimensional sub-space, looking for peaks. SIGMA fits a Gaussian curve to a peak. The various SORT routines sort different numbers of arrays, in different ways. FINDPK determines who the peaks found by HSTANAL link together to define cluster centres. SEGMENT segments an image in cooccurrence space, and produces an edge probability image. And, DATAOUT writes out various types of image data to a file.

The "ERIC" routines are a set of functions, and subroutines, used at King's College to provide a robust user interface to programs. INTIN gets an integer value from the user; OPEN opens a specified file; and YES gets a "yes" or "no" answer to a question.

The "Numerical Recipes" routines are taken from "Numerical Recipes - The Art of Scientific Computing" by W. H. Press, B. P. Flannery, S. A. Teukolsky, & W. T. Vetterling, published by Cambridge University Press. LUDCMP performs LU-decomposition on a matrix; LUBKSB does back substitution on the decomposed matrix; JACOBI finds the eigenvalues, and eigenvectors, of a matrix; and EIGSRT sorts the eigenvectors according to the values of the eigenvalues.

The program is nowhere near as modular as it could be. In particular, FINDPK is rather rambling. This is due to the programs being still, very much, a development version.

Apart from in one, or two, places, no effort has been made to optimise the program with respect to speed. This is because the most important factor in the writing of the first version was getting it to fit in the 6 Mb of available memory. Even when computers with more memory became available, speed optimisation was not really a consideration, since the new machines were faster anyway.

The code has been extensively commented, in order to allow for easy development (possibly by other people), and for easy translation into other languages (almost definitely by other people). It should be possible to understand what the program is doing without having to resort to 'reading' the `FORTRAN` code.

COOCSEG :

```
c
c +---------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.   |
c |   work at King's College London, 1988-1992.  This work was funded   |
c |   by the SERC and BP Research International.  This software comes   |
c |   with no guarantees, and you use it at your own risk.  The author  |
c |   will probably be prepared to help you out with any problems, if   |
c |   you can track him down.            Philip J. Naylor, July 1992.   |
c +---------------------------------------------------------------------+
c
c ---------------------------------------------------------------------
c #####################################################################
c ---------------------------------------------------------------------
c
      program coocseg
c
c   ***************************************************************
c   *                                                             *
c   *           a program to segment multi-spectral               *
c   *           images, using cooccurrence matrices.              *
c   *  [n.b. this program uses routines from "Numerical Recipes -  *
c   *    the Art of Scientific Computing" by Press, Flannery,      *
c   *     Teukolsky, and Vetterling, Cambridge University Press,   *
c   *         also various "in house" robust I/O routines.         *
c   *                                                             *
c   ***************************************************************
c
c =====================================================================
c >>>>>>>>>>>>           variable declarations            <<<<<<<<<<<<
c ---------------------------------------------------------------------
c integer variables :
c
c   band         - the spectral band number.
c   count        - loop counter.
c   fsize        - the cooccurrence distance to be used, i.e. length of
c                  half cooccurrence filter.
c   felem        - element number in half coocurrence filter.
c   nband        - the number of spectral bands.
c   nclass       - the number of classes found in the image.
c   num          - record number, used when reading a (direct access)
c                  byte data file.
c   numlev       - the number of grey levels to which image is
c                  digitized.
c   offax        - the off-'diagonal' distance, in cooccurrence space,
c                  which defines the boundary between region pixels and
c                  edge pixels.
```

```
c   pos          - indicates position in an array.
c   positn       - alternative indication of position in an array.
c   xsize        - the size of the image, in the x direction.
c   ysize        - the size of the image, in the y direction.
c
      integer    band,count,fsize,felem,nband,nclass,num,numlev
      integer    offax,pos,positn,xsize,ysize,bnd
c
c ---------------------------------------------------------------------
c integer arrays :
c
c   classave     - the nband dimensional vectors, giving the along-axis
c                  positions of the region cluster centres.
c   classwid     - the along-axis half-widths of the region clusters,
c                  in each of the nband dimensions.
c
      integer    classave(16,800),classwid(16,800)
c
c ---------------------------------------------------------------------
c byte arrays :
c
c   images       - the (xsize by ysize) by nband image data.
c   segim        - the xsize by ysize segmented image.
c
      byte       images(4194304),segim(262144)
c
c ---------------------------------------------------------------------
c real variables :
c
c   area         - the area under a particular section of the
c                  cooccurrence filter.
c   disp         - the displacement from the centre of the cooccurrence
c                  filter of a particular element of the half filter.
c   dummy        - dummy variable for passing to dataout.
c   f            - the value taken by a pariticuar element of the
c                  un-normalized half cooccurrence filter.
c   rtemp        - temporary storage for a real value.
c   total        - the total area under the un-normalized half
c                  cooccurrence filter.
c
      real       area,disp,dummy,f,total
c ---------------------------------------------------------------------
c real arrays :
c
c   filter       - the, normalized, smoothing filter.
c
      real       filter(2500)
```

```
c
c ----------------------------------------------------------------------
c character variables :
c
c   screen       - this variable holds an escape sequence which is used
c                  to clear the screen, and send the cursor 'home'.
c
      character  screen*7
c
c ----------------------------------------------------------------------
c ======================================================================
c >>>>>>>>>>>>            function declarations            <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer functions :
c
c   intin        - 'eric' library routine, used to input an integer, in
c                  response to a given prompt.
c
      integer    intin
c
c ----------------------------------------------------------------------
c ======================================================================
c ######################################################################
c ----------------------------------------------------------------------
c
c
c the start of the program proper :
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c define a string, which, when sent to the terminal, will clear the
c screen, and send the cursor 'home' :
c
      screen=char(27)//'[2j'//char(27)//'[h'
c
c ----------------------------------------------------------------------
c announce the program :
c
      write(*,*) screen
      write(*,'(11x,a58///)') 'Cooccurrence segmentation program, versio
     &n 3.4  (PJN 1989)'
c
c ----------------------------------------------------------------------
c read in the data (the data are stored internally as signed bytes, i.e.
c values in the range -128 to 128) :
c
```

```
      call datain(images,xsize,ysize,nband)
      numlev=256
c
c ----------------------------------------------------------------------
c set up a log file :
      call open('Log file >','coocseg','log',9,'fn')
c
c ----------------------------------------------------------------------
c if there is more than one spectral band, remove the lighting effects,
c by dividing each band by the sum of all the bands :
c
      if (nband .gt. 1) then
          call nolight(images,xsize,ysize,nband,numlev)
      end if
c
c ----------------------------------------------------------------------
c find out what type and size of filter should be used :
c
      call filtdef(filter,fsize)
      if (fsize .gt. 1) call convolve(images,xsize,ysize,nband,
     &                                  filter,fsize)
c
c ----------------------------------------------------------------------
c call the subroutine which produces, and analyses, the cooccurrence
c matrices :
c
      call glcooc(images,xsize,ysize,nband,numlev,classave,
     &            classwid,nclass,(fsize/2)+1)
c
c ----------------------------------------------------------------------
c call the subroutine which 'maps' the image through the 'labelled'
c cooccurrence matrix, in order to segment it :
c
      call segment(images,xsize,ysize,nband,classave,
     &             classwid,nclass,segim,(fsize/2)+1)
c
c ----------------------------------------------------------------------
c call the subroutine which 'relaxes' the segmentation labelling :
c
c                     ** not implemented as yet **
c
c     write(*,*) screen
c     call relax(segim,xsize,ysize,nclass)
c
c ----------------------------------------------------------------------
c output the segmented image (in readar format) :
c
```

```
      call dataout('File for segmented image',segim,dummy,xsize,ysize,
     &                                                            1,1)
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c the end of the main program :
c
      close(9,status='keep')
      write(*,*) screen
c
c ----------------------------------------------------------------------
c ######################################################################
c ======================================================================
c ----------------------------------------------------------------------
c
      end
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

DATAIN :

```
c
c +----------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.    |
c |   work at King's College London, 1988-1992.  This work was funded    |
c |   by the SERC and BP Research International.  This software comes    |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if    |
c |   you can track him down.            Philip J. Naylor, July 1992.    |
c +----------------------------------------------------------------------+
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
      subroutine datain(images,xsize,ysize,nband)
c
c ***********************************************************************
c *                                                                     *
c *       a subroutine to read image data from a raw byte file.         *
c *                                                                     *
c ***********************************************************************
c
c =======================================================================
c >>>>>>>>>>>>              variable declarations            <<<<<<<<<<<<
c -----------------------------------------------------------------------
c integer variables :
c
c band          - the spectral band number.
c nband         - the number of spectral bands.
c num           - the record number in the file.
c positn        - the position within the image.
c xsize         - the size of the image (x).
c ysize         - the size of the image (y).
c
      integer    band,nband,num
      integer    pos,positn,xsize,ysize
c
c -----------------------------------------------------------------------
c byte arrays :
c
c images        - the image data.
c
      byte       images(4194304)
c
c -----------------------------------------------------------------------
c logical variables :
```

```
c
c again          - true if a file read or open is to be reattempted after
c                  an error.
c square         - true if the image is square.
c
      logical     again,square
c
c ------------------------------------------------------------------------
c character variables :
c
c filename       - the name of the file containing a band of the image.
c
      character   filename*80
c
c ----------------------------------------------------------------------
c ======================================================================
c >>>>>>>>>>>>            function declarations             <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer functions :
c
c   intin        - 'eric' library routine, used to input an integer, in
c                  response to a given prompt.
c
      integer     intin
c
c ----------------------------------------------------------------------
c logical functions :
c
c   yes          - 'eric' library routine, used to get a yes or no
c                  answer to a given question.
c
      logical     yes
c
c ----------------------------------------------------------------------
c ======================================================================
c ######################################################################
c ----------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c get details of the data :
c
        square=yes('Is the image square ? >',.true.)
        if (square) then
```

```
          xsize=intin('Image size >',512,64,512)
          ysize=xsize
        else
          xsize=intin('Image size in x-direction >',512,64,1024)
          ysize=intin('Image size in y-direction >',xsize,64,
     &                                            262144/xsize)
        end if
        nband=intin('Number of spectral bands >',3,1,32)
c
c ------------------------------------------------------------------
c loop over the spectral bands :
c
        do 1,band=1,nband,1
c
c --------------------------------------------------------------------
c get the name of the file (assumes there will not be more than 26
c bands) :
c
 1000       write(*,666) 'Image file '//char(64+band)//' >'
            read(*,'(a)') filename
 666        format(1x,a,' : ',$)
c
c --------------------------------------------------------------------
c open the file :
c
            open (unit=1,file=filename,access='direct',status='old',
     &          organization='sequential',recl=128,err=2)
c
c --------------------------------------------------------------------
c read in the data (the data are stored internally as signed bytes,
c i.e. -128 to 127, but stored in the files as signed bytes,
c i.e. 0 to 256) :
c
            do 3,num=1,(xsize*ysize)/512,1
              read(1,rec=num,err=4) (images(((band-1)*(xsize*ysize))
     &                  +positn),positn=((num-1)*512)+1,num*512)
    3       continue
c
c --------------------------------------------------------------------
c take account of 2's complementation :
c
            do 6,positn=((band-1)*xsize*ysize)+1,(band*xsize*ysize),1
              if (images(positn) .lt. 0) then
                images(positn)=images(positn)+128
              else
                images(positn)=images(positn)-128
              end if
    6       continue
c
c ------------------------------------------------------------------------
c close the file
c
          close(1)
c
c ------------------------------------------------------------------------
c go on to the next spectral band :
c
    1   continue
c
c ------------------------------------------------------------------------
c skip to the end :
c
        goto 3000
c
c ----------------------------------------------------------------------
c ######################################################################
c ======================================================================
c ----------------------------------------------------------------------
c code for 'dealing' with problems which occur whilst handling byte data
c files :
c
    2 write(*,*) 'Unable to open file ',filename
      again=yes('Try again ? >',.true.)
      if (again) goto 1000
      stop
    4 write(*,*) 'Error in reading from file ',filename
      again=yes('Try again ? >',.false.)
      close(1)
      if (again) goto 1000
      stop
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c
 3000 end
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

NOLIGHT :

```
c
c +------------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.    |
c |   work at King's College London, 1988-1992.  This work was funded    |
c |   by the SERC and BP Research International.  This software comes    |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if    |
c |   you can track him down.             Philip J. Naylor, July 1992.   |
c +------------------------------------------------------------------------+
c
c -------------------------------------------------------------------------
c #########################################################################
c -------------------------------------------------------------------------
c
      subroutine nolight(images,xsize,ysize,nband,numlev)
c
c *************************************************************************
c *                                                                       *
c *         a subroutine to remove the lighting effects from a           *
c *      multi-spectral image, by removing the overall intensity         *
c *                            variations.                                *
c *                                                                       *
c *************************************************************************
c
c =========================================================================
c >>>>>>>>>>>>               variable declarations          <<<<<<<<<<<<
c -------------------------------------------------------------------------
c integer variables :
c
c band          - spectral band number.
c bnd           - spectral band number.
c bnd2          - spectral band number.
c itemp         - integer pixel value.
c nband         - number of spectral bands.
c numlev        - number of geylevels per band.
c xsize         - image size in x direction.
c ysize         - image size in y direction.
c
      integer    band,nband,numlev
      integer    bnd2,xsize,ysize,bnd,itemp
c
c -------------------------------------------------------------------------
c byte arrays :
c
c images        - the image data.
c
```

```
      byte       images(xsize,ysize,nband)
c
c ---------------------------------------------------------------------------
c real variables :
c
c i             - real pixel value.
c tot           - normalisation parameter.
c
      real       tot,i
c
c ---------------------------------------------------------------------------
c real arrays :
c
c trans         - the forward transformation matrix.
c itrans        - the reverse transformation matrix.
c dtrans        - the combined forward and reverse transformation matrix,
c                 including setting intensity component to zero in
c                 between).
c inten         - the colour of a pixel (i.e. its spectral values).
c
      real       trans(16,16),itrans(16,16),dtrans(16,16),inten(16)
c
c ---------------------------------------------------------------------------
c ===========================================================================
c ###########################################################################
c ---------------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c ---------------------------------------------------------------------------
c ===========================================================================
c ---------------------------------------------------------------------------
c announce what stage the program has reached :
c
      write(*,*) ' Removing the lighting effects.'
c
c ---------------------------------------------------------------------------
c define the forward transformation matrix :
c
      do 10,band=1,nband,1
        if (band .eq. 1) then
          do 11,bnd=1,nband,1
            trans(bnd,band)=1
   11     continue
        else
          do 12,bnd=1,band-1,1
            trans(bnd,band)=-1
   12     continue
          trans(band,band)=band-1
          do 13,bnd=band+1,nband,1
            trans(bnd,band)=0
   13     continue
        end if
   10 continue
c
c -----------------------------------------------------------------------
c invert the matrix :
c
      call matrix_inv(trans,itrans,nband,16)
c
c -----------------------------------------------------------------------
c modify the inverse matrix so that the intensity component is set to
c zero (n.b. intensities are in the range -128 to 127, so this is middle
c of the range) :
c
      do 14,band=1,nband,1
        itrans(1,band)=0
   14 continue
c
c -----------------------------------------------------------------------
c multiply the forward and, modified, reverse transformation matrices
c together, to get the full transformation matrix :
c
      do 15,band=1,nband,1
        do 16,bnd=1,nband,1
          dtrans(bnd,band)=0
          do 17,bnd2=1,nband,1
            dtrans(bnd,band)=dtrans(bnd,band)+
     &                        (itrans(bnd2,band)*trans(bnd,bnd2))
   17     continue
   16   continue
   15 continue
c
c -----------------------------------------------------------------------
c normalise it :
c
      do 18,band=1,nband,1
        tot=0
        do 19,bnd=1,nband,1
          tot=tot+abs(dtrans(bnd,band))
   19   continue
        do 20,bnd=1,nband,1
          dtrans(bnd,band)=dtrans(bnd,band)/tot
   20   continue
   18 continue
c
c ------------------------------------------------------------------------
c loop over the image :
c
      do 21,y=1,ysize,1
        do 22,x=1,xsize,1
c
c ------------------------------------------------------------------------
c apply the transformation matrix to a pixel :
c
          do 23,band=1,nband,1
            inten(band)=0
            do 24,bnd=1,nband,1
              itemp=images(x,y,bnd)
              i=max(-127.0,itemp)
              inten(band)=inten(band)+(dtrans(bnd,band)*i)
   24       continue
   23     continue
c
c ------------------------------------------------------------------------
c put the modified pixel values back into the image :
c
          do 25,band=1,nband,1
            images(x,y,band)=nint(inten(band))
   25     continue
c
c ------------------------------------------------------------------------
c go on to the next pixel :
c
   22   continue
   21 continue
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
```

```
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

MATRIX_INV :

```
c
c +------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.  |
c |   work at King's College London, 1988-1992.  This work was funded  |
c |   by the SERC and BP Research International.  This software comes  |
c |   with no guarantees, and you use it at your own risk.  The author |
c |   will probably be prepared to help you out with any problems, if  |
c |   you can track him down.           Philip J. Naylor, July 1992.   |
c +------------------------------------------------------------------+
c
c -------------------------------------------------------------------
c ###################################################################
c -------------------------------------------------------------------
c
      subroutine matrix_inv(matrix,inv_mat,n,np)
c
c *******************************************************************
c *                                                                 *
c *       subroutine to invert a matrix using LU decomposition.     *
c *    [ ludcmp & lubksb are from "Numerical Recipes - the Art of   *
c *       Scientific Computing" by Press, Flannery, Teukolsky,      *
c *           and Vetterling, Cambridge University Press.           *
c *                                                                 *
c *******************************************************************
c
c ===================================================================
c >>>>>>>>>>>>            variable declarations            <<<<<<<<<<<<
c -------------------------------------------------------------------
c integer variables :
c
c i             - loop counter for columns of matrix.
c j             - loop counter for rows of matrix.
c n             - size of matrix (n by n).
c np            - size of array holding matrix (np by np).
c
      integer    i,j,n,np
c
c -------------------------------------------------------------------
c real variables :
c
c d             - the parity of the number of row swaps in decomposition.
c
      real       d
c
c -------------------------------------------------------------------
c real arrays :
```

```
c
c matrix       - the matrix.
c inv_mat      - the inverse of the matrix.
c index        - the row permutation affected by the partial pivot.
c ident        - the identity matrix.
c
      real       matrix(np,np),inv_mat(np,np),index(16),ident(16,16)
c
c ------------------------------------------------------------------------
c ========================================================================
c ########################################################################
c ------------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c stop if the dimensions of the arrays are too small :
c
      if (np. gt. 16) then
        write(*,*) 'ERROR, in MATRIX_INV - INDEX & IDENT are too small.'
        stop
      end if
c
c ------------------------------------------------------------------------
c since the Numerical Recipes routine overwrites the matrix it is given
c with the inverse (and we want to preserve the original) copy matrix
c into inv_mat (and set up the identity matrix, at the same time) :
c
      do 10,j=1,n,1
        do 11,i=1,n,1
          ident(i,j)=0
          inv_mat(i,j)=matrix(i,j)
   11   continue
        ident(j,j)=1
   10 continue
c
c ------------------------------------------------------------------------
c do the LU decomposition :
c
      call ludcmp(inv_mat,n,np,index,d)
c
c ------------------------------------------------------------------------
c do back substitution to get the inverse (returned in ident) :
      do 12,j=1,n,1
        call lubksb(inv_mat,n,np,index,ident(1,j))
   12 continue
c
c ------------------------------------------------------------------------
c copy the inverse to where we want it :
c
      do 13,j=1,n,1
        do 14,i=1,n,1
          inv_mat(i,j)=ident(i,j)
   14   continue
   13 continue
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
c

      return
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
```

FILTDEF :

```
c
c +------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.   |
c |   work at King's College London, 1988-1992.  This work was funded   |
c |   by the SERC and BP Research International.  This software comes   |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if   |
c |   you can track him down.             Philip J. Naylor, July 1992.   |
c +------------------------------------------------------------------+
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
       subroutine filtdef(filter,fsize)
c
c ************************************************************************
c *                                                                      *
c *       a subroutine to define one of a set of normalised, 2-D,        *
c *              circularly symmetric smoothing filters.                 *
c *      [ taken from unattributed Pascal code in another program ]      *
c *                                                                      *
c ************************************************************************
c
c ========================================================================
c >>>>>>>>>>>>            variable declarations             <<<<<<<<<<<<
c ------------------------------------------------------------------------
c
      implicit real (a-z)
c
c ------------------------------------------------------------------------
c integer variables :
c
c ftype         - the type of filter.
c fsize         - the diameter of the filter.
c
      integer    ftype,fsize
c
c ------------------------------------------------------------------------
c real arrays :
c
c filter        - the filter.
c
      real       filter(2500)
c
c ------------------------------------------------------------------------
```

```
c ==========================================================================
c >>>>>>>>>>>>               function declarations           <<<<<<<<<<<<
c --------------------------------------------------------------------------
c integer functions :
c
c intin         - 'eric' library routine for prompting for, and inputting,
c                 an integer.
c
      integer    intin
c
c --------------------------------------------------------------------------
c logical functions :
c
c yes           - 'eric' library routine for asking a question, and
c                 inputting a yes/no response.
c
      logical    yes
c
c --------------------------------------------------------------------------
c ==========================================================================
c ##########################################################################
c --------------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c --------------------------------------------------------------------------
c ==========================================================================
c --------------------------------------------------------------------------
c get the details of the desired filter, from the user :
c
      write(*,*) ' Smoothing filter type :  1 - Canny (Gaussian)'
      write(*,*) '                          2 - Cubic Spline'
      write(*,*) '                          3 - Petrou'
      write(*,*) '                          4 - Spacek'
      ftype=intin('Choice >',1,1,4)
      fsize=intin('Smoothing filter size (square) >',7,1,50)
c
c --------------------------------------------------------------------------
c
      if (ftype .eq. 1) then
c
c{ Generate 2-D Gaussian smoothing filter }
c
        emax = (fsize + 1.0 )/2.0
        iend = int(emax)
        w = emax
        s = -log(0.001)/(w*w)
        tot=0.0
        do 10, i=1,fsize,1
          i1 = i
          i2 = fsize+1-i1
          eye1 = iend-i1
          do 20,j=1,fsize,1
            j1 = j
            j2 = fsize+1-j1
            jay1 = iend-j1
            x = sqrt(eye1*eye1 + jay1*jay1)
            if (x .gt. w) then
              filter(((j1-1)*fsize)+i1) = 0.0
            else
              a = exp(-s*x*x)
              tot=tot+a
              filter(((j1-1)*fsize)+i1) = a
            end if
   20     continue
   10   continue
        do 25,i=1,(fsize*fsize),1
          filter(i)=filter(i)/tot
   25   continue
c
c ------------------------------------------------------------------------
c
      else if (ftype .eq. 2) then
c
c{ Generate 2-D spline smoothing filter }
c
        emax = (fsize + 1.0 )/2.0
        iend = int(emax)
        w = emax
        tot=0.0
        do 30,i=1,fsize,1
          i1 = i
          i2 = fsize+1-i1
          eye1 = iend-i1
          do 40,j=1,fsize,1
            j1 = j
            j2 = fsize+1-j1
            jay1 = iend-j1
            x = sqrt(eye1*eye1 + jay1*jay1)
            if (x .gt. w) then
              filter(((j1-1)*fsize)+i1) = 0.0
            else
              a = -3.0*((x/w)*(x/w)*(x/w)*(x/w)) + 8.0*(x/w)*(x/w)*(x/w)
     &            - 6.0*(x/w)*(x/w) + 1.0
```

```
             tot=tot+a
             filter(((j1-1)*fsize)+i1) = a
           end if
   40    continue
   30  continue
       do 45,i=1,(fsize*fsize),1
         filter(i)=filter(i)/tot
   45  continue
c
c -------------------------------------------------------------------------
c
      else if (ftype .eq. 3) then
c
c{ Generate 2-D optimal smoothing filter (Petrou) }
c
        c1 = 1.316134
        c2 = 0.8223482
        c3 = -0.03200267
        c4 = -0.03552345
        c5 = -0.78682475
        al = 3.16
        aki = -0.5164
        akk = -1.715564
        emax = (fsize + 1.0 )/2.0
        iend = int(emax)
        w = emax
        ak = akk/w
        tot=0.0
        r1 = w
        rw = r1*al
        x1 = -sin(rw/w)-cos(rw/w)
        x2 = c1*exp((-rw)/w) + c4*exp(rw/w)
        x3 = c2*exp((-rw)/w) - c3*exp(rw/w)
        x4 = -sin(rw/w) + cos(rw/w)
        x5 = w*x1*x2/(al*2.0) + w*x4*x3/(al*2.0) - c5*r1+aki*w
        amin = x5*ak
        do 50,i=1,fsize,1
          i1 = i
          i2 = fsize+1-i1
          eye1 = iend-i1
          do 60,j=1,fsize,1
            j1 = j
            j2 = fsize+1-j1
            jay1 = iend-j1
            r1 = sqrt(eye1*eye1 + jay1*jay1)
            rw = r1*al
            if (r1 .gt. w) then
```

```
              filter(((j1-1)*fsize)+i1) = 0.0
            else
              rww = rw/w
              ew = exp(rww)
              sw = sin(rww)
              cw = cos(rww)
              x1 = -sw-cw
              x2 = (c1/ew) + (c4*ew)
              x3 = (c2/ew) - (c3*ew)
              x4 = -sw+cw
              x5 = w*x1*x2/(al*2.0) + w*x4*x3/(al*2.0) - r1*c5 + aki*w
              a = (x5*ak-amin)/(1.0-amin)
              tot=tot+a
              filter(((j1-1)*fsize)+i1) = a
            end if
   60     continue
   50   continue
        do 65,i=1,(fsize*fsize),1
          filter(i)=filter(i)/tot
   65   continue
c
c ----------------------------------------------------------------------
c
      else if (ftype .eq. 4) then
c
c{ Generate 2-D optimal smoothing filter (Spacek) }
c
        c1 = -13.3816
        c2 = 2.7953
        c3 = 0.0542
        c4 = -3.7953
        c5 = 1.0
        al = 1.0
        aki = -9.3961
        akk = 1.776
        emax = (fsize + 1.0 )/2.0
        iend = int(emax)
        w = emax
        ak = akk/w
        tot=0.0
        r1 = w
        rw = r1*al
        x1 = -sin(rw/w)-cos(rw/w)
        x2 = c1*exp(-rw/w) + c4*exp(rw/w)
        x3 = c2*exp(-rw/w) - c3*exp(rw/w)
        x4 = -sin(rw/w) + cos(rw/w)
        x5 = w*x1*x2/(al*2.0) + w*x4*x3/(al*2.0) - c5*r1 + aki*w
        amin = x5*ak
        do 70,i=1,fsize,1
          i1 = i
          i2 = fsize+1-i1
          eye1 = iend-i1
          do 80,j=1,fsize,1
            j1 = j
            j2 = fsize+1-j1
            jay1 = iend-j1
            r1 = sqrt(eye1*eye1 + jay1*jay1)
            rw = r1*al
            if (r1 .gt. w) then
              filter(((j1-1)*fsize)+i1) = 0.0
            else
              rww = rw/w
              ew = exp(rww)
              sw = sin(rww)
              cw = cos(rww)
              x1 = -sw-cw
              x2 = (c1/ew) + (c4*ew)
              x3 = (c2/ew) - (c3*ew)
              x4 = -sw+cw
              x5 = w*x1*x2/(al*2.0) + w*x4*x3/(al*2.0) - r1*c5 + aki*w
              a = (x5*ak-amin)/(1.0-amin)
              tot=tot+a
              filter(((j1-1)*fsize)+i1) = a
            end if
   80     continue
   70   continue
        do 85,i=1,(fsize*fsize),1
          filter(i)=filter(i)/tot
   85   continue
c
c -----------------------------------------------------------------------
c selected filter number is not on the list :
c
      else
        write(*,*) 'Unknown filter type...'
        stop
      end if
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c the end of the subroutine :
c
      return
```

```
c
c -------------------------------------------------------------------------
c #########################################################################
c =========================================================================
c -------------------------------------------------------------------------
c
      end
c
c -------------------------------------------------------------------------
c #########################################################################
c -------------------------------------------------------------------------
c
```

CONVOLVE :

```
c
c +---------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.   |
c |   work at King's College London, 1988-1992.  This work was funded   |
c |   by the SERC and BP Research International.  This software comes   |
c |   with no guarantees, and you use it at your own risk.  The author  |
c |   will probably be prepared to help you out with any problems, if   |
c |   you can track him down.           Philip J. Naylor, July 1992.    |
c +---------------------------------------------------------------------+
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
      subroutine convolve(image,xsize,ysize,nband,filter,fsize)
c
c ***********************************************************************
c *                                                                     *
c *     a subroutine to convolve image bands with a given filter.       *
c *                                                                     *
c ***********************************************************************
c
c =======================================================================
c >>>>>>>>>>>>            variable declarations             <<<<<<<<<<<<
c -----------------------------------------------------------------------
c integer variables :
c
c band          - the specral band number to be filtered.
c fsize         - size of the filter (in both x & y).
c i             - loop counter for stepping through filter elements (x).
c j             - loop counter for stepping through filter elements (y).
c i1            - loop counter for stepping through patch of image (x).
c j1            - loop counter for stepping through patch of image (y).
c itemp         - integer pixel value.
c x             - loop counter for stepping through image (x).
c y             - loop counter for stepping through image (y).
c xsize         - size of image in x direction.
c ysize         - size of image in y direction.
c nband         - number of spectral bands.
c
      integer    xsize,ysize,nband,fsize,x,y,i,j,band,i1,j1,itemp
c
c -----------------------------------------------------------------------
c byte arrays :
c
c image         - the image data.
```

```
c temp         - temporary storage for a single filtered band.
c
      byte       image(xsize,ysize,nband),temp(512,512)
c
c ---------------------------------------------------------------------------
c real variables :
c
c conv         - the local result of filtering a patch of the image.
c
      real       conv
c
c ---------------------------------------------------------------------------
c real arrays :
c
c filter       - the smoothing filter.
c
      real       filter(fsize,fsize)
c
c ---------------------------------------------------------------------------
c ===========================================================================
c ###########################################################################
c ---------------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c ---------------------------------------------------------------------------
c ===========================================================================
c ---------------------------------------------------------------------------
c announce what stage the program has reached :
c
      write(*,*) ' Smoothing the data.'
c
c ---------------------------------------------------------------------------
c loop over all the spectral bands :
c
      do 10,band=1,nband,1
c
c ---------------------------------------------------------------------------
c copy the current band into the array where the result will be stored
c (these means that there is no "black" border when the result is copied
c back into the original array) :
c
        do 1,y=1,ysize,1
          do 2,x=1,xsize,1
            temp(x,y)=image(x,y,band)
    2     continue
    1   continue
c
c ----------------------------------------------------------------------
c loop over the region of the image which does not result in the filter
c going off of the edge :
c
        do 20,y=(fsize/2)+1,ysize-(fsize/2),1
          do 30,x=(fsize/2)+1,xsize-(fsize/2),1
c
c ----------------------------------------------------------------------
c initialise the local result to zero :
c
            conv=0.0
c
c ----------------------------------------------------------------------
c loop over the filter, and the associated patch of image, doing the
c convoloution (n.b. the image coordinates are relative to the centre of
c the filter) :
c
            do 40,j=1,fsize,1
              do 50,i=1,fsize,1
                i1=i-((fsize/2)+1)
                j1=j-((fsize/2)+1)
                itemp=image(x+i1,y+j1,band)
                conv=conv+
     &            ((itemp+128)*filter(i,j))
   50         continue
   40       continue
c
c ----------------------------------------------------------------------
c store the local result in the output array :
c
            temp(x,y)=nint(conv)-128
c
c ----------------------------------------------------------------------
c go on to the next patch position :
c
   30     continue
   20   continue
c
c ----------------------------------------------------------------------
c copy the result back into the original image band :
c
        do 60,y=1,ysize,1
          do 70,x=1,xsize,1
            image(x,y,band)=temp(x,y)
   70     continue
   60   continue
c
c -----------------------------------------------------------------------
c go on to the next spectral band :
c
   10 continue
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c -----------------------------------------------------------------------
c #######################################################################
c =======================================================================
c -----------------------------------------------------------------------
c
      end
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
```

GLCOOC :

```
c
c +------------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.      |
c |   work at King's College London, 1988-1992.  This work was funded      |
c |   by the SERC and BP Research International.  This software comes      |
c |   with no guarantees, and you use it at your own risk.  The author     |
c |   will probably be prepared to help you out with any problems, if      |
c |   you can track him down.           Philip J. Naylor, July 1992.       |
c +------------------------------------------------------------------------+
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
      subroutine glcooc(images,xsize,ysize,nband,numlev,
     &                 classave,classwid,nclass,dist)
c
c   ***************************************************************
c   *                                                             *
c   *      a subroutine to find the peaks in coocurrence space.   *
c   *                                                             *
c   ***************************************************************
c
c ======================================================================
c >>>>>>>>>>>>            variable declarations            <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer variables :
c
c   band         - spectral band number.
c   dist         - the width of border to ignore (to avoid edge effects
c                  of smoothing).
c   nband        - number of spectral bands.
c   nclass       - number of classes.
c   numlev       - number of data bins.
c   peak         - peak number (in a particular band).
c   pk           - alternative to peak.
c   pos          - pointer to an array element.
c   positn       - alternative pointer to an array element.
c   smooth       - standard deviation of the gaussian used (in fourier
c                  space) to smooth the leading diagonal of the
c                  cooccurrence matrix and its first two derivatives.
c   xsize        - image size (x).
c   ysize        - image size (y).
c
      integer    band,nband,nclass,numlev,peak,pk,pos,positn
      integer    xsize,ysize,dist
```

```
c
c ------------------------------------------------------------------------
c integer arrays :
c
c   classave     - class mean vectors.
c   classwid     - class widths (parallel to axes).
c   height       - the heights of the peaks (in a particular band).
c   npeak        - the number of peaks found in each band.
c   peakpos      - the positions of the peaks found in each band.
c   peakwid      - the widths of the peaks found in each band.
c   vector       - a unit vector, describing the cooccurrence direction
c                  (orientation of the cooccurrence filter) :
c                    {0,-1}  =>  vertical (north, 0 degrees).
c                    {1,-1}  =>  diagonal (northeast, 45 degrees).
c                    {1,0}   =>  horizontal (east, 90 degrees).
c                    {1,1}   =>  diagonal (southeast, 135 degrees).
c
      integer    classave(16,800),classwid(16,800),npeak(16)
      integer    peakpos(16,800),peakwid(16,800),vector(2)
c
c ------------------------------------------------------------------------
c byte arrays :
c
c   images       - as in the main program, but redimensioned so as to
c                  make the accessing of the data more obvious.
c
      byte       images(xsize,ysize,nband)
c
c ------------------------------------------------------------------------
c real arrays :
c
c   hist         - the leading diagonal of an infra-band cooccurrence
c                  matrix, after the matrix has been convolved with a
c                  / 1 1 0 \ filter.
c                  | 1 2 1 |
c                  \ 0 1 1 /
c
      real       hist(0:255)
c
c ------------------------------------------------------------------------
c ========================================================================
c ########################################################################
c ------------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
```

```
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c announce what stage the program has reached :
c
      write(*,*) ' Analysing infra-band cooccurrence matrices.'
c
c ----------------------------------------------------------------------
c initialize the number of peaks found, in each band, as zero :
c
      do 10,band=1,nband,1
        npeak(band)=0
   10 continue
c
c ----------------------------------------------------------------------
c step through the bands :
c
      do 20,band=1,nband,1
c
c ----------------------------------------------------------------------
c call the subroutine which forms the leading diagonal of the glcm of
c this band :
c
        call diag(images,band,xsize,ysize,nband,numlev,
     &            dist,hist)
c
c ----------------------------------------------------------------------
c call the subroutine which finds the interesting features (maxima,
c minima, etc.) in the leading diagonal :
c
        call hstanal(hist,numlev,peakpos,peakwid,nband,npeak,band)
c
c ----------------------------------------------------------------------
c sort the peaks into ascending order of position :
c
        call sort(peakpos,peakwid,npeak,band)
c
c ----------------------------------------------------------------------
c if more than one peak was found in this band, go through all the peaks
c and merge together any that are very close together :
c
        if (npeak(band) .gt. 1) then
          positn=1
   32       if ((peakpos(band,(positn+1))-peakpos(band,positn))
     &                                                    .le. 3) then
              peakpos(band,positn)=
     &                 (peakpos(band,positn)+peakpos(band,(positn+1)))/2
```

```
              peakwid(band,positn)=
     &                max(peakwid(band,positn),peakwid(band,(positn+1)))
              do 33,pos=(positn+1),(npeak(band)-1),1
                peakpos(band,pos)=peakpos(band,(pos+1))
                peakwid(band,pos)=peakwid(band,(pos+1))
   33         continue
              npeak(band)=npeak(band)-1
              positn=positn-1
            end if
            positn=positn+1
          if (positn .le. (npeak(band)-1)) goto 32
        end if
c ------------------------------------------------------------------------
c output the number of peaks found, in this band, their locations, and
c their along-axis half-widths :
c
        write(*,'(1x,i2,1x,a19,1x,i3)') npeak(band),
     &                                    'peaks found in band',band
        write(9,'(1x,i2,1x,a19,1x,i3)') npeak(band),
     &                                    'peaks found in band',band
        do 35,peak=1,npeak(band),1
          write(9,'(1x,a10,1x,i3,4x,a7,1x,i3)')
     &         'location :',peakpos(band,peak),
     &         'width :',peakwid(band,peak)
   35   continue
c
c ------------------------------------------------------------------------
c go on to the next spectral band :
c
   20 continue
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c if there is more than one spectral band of data, call the subroutine
c which finds out how the peaks link together across the bands :
c
      if (nband .gt. 1) then
        call findpk(images,xsize,ysize,nband,npeak,peakpos,
     &              peakwid,nclass,classave,classwid)
c
c otherwise, just set the class means equal to the peak positions, the
c class along-axis half-widths equal to the peak along-axis half-widths,
c and the number of class found to the number of peaks found :
c
      else
        do 40,peak=1,npeak(1),1
```

```
          classave(1,peak)=peakpos(1,peak)
          classwid(1,peak)=peakwid(1,peak)
   40   continue
        nclass=npeak(1)
      end if
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ----------------------------------------------------------------------
c ######################################################################
c ======================================================================
c ----------------------------------------------------------------------
c
      end
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

DIAG :

```
c
c +-------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.    |
c |   work at King's College London, 1988-1992.  This work was funded    |
c |   by the SERC and BP Research International.  This software comes    |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if    |
c |   you can track him down.            Philip J. Naylor, July 1992.   |
c +-------------------------------------------------------------------+
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
      subroutine diag(images,band,xsize,ysize,nband,numlev,dist,hist)
c
c   ******************************************************************
c   *                                                                *
c   *       a subroutine to form the diagonal of an infra-band      *
c   *                     coocurrence matrix.                        *
c   *                                                                *
c   ******************************************************************
c
c ======================================================================
c >>>>>>>>>>>>            variable declarations            <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer variables :
c
c   band         - spectral band number.
c   direct       - indicates cooccurrence direction :
c                     1 =>   0 degrees to the vertical (north).
c                     2 =>  45 degrees to the vertical (north).
c                     3 =>  90 degrees to the vertical (north).
c                     4 => 135 degrees to the vertical (north).
c   dist         - the width of image border to ignore.
c   inten1       - the value returned by the first half of the
c                  cooccurrence filter.
c   inten2       - the value returned by the second half of the
c                  cooccurrence filter.
c   itemp        - temporary variable, used to convert byte type
c                  to integer type.
c   nband        - number of spectral bands.
c   numlev       - number of data bins.
c   x            - x coordinate of position in image.
c   xsize        - size of image in x direction.
c   y            - y coordinate of position in image.
```

```
c   ysize        - size of image in y direction.
c
      integer    band,direct,inten1,inten2,itemp,nband,numlev,dist,x
      integer    xsize,y,ysize
c
c -----------------------------------------------------------------------
c integer arrays :
c
c   vector       - cooccurrence direction vector.
c
      integer    vector(2)
c
c -----------------------------------------------------------------------
c byte arrays :
c
c   images       - the image data.
c
      byte       images(xsize,ysize,nband)
c
c -----------------------------------------------------------------------
c real arrays :
c
c   hist         - the diagonal of the cooccurrence matrix (histogram).
c
      real       hist(0:255)
c
c -----------------------------------------------------------------------
c ======================================================================
c ######################################################################
c ----------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c initialize the elements of the diagonal of the cooccurrence matrix to
c zero :
c
      do 5,inten1=0,(numlev-1),1
        hist(inten1)=0
    5 continue
c
c -----------------------------------------------------------------------
c loop over the portion of the image which won’t result in any part of
c the coocurrence filter going off of the edge :
```

```
c
      do 10,y=(1+dist),(ysize-dist),1
        do 20,x=(1+dist),(xsize-dist),1
          itemp=images(x,y,band)
          inten1=itemp+128
c
c -----------------------------------------------------------------
c loop over the cooccurrence directions :
c
          do 1,direct=1,4,1
            vector(1)=min(1.0,direct-1)
            vector(2)=max(-1.0,direct-3)
            itemp=images(x+vector(1),y+vector(2),band)
            inten2=itemp+128
c
c if (inten1,inten2) lies within one greylevel of the leading diagonal,
c then add its effect to the smoothed leading diagonal :
c
            if (inten1 .eq. inten2) then
              hist(inten1)=hist(inten1)+2
              if (inten1 .lt. (numlev-1)) then
                hist(inten1+1)=hist(inten1+1)+1
              else if (inten1 .gt. 0) then
                hist(inten1-1)=hist(inten1-1)+1
              end if
            else if (abs(inten1-inten2) .eq. 1) then
              hist(inten1)=hist(inten1)+1
              hist(inten2)=hist(inten2)+1
            end if
c
c -----------------------------------------------------------------
c go on to the next cooccurrence direction :
c
    1     continue
c -----------------------------------------------------------------
c go on to the next pixel :
c
   20   continue
   10 continue
c
c -----------------------------------------------------------------
c =================================================================
c -----------------------------------------------------------------
c the end of the subroutine :
c
      return
c
```

```
c ----------------------------------------------------------------------
c ######################################################################
c ======================================================================
c ----------------------------------------------------------------------
c
      end
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

HSTANAL :

```
c
c +-------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.   |
c |   work at King's College London, 1988-1992.  This work was funded   |
c |   by the SERC and BP Research International.  This software comes   |
c |   with no guarantees, and you use it at your own risk.  The author  |
c |   will probably be prepared to help you out with any problems, if   |
c |   you can track him down.            Philip J. Naylor, July 1992.   |
c +-------------------------------------------------------------------+
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
      subroutine hstanal(hist,numlev,peakpos,peakwid,nband,npeak,band)
c
c **********************************************************************
c *                                                                    *
c *          a subroutine to find the peaks in a one dimensional       *
c *      feature (sub)space, using an adaptive version of the CLEAN    *
c *                        deconvolution algorithm.                    *
c *                                                                    *
c **********************************************************************
c
c ======================================================================
c >>>>>>>>>>>>              variable declarations           <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer variables :
c
c band          - spectral band number.
c mpos          - position of a maximum.
c nband         - number of spectral bands.
c numlev        - number of data bins.
c offset        - position within the smoothing filter.
c peak          - current peak number.
c pos           - position within the feature space.
c smooth        - half-width of smoothing filter.
c
      integer    numlev,nband,band,pos,mpos,peak,offset,smooth
c
c ----------------------------------------------------------------------
c integer arrays :
c
c npeak         - the number of peaks found in each dimension.
c peakpos       - the positions of the peaks found in each dimension.
c peakwid       - the widths of the peaks found in each dimension
```

```
c                (taken to be five sigma).
c
      integer    peakpos(16,800),peakwid(16,800),npeak(16)
c
c ------------------------------------------------------------------------
c real variables :
c
c amaxi         - the height of the first maximum found (the global max.).
c gauss         - the value of a Gaussian curve at some point.
c height        - height of the current peak (taken to be the same as the
c                 height of the current maximum).
c maxi          - the height of the current maximum.
c stdev         - the standard deviation of the current peak.
c thresh        - threshold at which to stop deconvolution (this occurs
c                 when the height of a peak is less than this fraction
c                 of the height of the first peak found).
c
      real       gauss,maxi,stdev,height,thresh,amaxi
c
c ------------------------------------------------------------------------
c real arrays :
c
c hist          - the 1-D feature space (histogram).
c temp          - temporary storage, used when smoothing the histogram.
c
      real       hist(0:(numlev-1)),temp(0:255)
c
c ------------------------------------------------------------------------
c ========================================================================
c >>>>>>>>>>>>             function declarations            <<<<<<<<<<<<
c ------------------------------------------------------------------------
c real functions :
c
c sigma         - returns the standard deviation of a peak.
c
      real       sigma
c
c ------------------------------------------------------------------------
c ========================================================================
c ########################################################################
c ------------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
```

```
c initialise the number of peaks found :
c
      peak=0
      npeak(band)=0
c
c ------------------------------------------------------------------------
c set the various parameters (n.b. these are "MAGIC NUMBERS", they may not
c be optimal, and it may be better to put them under user control) :
c
      thresh=0.005
      if (nband .eq. 1) thresh=0.05
      smooth=2
c
c ------------------------------------------------------------------------
c smooth the histogram, to reduce the effects of noise :
c
      do 5,pos=smooth,(numlev-smooth-1),1
        temp(pos)=0
        do 6,offset=-1*smooth,smooth,1
          temp(pos)=temp(pos)+(hist(pos+offset)/((2*smooth)+1))
    6   continue
    5 continue
      do 7,pos=smooth,(numlev-smooth-1),1
        hist(pos)=temp(pos)
    7 continue
c
c ------------------------------------------------------------------------
c find the maximum (if this is the first pass, record it for later use) :
c
   40 maxi=-1000000
      do 10,pos=0,(numlev-1),1
        if (hist(pos) .gt. maxi) then
          maxi=hist(pos)
          mpos=pos
          if (peak .eq. 0) amaxi=maxi
        end if
   10 continue
c
c ------------------------------------------------------------------------
c if we have deconvolved down to the threshold (i.e. we are now in the
c noise), stop :
c
      if (maxi .lt. (amaxi*thresh)) goto 30
c
c ------------------------------------------------------------------------
c find the standard deviation of the peak associated with the maximum :
c
```

```
      stdev=sigma(mpos,hist,numlev)
c
c -------------------------------------------------------------------------
c find the amplitude of the peak associated with the maximum :
c
      height=hist(mpos)
c
c -------------------------------------------------------------------------
c subtract out the peak found (setting negative values to zero) :
c
      do 20,pos=0,(numlev-1),1
        gauss=height*exp(-0.5*(((pos-mpos)**2.0)/(stdev**2.0)))
        hist(pos)=max(0,nint(real(hist(pos))-gauss))
   20 continue
c
c -------------------------------------------------------------------------
c ignoring peaks which have standard deviations less than one bin width
c (they are, most likely, due to noise) record details of the peak found :
c
      if (stdev .gt. 1.0) then
        peak=peak+1
        if (peak .gt. 800) then
          write(*,'(1x,a23,i1,a1)') 'Too many peaks in band ',band,'.'
          stop
        end if
        peakpos(band,peak)=mpos
        peakwid(band,peak)=nint(5.0*stdev)
      end if
c
c -------------------------------------------------------------------------
c repeat for the next peak :
c
      goto 40
c
c -------------------------------------------------------------------------
c now we have finished, record the total number of peaks found in this
c dimension :
c
   30 npeak(band)=peak
c
c -------------------------------------------------------------------------
c =========================================================================
c -------------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
```

```
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
```

SIGMA :

```
c
c +---------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.   |
c |   work at King's College London, 1988-1992.  This work was funded   |
c |   by the SERC and BP Research International.  This software comes   |
c |   with no guarantees, and you use it at your own risk.  The author  |
c |   will probably be prepared to help you out with any problems, if   |
c |   you can track him down.           Philip J. Naylor, July 1992.    |
c +---------------------------------------------------------------------+
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
      real function sigma(mpos,hist,numlev)
c
c  ******************************************************************
c  *                                                                *
c  *       a function to estimate the standard deviation of         *
c  *      a peak, by trying to fit a gaussian function to it.       *
c  *                                                                *
c  ******************************************************************
c
c ======================================================================
c >>>>>>>>>>>>            variable declarations             <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer variables :
c
c   band         - spectral band number.
c   maxi         - the upper limit of the range over which the gaussian
c                  is to be fitted.
c   mean         - the position of the centre of the gaussian.
c   mini         - the lower limit to the range over which the gaussian
c                  is to be fitted.
c   pos          - variable which the gaussian is a function of.
c
      integer    band,maxi,mean,mini,pos,mpos
c
c ----------------------------------------------------------------------
c real variables :
c
c   diff         - the difference (in area) between the gaussian and the
c                  peak.
c   gauss        - the value taken by the gaussian function, for a
c                  particular value of pos.
c   oldst        - the previous estimate of the standard deviation.
```

```
c   oldst2       - the estimate of the standard deviation before the
c                  previous one.
c
      real       diff,gauss,oldst,oldst2
c
c ----------------------------------------------------------------------
c real arrays :
c
c   hist         - the histogram being analysed.
c
      real       hist(0:(numlev-1))
c
c ----------------------------------------------------------------------
c ======================================================================
c ######################################################################
c ----------------------------------------------------------------------
c
c
c the start of the function proper :
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c initialize the estimates of the standard deviation :
c
      oldst=0
      oldst2=0
c
c start with a very narrow gaussian as the initial estimate.
c
      sigma=1.0
c
c ----------------------------------------------------------------------
c centre the gaussian on the peak position :
c
      mean=mpos
c
c ----------------------------------------------------------------------
c the gaussian is to be fitted to a range of 21 points over the centre
c of the peak (provided this range does not run off either end of the
c leading diagonal) :
c
      mini=max(0,(mean-10))
      maxi=min((mean+10),(numlev-1))
c
c ----------------------------------------------------------------------
c initialize the difference, between the gaussian and the peak, to
```

```
c zero :
c
   30 diff=0
c
c ------------------------------------------------------------------------
c loop over the selected range, calculating the difference
c between the area under the gaussian and the area under the peak :
c
        do 10,pos=mini,maxi,1
          gauss=real(hist(mpos))*exp(-0.5*(real((pos-mean)**2)
     &          /real(sigma**2.0)))
          diff=diff+gauss-hist(pos)
   10   continue
c
c ------------------------------------------------------------------------
c if the difference is greater than zero, the gaussian must be too wide,
c so reduce sigma :
c
        if (diff .gt. 0.0) sigma=sigma*0.9
c
c ------------------------------------------------------------------------
c if the difference is less than zero, the gaussian must be too narrow,
c so increase sigma :
c
        if (diff .lt. 0.0) sigma=sigma*1.1
c
c ------------------------------------------------------------------------
c if diff has not converged, or seriously diverged, record this value
c of diff, and the previous one, and try another iteration :
c
        if ((abs(sigma-oldst2) .ge. 0.2) .and. (sigma .lt. 25)) then
          oldst2=oldst
          oldst=sigma
          goto 30
        end if
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c the end of the function :
c
      return
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
```

```
c
      end
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
```

SORT :

```
c
c +-----------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.      |
c |   work at King's College London, 1988-1992.  This work was funded      |
c |   by the SERC and BP Research International.  This software comes      |
c |   with no guarantees, and you use it at your own risk.  The author     |
c |   will probably be prepared to help you out with any problems, if      |
c |   you can track him down.            Philip J. Naylor, July 1992.      |
c +-----------------------------------------------------------------------+
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
      subroutine sort(peakpos,peakwid,npeak,band)
c
c ***********************************************************************
c *                                                                     *
c *    a subroutine to sort the columns of one row of two arrays        *
c *          with respect to the columns of the first one.              *
c *                        (ascending order)                            *
c *                          [NOT OPTIMAL]                              *
c *                                                                     *
c ***********************************************************************
c
c =======================================================================
c >>>>>>>>>>>>            variable declarations            <<<<<<<<<<<<
c =======================================================================
c integer variables :
c
c band           - the row number
c first          - the smallest value in the unsorted part
c fst            - the position of the smallest value in the unsorted
c                  part
c pos            - loop counter for going through array
c start          - start of unsorted portion of array
c temp1          - temporary storage for swapping elements of first
c                  array
c temp2          - temporary storage for swapping elements of second
c                  array
c
      integer    band,pos,temp1,temp2,first,start,fst
c
c -----------------------------------------------------------------------
c integer arrays :
c
```

```
c npeak          - the number of elements in each row
c peakpos        - the first (index) array
c peakwid        - the second array
c
      integer    peakpos(16,800),peakwid(16,800),npeak(16)
c
c ------------------------------------------------------------------------
c ========================================================================
c ########################################################################
c ------------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c start off assuming the whole array is unsorted :
c
      start=1
c
c ------------------------------------------------------------------------
c find the minimum element in the unsorted part of the array :
c
   20 first=1000000
      do 10,pos=start,npeak(band),1
        if (peakpos(band,pos) .lt. first) then
          first=peakpos(band,pos)
          fst=pos
        end if
   10 continue
c
c ------------------------------------------------------------------------
c swap the array elements over, so that the minimum found in the unsorted
c part becomes the last element in the sorted part :
c
      temp1=peakpos(band,start)
      temp2=peakwid(band,start)
      peakpos(band,start)=peakpos(band,fst)
      peakwid(band,start)=peakwid(band,fst)
      peakpos(band,fst)=temp1
      peakwid(band,fst)=temp2
c
c ------------------------------------------------------------------------
c one more element has been placed in the right order :
c
      start=start+1
```

```
c
c ------------------------------------------------------------------------
c if there are any more elements to sort, go back and do them :
c
      if (start .lt. npeak(band)) goto 20
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
```

FINDPK :

```
c
c +-----------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.    |
c |   work at King's College London, 1988-1992.  This work was funded    |
c |   by the SERC and BP Research International.  This software comes    |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if    |
c |   you can track him down.            Philip J. Naylor, July 1992.   |
c +-----------------------------------------------------------------------+
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
      subroutine findpk(images,xsize,ysize,nband,npeak,peakpos,
     &                  peakwid,nclass,classave,classwid)
c
c   ***************************************************************
c   *                                                             *
c   *         a subroutine to associate, across all the bands,    *
c   *       the peaks found in single band cooccurrence matrices. *
c   *     [ jacobi & eigsrt are from "Numerical Recipes - the Art *
c   *          of Scientific Computing" by Press, Flannery,       *
c   *                 Teukolsky, and Vetterling,                  *
c   *                Cambridge University Press.                  *
c   *                                                             *
c   ***************************************************************
c
c ======================================================================
c >>>>>>>>>>>>            variable declarations            <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer variables :
c
c   band         - spectral band number.
c   bigc         - class number of the "King of the Castle".
c   bnd          - spectral band number.
c   class        - class number.
c   class2       - class number.
c   classif      - peak combination number, e.g. :
c                    {peak 1 in band 1, peak 1 in band 2,..., peak 1 in
c                  band n}  => classif = 1,
c                    {peak 2 in band 1, peak 1 in band 2,..., peak 1 in
c                  band n}  => classif = 2,
c                    etc..
c   count        - used to loop through the possible values of classif.
c   dis          - the euclidean distance (in 1-dimensional histogram
c                    space) between the point associated with a particular
c                    pixel and a particular peak (in a particular band).
c   itemp          - temporary variable, used to convert from byte type to
c                    integer type.
c   limit          - since posclass (see below) could be a very large
c                    number, the possible classes (values of classif) are
c                    considered in blocks of 1000000 at a time.  limit
c                    indicates the value of classif at the start of the
c                    block currently under consideration.
c   min            - the euclidean distance (in 1-dimensional histogram
c                    space) between the point associated with a particular
c                    pixel and the nearest peak (in a particular band).
c   nband          - number of bands.
c   nc             - class number.
c   nclass         - number of classes.
c   near           - the number of the peak nearest to point associated
c                    with a particular pixel (in 1-dimensional histogram
c                    space, in a particular band).
c   nrot           - number of rotations used by Numerical Recipes jacobi
c                    r
c   posclass       - the number of possible classes, given the number of
c                    peaks found in each band.
c   pk             - peak number (in a particular band).
c   small          - the smallest number of pixels which will be
c                    considered as constituting a cluster.
c   x              - loop counter for going through image (x).
c   xsize          - image size (x).
c   y              - loop counter for going through image (y).
c   ysize          - image size (y).
c
      integer    band,bnd,class,classif,count,dis,limit,min,nband,nclass
      integer    near,posclass,pk,small,x,xsize,y,ysize
      integer    class2,nc,bigc,nrot,itemp
c
c -------------------------------------------------------------------
c integer arrays :
c
c   classave       - class mean vectors.
c   classave2      - class mean vectors.
c   classwid       - class widths.
c   classwid2      - class widths.
c   npeak          - number of peaks in 1-D sub-spaces.
c   peak           - the set of peak numbers with which a particular pixel
c                    is associated.
c   peakpos        - positions of peaks in 1-D sub-spaces.
c   peakwid        - widths of peaks in 1-D sub-spaces.
c   prod           - used in calculating classif.
```

```
c
      integer    classave(16,800),classwid(16,800),classave2(16,800)
      integer    npeak(16),peak(16),peakpos(16,800),peakwid(16,800)
      integer    prod(16),classwid2(16,800)
c
c -------------------------------------------------------------------
c 2 byte integer arrays :
c
c   number       - the number of pixels in a possible class.
c
      integer*2  number(1000000)
c
c -------------------------------------------------------------------
c byte arrays :
c
c   images       - image data.
c
      byte       images(xsize,ysize,nband)
c
c -------------------------------------------------------------------
c real variables :
c
c   d            - Euclidean distance from a feature vector to a cluster
c                  centre.
c   div          - the transformed divergence between two classes.
c   div1         - the Euclidean distance, in one dimension, between a
c                  feature vector and a cluster centre.
c   div2         - the Euclidean distance, in one dimension, between a
c                  feature vector and a cluster centre.
c   mcs          - maximum class size.
c   md           - the minimum Euclidean distance from a feature vector
c                  to a cluster centre (with respect to the cluster
c                  number).
c   thresh       - the threshold on the transformed divergence, which
c                  defines the "neighbourhood" of a cluster.
c
      real       d,md,div,thresh,div1,div2,mcs
c
c -------------------------------------------------------------------
c real arrays :
c
c   classsiz     - cluster sizes.
c   covar        - cluster covariance matrix.
c   covar2       - cluster covariance matrix.
c   diagn        - diagonal matrix (used in matrix inversion).
c   eval         - eigenvalues of a matrix (used in matrix inversion).
c   evec         - eigenvectors of a matrix (used in matrix inversion).
```

```
c   icovar       - cluster inverse covariance matrix.
c   icovar2      - cluster inverse covariance matrix.
c
      real       covar(16,16,800),icovar(16,16,800),eval(16),evec(16,16)
      real       diagn(16,16),classsiz(800),covar2(16,16,800)
      real       icovar2(16,16,800)
c
c ----------------------------------------------------------------------
c ======================================================================
c ######################################################################
c ----------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c announce what stage the program has reached :
c
      write(*,*) ' Linking the peaks.'
c
c ----------------------------------------------------------------------
c determine a lower limit on the size of cluster that will be considered
c to constitute a class, 0.025% of the image, or 200 pixels, whichever is
c the larger, (n.b. this is a "MAGIC NUMBER", it is not necessarily
c optimal, and may need to be under user control) :
c
      small=(xsize*ysize)/4000
      if (small .lt. 200) small=200
      small=small-32768
c
c ** n.b. -32768 is the smallest number which can be stored in a **
c **  two byte variable, and in this routine is used as 'zero'.  **
c
c ----------------------------------------------------------------------
c calculate the maximum possible number of classes, given the number of
c peaks found in each band :
c
      posclass=1
      do 10,band=1,nband,1
        posclass=posclass*npeak(band)
   10 continue
c
c ----------------------------------------------------------------------
c initialize the number of classes found to zero :
c
```

```
      class=0
c
c ------------------------------------------------------------------
c because of memory restrictions, the possible classes can only be
c checked out so many at a time (in this case 1000000 at a time) :
c
      do 20,limit=1,posclass,1000000
c
c ------------------------------------------------------------------
c initialize the number of pixels in each possible class under
c consideration to 'zero' (-32768) :
c
        do 30,classif=1,1000000,1
          number(classif)=-32768
   30   continue
c
c ------------------------------------------------------------------
c step through the image :
c
        do 40,y=1,ysize,1
          do 50,x=1,xsize,1
c
c ------------------------------------------------------------------
c step through the bands :
c
            do 60,band=1,nband,1
c
c ------------------------------------------------------------------
c find out which peak (in the current band) this pixel is closest to
c (in 1-dimensional histogram space, using euclidean distance) :
c
              min=1000000
              do 70,pk=1,npeak(band),1
                dis=abs(images(x,y,band)+128-peakpos(band,pk))
                if (dis .lt. min) then
                  min=dis
                  near=pk
                end if
   70         continue
c
c partial calculation of the peak combination number for the set of
c peaks that this pixel is closest to :
c
              prod(band)=1
              do 80,bnd=1,(band-1),1
                prod(band)=prod(band)*npeak(bnd)
   80         continue
```

```
              if (band .eq. 1) then
                classif=near
              else
                classif=classif+(prod(band)*(near-1))
              end if
c
c ---------------------------------------------------------------------
c go on to the next spectral band :
c
   60       continue
c
c ---------------------------------------------------------------------
c if the peak combination number is one of the ones under consideration
c at the moment, and the number of pixels in the corresponding class yet
c reached the lower limit, add 1 to the number of pixels in that class
c (otherwise go on to the next pixel, or the next block of possible
c classes, as appropriate) :
c
            if ((classif .lt. limit) .or.
     &                              (classif .gt. (limit+999999))) then
              goto 50
            else
              if (number(classif-limit+1) .lt. small) then
                number(classif-limit+1)=number(classif-limit+1)+1
              end if
            end if
c
c ---------------------------------------------------------------------
c go on to the next pixel :
c
   50     continue
   40   continue
c
c ---------------------------------------------------------------------
c step through the classes under consideration :
c
        do 90,count=limit,(limit+999999),1
c
c ---------------------------------------------------------------------
c set classif equal to the peak combination number :
c
          classif=count
c
c if the number of pixels in this class has reached the lower limit,
c a new class has been found :
c
          if (number(classif-limit+1) .ge. small) then
```

```
            class=class+1
c
c if this class takes the number of class found over 100, the program
c will not be able to cope (due to the dimensioning of the arrays), so
c stop :
c
            if (class .gt. 800) then
              write(*,*) 'too many clusters, (locating clusters).'
              stop
            end if
c
c ------------------------------------------------------------------------
c step through the bands, in reverse order :
c
            do 100,band=nband,1,-1
c
c ------------------------------------------------------------------------
c decode the peak combination number, to get the relevent set of peak
c numbers :
c
              prod(band)=1
              do 110,bnd=1,(band-1),1
                prod(band)=prod(band)*npeak(bnd)
  110         continue
              peak(band)=1+(classif/prod(band))
              if (peak(band) .eq. (npeak(band)+1)) then
                peak(band)=npeak(band)
                if (band .eq. 1) peak(band)=peak(band)+1
              else if ((band .eq. 1) .and. (peak(band) .eq. 1)) then
                peak(band)=npeak(band)+1
                peak(band+1)=peak(band+1)-1
                do 112,bnd=2,(nband-1),1
                  if (peak(bnd) .eq. 0) then
                    peak(bnd)=npeak(bnd)
                    peak(bnd+1)=peak(bnd+1)-1
                  end if
  112           continue
              end if
              if (band .eq. 1) then
                peak(band)=peak(band)-1
              else
                classif=count
                do 115,bnd=nband,band,-1
                  classif=classif-(prod(bnd)*(peak(bnd)-1))
  115           continue
              end if
c
```

```
c -----------------------------------------------------------------------
c go on to the next spectral band :
c
 100      continue
c
c -----------------------------------------------------------------------
c set the elements of the class mean vector to the positons of the set
c of peaks, and the multi-dimensional along-axis cluster half-widths to
c be the along-axis half-widths of the set of peaks :
c
          do 120,band=1,nband,1
            classave(band,class)=peakpos(band,peak(band))
            classwid(band,class)=peakwid(band,peak(band))
 120      continue
        end if
c
c -----------------------------------------------------------------------
c go on to the next possible class :
c
  90    continue
c
c -----------------------------------------------------------------------
c go on to the next block of possible classes :
c
  20  continue
c
c -----------------------------------------------------------------------
c set nclass to the number of classes found :
c
      nclass=class
c
c -----------------------------------------------------------------------
c write the number of classes found to the screen, and the log file :
c
      write(*,*) nclass,' clusters found in first stage.'
      write(9,*) nclass,' clusters found in first stage.'
c
c -----------------------------------------------------------------------
c initialise class sizes, and new class means and widths :
c
      do 285,class=1,nclass,1
        classsiz(class)=0.0
        do 305,band=1,nband,1
          classave2(band,class)=0
          classwid2(band,class)=0
 305    continue
 285  continue
```

```
c
c ---------------------------------------------------------------------
c loop over the image :
c
      do 282,y=1,ysize,1
        do 283,x=1,xsize,1
c
c ---------------------------------------------------------------------
c classify all feature vectors which are within a Euclidean distance of
c ten of the current set of cluster centres :
c
          md=1000000.0
          do 281,class=1,nclass,1
            d=0.0
            do 284,band=1,nband,1
              itemp=images(x,y,band)
              d=d+(classave(band,class)-(itemp+128.0))**2.0
  284       continue
            d=sqrt(d)
            if (d .lt. 10) then
              md=d
              nc=class
              classsiz(nc)=classsiz(nc)+1.0
              do 301,band=1,nband,1
                itemp=images(x,y,band)
                classave2(band,nc)=classave2(band,nc)+(itemp+128.0)
  301         continue
            end if
  281     continue
c
c ---------------------------------------------------------------------
c go on to the next pixel :
c
  283   continue
  282 continue
c
c ---------------------------------------------------------------------
c divide by class sizes :
c
      do 302,class=1,nclass,1
        if (classsiz(class) .gt. 0) then
          do 303,band=1,nband,1
            classave(band,class)=nint(real(classave2(band,class))
     &                                  /real(classsiz(class)))
  303     continue
        end if
  302 continue
```

```
c
c ----------------------------------------------------------------------
c reinitialise the class sizes :
c
      do 1482,class=1,nclass,1
        classsiz(class)=0.0
 1482 continue
c
c ----------------------------------------------------------------------
c loop over the image :
c
      do 482,y=1,ysize,1
        do 483,x=1,xsize,1
c
c ----------------------------------------------------------------------
c calculate the new class widths, and the cluster covariance matrices,
c based on the new cluster means (again, based on only those feature
c vectors which are within ten of the cluster centres) :
c
          md=1000000.0
          do 481,class=1,nclass,1
            d=0.0
            do 484,band=1,nband,1
              itemp=images(x,y,band)
              d=d+(classave(band,class)-(itemp+128.0))**2.0
  484       continue
            d=sqrt(d)
            if (d .lt. 10) then
              md=d
              nc=class
              classsiz(nc)=classsiz(nc)+1.0
              do 1501,band=1,nband,1
                itemp=images(x,y,band)
                classwid2(band,nc)=classwid2(band,nc)+abs((itemp+128.0)
     &                                        -classave(band,nc))
                do 1502,bnd=band,nband,1
                  itemp=images(x,y,band)
                  div1=(itemp+128)-classave(band,class)
                  itemp=images(x,y,bnd)
                  div2=(itemp+128)-classave(bnd,class)
                  covar(band,bnd,class)=covar(band,bnd,class)
     &                                               +(div1*div2)
 1502           continue
 1501         continue
            end if
  481     continue
  483   continue
  482 continue
c
c ---------------------------------------------------------------------
c divide by class sizes :
c
      do 502,class=1,nclass,1
        if (classsiz(class) .gt. 0) then
          do 503,band=1,nband,1
            classwid(band,class)=max(1,
     &                    nint((5.0*real(classwid2(band,class)))
     &                            /classsiz(class)))
          do 504,bnd=band,nband,1
            covar(band,bnd,class)=covar(band,bnd,class)
     &                                    /classsiz(class)
            covar(bnd,band,class)=covar(band,bnd,class)
  504     continue
  503     continue
        end if
  502 continue
c
c ---------------------------------------------------------------------
c loop over the classes :
c
      do 507,class=1,nclass,1
c
c ---------------------------------------------------------------------
c copy the covariance matrices :
c
        do 803,band=1,nband,1
          do 804,bnd=1,nband,1
            covar2(bnd,band,class)=covar(bnd,band,class)
  804     continue
  803   continue
c
c ---------------------------------------------------------------------
c invert the covariance matrices, using eigenvectors and eigenvalues,
c this allows a fiddle which is necessary because of the nature of the
c clusters (having removed the lighting effects, the clusters are
c roughly "flat" in one direction, so it is likely that one of the
c eigenvalues will be zero, and the matrix uninvertable, but, since we
c are using a discrete space, we can replace any zero eigenvectors with
c very small ones) :
c
c find, and sort, the eigenvectors and eigenvalues, using "Numerical
c Recipes" routines :
c
        call jacobi(covar(1,1,class),nband,16,eval,evec,nrot)
```

```
        call eigsrt(eval,evec,nband,16)
c
c calculate the diagonal matrix for doing the inversion :
c
        do 505,band=1,nband,1
          do 506,bnd=1,nband,1
            diagn(bnd,band)=0
  506     continue
          if (eval(band) .gt. 0) then
            diagn(band,band)=1.0/eval(band)
          else
            diagn(band,band)=1.0e+16
          end if
  505   continue
c
c calculate the inverse matrix :
c
        do 509,band=1,nband,1
          do 510,bnd=1,nband,1
            icovar(bnd,band,class)=0
            do 511,nc=1,nband,1
              icovar(bnd,band,class)=icovar(bnd,band,class)+
     &                               (diagn(bnd,nc)*evec(band,nc))
  511       continue
  510     continue
  509   continue
        do 512,band=1,nband,1
          do 513,bnd=1,nband,1
            diagn(bnd,band)=0
            do 514,nc=1,nband,1
              diagn(bnd,band)=diagn(bnd,band)+
     &                       (evec(bnd,nc)*icovar(nc,band,class))
  514       continue
  513     continue
  512   continue
        do 515,band=1,nband,1
          do 516,bnd=1,nband,1
            icovar(bnd,band,class)=diagn(bnd,band)
  516     continue
  515   continue
c
c ------------------------------------------------------------------
c copy the covariance matrix back (original has been destroyed during
c inversion) :
c
        do 9507,band=1,nband,1
          do 9508,bnd=1,nband,1
```

```
            covar(bnd,band,class)=covar2(bnd,band,class)
 9508     continue
 9507   continue
c
c ----------------------------------------------------------------------
c go on to the next class :
c
  507 continue
c
c ----------------------------------------------------------------------
c sort the class details into size order :
c
      call sort4(classsiz,classave,classwid,covar,icovar,
     &           nclass,nband)
c
c ----------------------------------------------------------------------
c set the threshold which defines the neighbourhood of a cluster
c (n.b. this is a "MAGIC NUMBER", it is not necessarily optimal, and may
c be put under user control) :
c
      thresh=1.7
c
c ----------------------------------------------------------------------
c go through the possible permutations of pairs of classes :
c
      do 287,class=1,nclass,1
        mcs=0.0
        do 288,nc=1,nclass,1
c
c ----------------------------------------------------------------------
c calculate the divergence between the two clusters :
c
          div=0.0
          do 289,band=1,nband,1
            do 290,bnd=1,nband,1
              div=div+(0.5*((covar(band,bnd,class)-
     &                      covar(band,bnd,nc))*
     &                 (icovar(bnd,band,nc)-icovar(bnd,band,class))))
     &               +(0.5*((icovar(band,bnd,class)+
     &                      icovar(band,bnd,nc))*
     &                 ((classave(bnd,class)-classave(bnd,nc))*
     &                  (classave(band,class)-classave(band,nc)))))
  290       continue
  289     continue
c
c ----------------------------------------------------------------------
c change to transformed divergence (with a fudge if the divergence so
c how comes out negative [it shouldn't]) :
c
          if (div .ge. 0) then
            div=2.0*(1.0-exp(-0.125*div))
          else
            write(*,*) 'class ',class,nc,' divergence is negative !',div
            div=2.0*(1.0-exp(0.125*div))
          end if
c
c ---------------------------------------------------------------------
c if the classes are not in the same neighbourhood, go to the next pair
c of classes :
c
          if (div .ge. thresh) goto 288
c
c ---------------------------------------------------------------------
c if the two classes are in the same neighbourhood, record whether the
c second is the largest in the neighbourhood of the first :
c
          if (classsiz(nc) .gt. mcs) then
            mcs=classsiz(nc)
            bigc=nc
          end if
c
c ---------------------------------------------------------------------
c go on to the next second cluster :
c
  288   continue
c
c ---------------------------------------------------------------------
c overwrite the details of the first cluster with those of the largest
c one in its neighbourhood (this may be itself) :
c
        do 292,band=1,nband,1
          classave(band,class)=classave(band,bigc)
          classwid(band,class)=classwid(band,bigc)
          do 293,bnd=1,nband,1
            covar(bnd,band,class)=covar(bnd,band,bigc)
            icovar(bnd,band,class)=icovar(bnd,band,bigc)
  293     continue
  292   continue
c
c ---------------------------------------------------------------------
c go on to the next first class :
c
  287 continue
c
```

```
c -----------------------------------------------------------------------
c go through the list of clusters and remove any duplicate entries :
c
      count=0
      do 294,class=1,nclass,1
        do 295,class2=1,count,1
          do 296,band=1,nband,1
            if (classave(band,class) .ne. classave2(band,class2))
     &                                                         goto 295
  296     continue
          goto 294
  295   continue
        count=count+1
        do 297,band=1,nband,1
          classave2(band,count)=classave(band,class)
          classwid2(band,count)=classwid(band,class)
  297   continue
  294 continue
      do 1294,class=1,count,1
        do 1295,band=1,nband,1
          classave(band,class)=classave2(band,class)
          classwid(band,class)=classwid2(band,class)
 1295   continue
 1294 continue
c
c -----------------------------------------------------------------------
c reset the number of classes :
c
      nclass=count
c
c -----------------------------------------------------------------------
c reinitialise the class sizes :
c
      do 6482,class=1,nclass,1
        classsiz(class)=0.0
 6482 continue
c
c -----------------------------------------------------------------------
c loop over the image :
c
      do 5482,y=1,ysize,1
        do 5483,x=1,xsize,1
c
c -----------------------------------------------------------------------
c recalculate the class means and widths by classifying on the basis
c of a non-Eucliden (class width modified) distance measure :
c
```

```
          md=1000000.0
          do 5481,class=1,nclass,1
            d=0.0
            do 5484,band=1,nband,1
              itemp=images(x,y,band)
              d=d+(((classave(band,class)-(itemp+128.0))**2.0)
     &             /(classwid(band,class)**2.0))
 5484       continue
            if (d .lt. md) then
              md=d
              nc=class
            end if
 5481     continue
          classsiz(nc)=classsiz(nc)+1.0
          do 6501,band=1,nband,1
            itemp=images(x,y,band)
            classwid2(band,nc)=classwid2(band,nc)+
     &                         abs((itemp+128.0)-classave(band,nc))
 6501     continue
c
c -----------------------------------------------------------------------
c go on to the next pixel :
c
 5483   continue
 5482 continue
c
c -----------------------------------------------------------------------
c divide by the class sizes :
c
      do 5502,class=1,nclass,1
        do 5503,band=1,nband,1
          classwid(band,class)=nint((7.5*real(classwid2(band,class)))
     &                             /real(classsiz(class)))
 5503   continue
 5502 continue
c
c -----------------------------------------------------------------------
c sort the class details into size order :
c
      call sort3(classsiz,classave,classwid,nclass,nband)
c
c -----------------------------------------------------------------------
c output the number of classes found, and send their mean vectors to the
c log file :
c
 1000 write(*,'(1x,i3,1x,a42)') nclass,'classes left, after discarding n
     &on-maxima.'
```

```
      do 130,class=1,nclass,1
        write(9,'(1x,i3,a1,1x,32i5)') class,'>',
     &                              (classave(band,class),band=1,nband)
        write(9,'(6x,32i5)') (classwid(band,class),band=1,nband)
  130 continue
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ----------------------------------------------------------------------
c ######################################################################
c ======================================================================
c ----------------------------------------------------------------------
c
      end
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

SORT3 :

```
c
c +-------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.   |
c |   work at King's College London, 1988-1992.  This work was funded   |
c |   by the SERC and BP Research International.  This software comes   |
c |   with no guarantees, and you use it at your own risk.  The author  |
c |   will probably be prepared to help you out with any problems, if   |
c |   you can track him down.            Philip J. Naylor, July 1992.   |
c +-------------------------------------------------------------------+
c
c ---------------------------------------------------------------------
c #####################################################################
c ---------------------------------------------------------------------
c
      subroutine sort3(array1,array2,array3,nelem,nband)
c
c *********************************************************************
c *                                                                   *
c *    a subroutine to sort the rows of two arrays with respect to    *
c *                  the elements of a another                        *
c *                      (descending order)                           *
c *                        [NOT OPTIMAL]                              *
c *                                                                   *
c *********************************************************************
c
c =====================================================================
c >>>>>>>>>>>>           variable declarations            <<<<<<<<<<<<
c =====================================================================
c integer variables :
c
c band          - loop counter for going through the rows
c first         - the largest value in the unsorted part
c fst           - the position of the largest value in the unsorted
c                 part
c nband         - the number of rows in the arrays
c nelem         - the number of elements in the arrays
c pos           - loop counter for going through array
c start         - start of unsorted portion of array
c temp2         - temporary storage for swapping elements of the second
c                 array
c temp3         - temporary storage for swapping elements of the third
c                 array
c
      integer    nband,pos,temp2,first,start,fst,nelem,band,temp3
c
c ---------------------------------------------------------------------
```

```
c integer arrays :
c
c array2         - the second array
c array3         - the third array
c
      integer    array2(16,800),array3(16,800)
c
c -----------------------------------------------------------------------
c real variables :
c
c temp1          - temporary storage for swapping elements of the first
c                  array
c
      real       temp1
c
c -----------------------------------------------------------------------
c real arrays :
c
c array1         - the first (index) array
c
      real       array1(800)
c
c -----------------------------------------------------------------------
c =======================================================================
c #######################################################################
c -----------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c start off assuming the whole array is unsorted :
c
      start=1
c
c -----------------------------------------------------------------------
c find the minimum element in the unsorted part of the array :
c
   20 first=0
      do 10,pos=start,nelem,1
        if (array1(pos) .gt. first) then
          first=array1(pos)
          fst=pos
        end if
   10 continue
```

```
c
c -----------------------------------------------------------------------
c swap the array elements over, so that the minimum found in the unsorted
c part becomes the last element in the sorted part :
c
      temp1=array1(start)
      array1(start)=array1(fst)
      array1(fst)=temp1
      do 30,band=1,nband,1
        temp2=array2(band,start)
        array2(band,start)=array2(band,fst)
        array2(band,fst)=temp2
        temp3=array3(band,start)
        array3(band,start)=array3(band,fst)
        array3(band,fst)=temp3
 30   continue
c
c -----------------------------------------------------------------------
c one more element has been placed in the right order :
c
      start=start+1
c
c -----------------------------------------------------------------------
c if there are any more elements to sort, go back and do them :
c
      if (start .lt. nelem) goto 20
c
c -----------------------------------------------------------------------
c #######################################################################
c =======================================================================
c -----------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c
      end
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
```

SORT4 :

```
c
c +----------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.    |
c |   work at King's College London, 1988-1992.  This work was funded    |
c |   by the SERC and BP Research International.  This software comes    |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if    |
c |   you can track him down.             Philip J. Naylor, July 1992.   |
c +----------------------------------------------------------------------+
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
      subroutine sort4(array1,array2,array3,array4,array5,nelem,nband)
c
c ************************************************************************
c *                                                                      *
c *       a subroutine to sort parts of four arrays with respect to      *
c *                   the elements of a another                          *
c *                       (descending order)                             *
c *                         [NOT OPTIMAL]                                *
c *                                                                      *
c ************************************************************************
c
c ========================================================================
c >>>>>>>>>>>>            variable declarations               <<<<<<<<<<<<
c ========================================================================
c integer variables :
c
c band           - loop counter for going through the rows
c bnd            - loop counter for going through another dimension
c first          - the largest value in the unsorted part
c fst            - the position of the largest value in the unsorted
c                  part
c nband          - the number of rows in the arrays
c nelem          - the number of elements in the arrays
c pos            - loop counter for going through array
c start          - start of unsorted portion of array
c temp2          - temporary storage for swapping elements of the second
c                  array
c temp3          - temporary storage for swapping elements of the third
c                  array
c
      integer    nband,pos,temp2,first,start,fst,nelem,band,temp3,bnd
c
```

```
c -------------------------------------------------------------------------
c integer arrays :
c
c array2         - the second array
c array3         - the third array
c
      integer    array2(16,800),array3(16,800)
c
c -------------------------------------------------------------------------
c real variables :
c
c temp1          - temporary storage for swapping elements of the first
c                  array
c temp4          - temporary storage for swapping elements of the fourth
c                  array
c temp5          - temporary storage for swapping elements of the fifth
c                  array
c
      real       temp1,temp4,temp5
c
c -------------------------------------------------------------------------
c real arrays :
c
c array1         - the first (index) array
c array4         - the fourth array
c array5         - the fifth array
c
      real       array1(800),array4(16,16,800),array5(16,16,800)
c
c -------------------------------------------------------------------------
c =========================================================================
c #########################################################################
c -------------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c -------------------------------------------------------------------------
c =========================================================================
c -------------------------------------------------------------------------
c start off assuming the whole array is unsorted :
c
      start=1
c
c -------------------------------------------------------------------------
c find the minimum element in the unsorted part of the array :
c
```

```
   20 first=0
      do 10,pos=start,nelem,1
        if (array1(pos) .gt. first) then
          first=array1(pos)
          fst=pos
        end if
   10 continue
c
c ------------------------------------------------------------------------
c swap the array elements over, so that the minimum found in the unsorted
c part becomes the last element in the sorted part :
c
      temp1=array1(start)
      array1(start)=array1(fst)
      array1(fst)=temp1
      do 30,band=1,nband,1
        temp2=array2(band,start)
        array2(band,start)=array2(band,fst)
        array2(band,fst)=temp2
        temp3=array3(band,start)
        array3(band,start)=array3(band,fst)
        array3(band,fst)=temp3
        do 40,bnd=1,nband,1
          temp4=array4(bnd,band,start)
          array4(bnd,band,start)=array4(bnd,band,fst)
          array4(bnd,band,fst)=temp4
          temp5=array5(bnd,band,start)
          array5(bnd,band,start)=array5(bnd,band,fst)
          array5(bnd,band,fst)=temp5
 40     continue
 30   continue
c
c ------------------------------------------------------------------------
c one more element has been placed in the right order :
c
      start=start+1
c
c ------------------------------------------------------------------------
c if there are any more elements to sort, go back and do them :
c
      if (start .lt. nelem) goto 20
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
```

```
c
      return
c
c -------------------------------------------------------------------------
c =========================================================================
c -------------------------------------------------------------------------
c
      end
c
c -------------------------------------------------------------------------
c #########################################################################
c -------------------------------------------------------------------------
c
```

SEGMENT :

```
c
c +--------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.    |
c |   work at King's College London, 1988-1992.  This work was funded    |
c |   by the SERC and BP Research International.  This software comes    |
c |   with no guarantees, and you use it at your own risk.  The author   |
c |   will probably be prepared to help you out with any problems, if    |
c |   you can track him down.             Philip J. Naylor, July 1992.   |
c +--------------------------------------------------------------------+
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
      subroutine segment(images,xsize,ysize,nband,classave,
     &                   classwid,nclass,segim,dist)
c
c   ******************************************************************
c   *                                                                *
c   *               a subroutine to segment the image,               *
c   *                     in coocurrence space.                      *
c   *                                                                *
c   ******************************************************************
c
c ======================================================================
c >>>>>>>>>>>>             variable declarations            <<<<<<<<<<<<
c ----------------------------------------------------------------------
c integer variables :
c
c band          - band number.
c c             - class number.
c class         - class number.
c classif       - alternative class number.
c direct        - cooccurrence direction number.
c dimn          - dimension number, in cooccurrence space.
c dist          - width of border to leave around edge of image.
c itemp         - temporary variable, used to convert byte type to
c                 integer type.
c limit         - the number of classes which will be considered when
c                 classifying pixels.
c nclass        - number of classes.
c nband         - number of bands.
c ndimn         - the number of dimensions in the cooccurrence space.
c near          - the class number of the class that a particular pixel
c                 most probably belongs to.
c positn        - indicates position in an array.
```

```
c x           - loop counter over image (x).
c xsize       - image size (x).
c y           - loop counter over image (y).
c ysize       - image size (y).
c
      integer    positn,nband,band,dist,xsize,ysize,nclass,class,c
      integer    ndimn,dimn,x,y,near,direct,limit,itemp
c
c ---------------------------------------------------------------------
c integer arrays :
c
c bound       - off diagonal class widths.
c classave    - the class mean vectors.
c classes     - list of class numbers.
c classwid    - along diagonal class widths.
c inten       - feature vector.
c nrclass     - the classes which are most similar to a class.
c vector      - cooccurrence direction vector.
c
      integer    classave(16,800),classwid(16,800),inten(32),vector(2)
      integer    nrclass(800,10),classes(800),bound(16)
c
c ---------------------------------------------------------------------
c byte variables :
c
c dummy       - dummy argument to pass to dataout.
c
      byte       dummy
c
c ---------------------------------------------------------------------
c byte arrays :
c
c images      - image data.
c segim       - segmented image.
c
      byte       images(xsize,ysize,nband),segim(xsize,ysize)
c
c ---------------------------------------------------------------------
c real variables :
c
c distn       - the Eucliden distance between two class mean vectors.
c edge        - single edge probability value.
c mahal       - non-Euclidean distance from a feature vector to a
c               class mean vector (n.b. despite appearances, this is
c               NOT the Mahalanobis distance).
c mini        - distance to the nearest class mean vector.
c secnear     - distance to the second nearest class mean vector.
```

```
c
      real       mini,distn,secnear,mahal,edge,offa
c
c ----------------------------------------------------------------------------
c real arrays :
c
c dis           - distances from a feature vector to all the class means.
c edges         - edge probability image.
c
      real       dis(800),edges(262144)
c
c ----------------------------------------------------------------------------
c ============================================================================
c ############################################################################
c ----------------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c ----------------------------------------------------------------------------
c ============================================================================
c ----------------------------------------------------------------------------
c announce what stage the program has reached :
c
      write(*,*) ' Segmenting the image.'
c
c ----------------------------------------------------------------------------
c the number of dimensions in the cooccurrence space is twice that in the
c histogram space :
c
      ndimn=2*nband
c
c ----------------------------------------------------------------------------
c initialise the segmented image, and edge probability map, arrays :
c
      do 1,y=1,ysize,1
        do 2,x=1,xsize,1
          segim(x,y)=nclass-127
          edges(((y-1)*xsize)+x)=0.0
    2   continue
    1 continue
c
c ----------------------------------------------------------------------------
c find the minimum along diagonal class width in each band, to use as the
c off diagonal widths (i.e. all clusters are either circular, or elongated
c along the diagonal) :
c
      do 91,band=1,nband,1
```

```
         bound(band)=65536
         do 92,class=1,nclass,1
            bound(band)=min(bound(band),classwid(band,class))
 92      continue
 91   continue
c
c ---------------------------------------------------------------------------
c if there are more than ten classes, make a list of the ten which are
c most similar to each class :
c
      limit=min(10,nclass)
      if (limit .eq. nclass) then
        do 8,class=1,limit,1
          do 9,c=1,limit,1
            nrclass(c,class)=class
    9     continue
    8   continue
      else
        do 3,class=1,nclass,1
          do 4,c=1,nclass,1
            if (c .ne. class) then
              dis(c)=0
              do 5,band=1,nband,1
                distn=classave(band,class)-classave(band,c)
                dis(c)=dis(c)+(distn*distn)
    5         continue
              dis(c)=sqrt(dis(c))
            end if
    4     continue
          dis(class)=0.0
          do 6,c=1,nclass,1
            classes(c)=c
    6     continue
          call sort2(dis,classes,nclass)
          do 7,c=1,10,1
            nrclass(class,c)=classes(c)
    7     continue
    3   continue
      end if
c
c ---------------------------------------------------------------------------
c loop over the part of the image which was smoothed :
c
      do 110,y=(dist+1),(ysize-dist),1
        do 120,x=(dist+1),(xsize-dist),1
c
c ---------------------------------------------------------------------------
```

```
c initialise the the non-Euclidean distance measure :
c
        mahal=1.0
c
c -----------------------------------------------------------------------
c loop over the cooccurrence directions :
c
        do 80,direct=1,4,1
c
c -----------------------------------------------------------------------
c set the cooccurrence direction vector :
c
         vector(1)=min(1,direct-1)
         vector(2)=max(-1,direct-3)
c
c -----------------------------------------------------------------------
c calculate the feature vector :
c
          do 30,dimn=2,ndimn,2
            band=dimn/2
            itemp=images(x,y,band)
            inten(dimn)=itemp+128
            itemp=images(x+vector(1),y+vector(2),band)
            inten(dimn-1)=itemp+128
   30     continue
c
c -----------------------------------------------------------------------
c rotate it to get the along diagonal, and off diagonal, components :
c
          do 40,dimn=2,ndimn,2
            inten(dimn-1)=(inten(dimn-1)+inten(dimn))/2
            inten(dimn)=inten(dimn-1)-inten(dimn)
   40     continue
c
c -----------------------------------------------------------------------
c if the pixel is currently unclassified, check ALL classes to see which
c it best fits :
c
         if (segim(x,y) .eq. nclass-127) then
c
c -----------------------------------------------------------------------
c calculate the non-Euclidean distance from the feature vector to each
c cluster centre :
c
c the along diagonal component :
c
          do 50,class=1,nclass,1
```

```
            dis(class)=0
            do 60,dimn=1,(ndimn-1),2
              dis(class)=dis(class)+
     &                   ((1.0*(inten(dimn)-classave((dimn/2)+1,class))
     &                   *(inten(dimn)-classave((dimn/2)+1,class)))
     &                   /(classwid((dimn/2)+1,class)*
     &                     classwid((dimn/2)+1,class)))
   60       continue
c
c the off diagonal component :
c
            do 65,dimn=2,ndimn,2
              dis(class)=dis(class)+
     &                   ((1.0*inten(dimn)*inten(dimn))/
     &                    (bound(band)*bound(band)))
   65       continue
   50      continue
c
c --------------------------------------------------------------------------
c find the nearest cluster centre :
c
           secnear=1000000
           mini=1000000
           do 70,class=1,nclass,1
             if (dis(class) .lt. secnear) secnear=dis(class)
             if (dis(class) .lt. mini) then
               secnear=mini
               mini=dis(class)
               near=class
             end if
   70      continue
c
c --------------------------------------------------------------------------
c classify the pixel :
c
           mahal=dis(near)
           segim(x,y)=near-128
c
c --------------------------------------------------------------------------
c calculate the empirical edge probability :
c
           offa=0.0
           do 1000,dimn=2,ndimn,2
             offa=offa+(inten(dimn)*inten(dimn))
 1000      continue
           offa=sqrt(offa)/(0.5*bound(band))
c
```

```
c general term :
c
          edges(((y-1)*xsize)+x)=min(1.0-exp(log(0.05)*(mini*mini)),
     &          1.0-exp(log(0.05)*(offa*offa)))
          if ((mini .lt. 1.0) .and. (secnear .lt. 1.0)) then
c
c term for feature vector between two class mean vectors :
c
            edges(((y-1)*xsize)+x)=max(edges(((y-1)*xsize)+x),
     &                                 (1.0/(1.0+secnear-mini))**8.0)
          end if
c
c -----------------------------------------------------------------------
c if the pixel has already been classified, check ONLY the ten classes
c most similar to the one it is already in :
c
         else
c
c -----------------------------------------------------------------------
c calculate the non-Euclidean distance from the feature vector to each
c cluster centre :
c
           c=segim(x,y)+128
           do 150,class=1,limit,1
c
c along diagonal term :
c
            dis(nrclass(c,class))=0
             do 160,dimn=1,(ndimn-1),2
               dis(nrclass(c,class))=dis(nrclass(c,class))+
     &         ((1.0*(inten(dimn)-
     &         classave((dimn/2)+1,nrclass(c,class)))
     &         *(inten(dimn)-classave((dimn/2)+1,nrclass(c,class))))
     &         /(classwid((dimn/2)+1,nrclass(c,class))*
     &         classwid((dimn/2)+1,nrclass(c,class))))
  160        continue
c
c off diagonal term :
c
             do 165,dimn=2,ndimn,2
               dis(nrclass(c,class))=dis(nrclass(c,class))+
     &                  ((1.0*inten(dimn)*inten(dimn))/
     &                   (bound(band)*bound(band)))
  165        continue
  150      continue
c
c -----------------------------------------------------------------------
```

```
c find the nearest cluster centre :
c
            mini=1000000
            secnear=1000000
            do 170,class=1,limit,1
              if (dis(nrclass(c,class)) .lt. secnear)
     &                                  secnear=dis(nrclass(c,class))
              if (dis(nrclass(c,class)) .lt. mini) then
                secnear=mini
                mini=dis(nrclass(c,class))
                near=nrclass(c,class)
              end if
  170       continue
c
c ---------------------------------------------------------------------------
c if this classification is better than the previous one, use it :
c
            if (dis(near) .lt. mahal) then
              segim(x,y)=near-128
              mahal=dis(near)
            end if
c
c ---------------------------------------------------------------------------
c calculate the empirical edge probability value :
c
            offa=0.0
            do 2000,dimn=2,ndimn,2
              offa=offa+(inten(dimn)*inten(dimn))
 2000       continue
            offa=sqrt(offa)/(0.5*bound(band))
c
c general term :
c
            edge=min(1.0-exp(log(0.05)*(mini*mini)),
     &                1.0-exp(log(0.05)*(offa*offa)))
c
c term for feature vector between two class mean vectors :
c
            if ((mini .lt. 1.0) .and. (secnear .lt. 1.0)) then
              edge=max(edges(((y-1)*xsize)+x),
     &                  (1.0/(1.0+secnear-mini))**8.0)
            end if
c
c ---------------------------------------------------------------------------
c if the pixel is more edge-like in this direction, use this probability :
c
            if (edge .gt. edges(((y-1)*xsize)+x))
     &                                edges(((y-1)*xsize)+x)=edge
c
c ------------------------------------------------------------------------
c end of the decision about whether or not a pixel is already classified :
c
           end if
c
c ------------------------------------------------------------------------
c go on to the next cooccurrence direction :
c
   80     continue
c
c ------------------------------------------------------------------------
c go on to the next pixel :
c
  120   continue
  110 continue
c
c ------------------------------------------------------------------------
c write out the edge probability image to a file :
c
      call dataout('File for edge probability image',dummy,edges,
     &             xsize,ysize,1,4)
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
```

SORT2 :

```
c
c +-----------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.     |
c |   work at King's College London, 1988-1992.  This work was funded     |
c |   by the SERC and BP Research International.  This software comes     |
c |   with no guarantees, and you use it at your own risk.  The author    |
c |   will probably be prepared to help you out with any problems, if     |
c |   you can track him down.             Philip J. Naylor, July 1992.    |
c +-----------------------------------------------------------------------+
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
      subroutine sort2(array1,array2,nelem)
c
c ************************************************************************
c *                                                                      *
c *    a subroutine to sort two arrays with respect to the first one     *
c *                      (ascending order)                               *
c *                        [NOT OPTIMAL]                                 *
c *                                                                      *
c ************************************************************************
c
c ========================================================================
c >>>>>>>>>>>>            variable declarations             <<<<<<<<<<<<
c ========================================================================
c integer variables :
c
c first          - the smallest value in the unsorted part
c fst            - the position of the smallest value in the unsorted
c                  part
c nelem          - the number of elements in the arrays
c pos            - loop counter for going through array
c start          - start of unsorted portion of array
c temp1          - temporary storage for swapping elements of first
c                  array
c temp2          - temporary storage for swapping elements of second
c                  array
c
      integer    pos,temp1,temp2,first,start,fst,nelem
c
c ------------------------------------------------------------------------
c integer arrays :
c
c array1        - the first (index) array
```

```
c array2       - the second array
c
      integer    array1(nelem),array2(nelem)
c
c -----------------------------------------------------------------------
c =======================================================================
c #######################################################################
c -----------------------------------------------------------------------
c
c
c the start of the subroutine proper :
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c start off assuming the whole array is unsorted :
c
      start=1
c
c -----------------------------------------------------------------------
c find the minimum element in the unsorted part of the array :
c
   20 first=1000000
      do 10,pos=start,nelem,1
        if (array1(pos) .lt. first) then
          first=array1(pos)
          fst=pos
        end if
   10 continue
c
c -----------------------------------------------------------------------
c swap the array elements over, so that the minimum found in the unsorted
c part becomes the last element in the sorted part :
c
      temp1=array1(start)
      temp2=array2(start)
      array1(start)=array1(fst)
      array2(start)=array2(fst)
      array1(fst)=temp1
      array2(fst)=temp2
c
c -----------------------------------------------------------------------
c one more element has been placed in the right order :
c
      start=start+1
c
c -----------------------------------------------------------------------
```

```
c if there are any more elements to sort, go back and do them :
c
      if (start .lt. nelem) goto 20
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
```

DATAOUT :

```
c
c +-------------------------------------------------------------------+
c |   This software produced by Philip J. Naylor as part of his Ph.D.  |
c |   work at King's College London, 1988-1992.  This work was funded  |
c |   by the SERC and BP Research International.  This software comes  |
c |   with no guarantees, and you use it at your own risk.  The author |
c |   will probably be prepared to help you out with any problems, if  |
c |   you can track him down.             Philip J. Naylor, July 1992.  |
c +-------------------------------------------------------------------+
c
c ---------------------------------------------------------------------
c #####################################################################
c ---------------------------------------------------------------------
c
      subroutine dataout(prompt,images,rimages,xsize,ysize,
     &                   lun,nbytes)
c
c *********************************************************************
c *                                                                   *
c *        a subroutine to write out single band image data.          *
c *                                                                   *
c *********************************************************************
c
c =====================================================================
c >>>>>>>>>>>>             variable declarations            <<<<<<<<<<<<
c ---------------------------------------------------------------------
c integer variables :
c
c band          - band number (always 1 in this case).
c lun           - logical unit number of file.
c nband         - the number of bands (always 1 in this case).
c nbytes        - 1 => byte data is to written,
c                 4 => real*4 data is to be written.
c num           - record number in file.
c positn        - position in the image.
c xsize         - image size (x).
c ysize         - image size (y).
c
      integer    band,count,nband,nclass,num,lun,nbytes
      integer    pos,positn,xsize,ysize,bnd
c
c ---------------------------------------------------------------------
c byte arrays :
c
c images        - image data of type byte.
c bimages       - byte version of real*4 data.
```

```
c
      byte       images(512*512*16),bimages(512*512*16)
c
c ------------------------------------------------------------------------
c real variables :
c
c maxm          - maximum value in the real data.
c minm          - minimum value in the real data.
c
      real       maxm,minm
c
c ------------------------------------------------------------------------
c real arrays :
c
c rimages       - image data of type real*4.
c
      real       rimages(512*512*16)
c
c ------------------------------------------------------------------------
c logical variables :
c
c again         - true if opening or writing a file is to be reattempted
c                 if an error occured.
c
      logical    again
c
c ------------------------------------------------------------------------
c character variables :
c
c filename      - the name of the file to be written.
c prompt        - the prompt to be used when asking for the file name.
c
      character  filename*80,prompt*(*)
c
c ----------------------------------------------------------------------
c ======================================================================
c >>>>>>>>>>>>           function declarations            <<<<<<<<<<<<
c ----------------------------------------------------------------------
c logical functions :
c
c   yes          - 'eric' library routine, used to get a yes or no
c                  answer to a given question.
c
      logical    yes
c
c ----------------------------------------------------------------------
c ======================================================================
c ######################################################################
c ----------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c get the name of the file :
c
 1000       write(*,666) prompt
            read(*,'(a)') filename
 666        format(1x,a,' : ',$)
c
c ------------------------------------------------------------------------
c set the number of bands (always 1) :
c
              nband=1
              band=1
c
c ------------------------------------------------------------------------
c if the data is from the byte array :
c
            if (nbytes .eq. 1) then
c
c ------------------------------------------------------------------------
c open the file :
c
                 open (unit=lun,file=filename//'.'//char(band+64),
     &              access='direct',status='new',
     &              organization='sequential',recl=128,err=2)
c
c ------------------------------------------------------------------------
c take the 2's complementation into account :
c
                 do 96,positn=((band-1)*xsize*ysize)+1,
     &                        (band*xsize*ysize),1
                    if (images(positn) .lt. 0) then
                      images(positn)=images(positn)+128
                    else
                      images(positn)=images(positn)-128
                    end if
   96            continue
c
c ------------------------------------------------------------------------
c write out the data :
c
```

```
                 do 93,num=1,(xsize*ysize)/512,1
                    write(lun,rec=num,err=4)
     &                    (images(((band-1)*(xsize*ysize))
     &                    +positn),positn=((num-1)*512)+1,num*512)
   93            continue
c
c ------------------------------------------------------------------------
c close the file :
c
                close(unit=lun)
c
c ------------------------------------------------------------------------
c if the data is from the real*4 array :
c
          else if (nbytes .eq. 4) then
c
c ------------------------------------------------------------------------
c initialise the extreme values :
c
               maxm=-1.0e+32
               minm=1.0e+32
c
c ------------------------------------------------------------------------
c find the extreme values in the data :
c
               do 192,band=1,nband,1
                 do 194,positn=((band-1)*xsize*ysize)+1,
     &                        (band*xsize*ysize),1
                   maxm=max(maxm,rimages(positn))
                   minm=min(minm,rimages(positn))
  194            continue
  192          continue
c
c ------------------------------------------------------------------------
c set the band number (always 1) :
c
               band=1
c
c ------------------------------------------------------------------------
c scale the real*4 values into the range -128 to 127, and copy into the
c byte array :
c
                 do 195,positn=((band-1)*xsize*ysize)+1,
     &                        (band*xsize*ysize),1
                   rimages(positn)=((rimages(positn)-minm)/
     &                             (maxm-minm))*255.0
                   bimages(positn)=int(rimages(positn))-128
```

```
 195             continue
c
c ---------------------------------------------------------------------
c open the file :
c
                 open (unit=lun,file=filename//'.'//char(band+32),
    &               access='direct',status='new',
    &               organization='sequential',recl=128,err=2)
c
c ---------------------------------------------------------------------
c take care of 2's complementation :
c
                 do 196,positn=((band-1)*xsize*ysize)+1,
    &                       (band*xsize*ysize),1
                    if (bimages(positn) .lt. 0) then
                      bimages(positn)=bimages(positn)+128
                    else
                      bimages(positn)=bimages(positn)-128
                    end if
  196            continue
c
c ---------------------------------------------------------------------
c write out the data :
c
                 do 193,num=1,(xsize*ysize)/512,1
                    write(lun,rec=num,err=4)
    &                     (bimages(((band-1)*(xsize*ysize))
    &                     +positn),positn=((num-1)*512)+1,num*512)
  193             continue
c
c ---------------------------------------------------------------------
c close the file :
c
                close(lun)
c
c ---------------------------------------------------------------------
c end of the decision about data type :
c
          end if
c
c ---------------------------------------------------------------------
c skip to the end :
c
        goto 3000
c
c -------------------------------------------------------------------
c #####################################################################
```

```
c ======================================================================
c ----------------------------------------------------------------------
c code for 'dealing' with problems which occur whilst handling byte data
c files :
c
    2 write(*,*) 'unable to open file ',filename
      again=yes('try again ? >',.true.)
      if (again) goto 1000
      stop
    4 write(*,*) 'error in reading from file ',filename
      again=yes('try again ? >',.false.)
      close(1)
      if (again) goto 1000
      stop
c
c ----------------------------------------------------------------------
c ======================================================================
c ----------------------------------------------------------------------
c
 3000 end
c
c ----------------------------------------------------------------------
c ######################################################################
c ----------------------------------------------------------------------
c
```

Appendix B

The `FORTRAN` source code for the implementations of Forgy's method, and MacQueen's k-means.

Both programs have the same main routine (`COMPARE`) which calls a single subroutine (`CLASSIFY`) to do the data analysis. There are two versions of `CLASSIFY`, one for Forgy's method, and one for MacQueen's k-means.

These programs also use some of the "ERIC" routines mentioned in appendix A.

COMPARE :

```
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
      program compare
c
c ***********************************************************************
c *                                                                     *
c *             a general program for data classification.             *
c *                                                                     *
c ***********************************************************************
c
c =======================================================================
c >>>>>>>>>>>>           variable declarations            <<<<<<<<<<<<
c -----------------------------------------------------------------------
c integer variables :
c
c class        - class number.
c dimn         - dimension number.
c sample       - sample number.
c nclass       - number of classes.
c ndimn        - number of dimensions.
c nsample      - number of samples.
c
      integer   dimn,ndimn,nsample,sample,class,nclass
c
c -----------------------------------------------------------------------
c integer arrays :
c
c data         - the data.
c
      integer   data(16,256000)
c
c -----------------------------------------------------------------------
c real arrays :
c
c mean         - the class mean vectors.
c
      real      mean(16,16)
c
c -----------------------------------------------------------------------
c =======================================================================
c #######################################################################
c -----------------------------------------------------------------------
c
```

```
c the start of the program proper :
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c get details of the data set :
c
      ndimn=intin('Number of dimensions >',8,1,64)
      nclass=intin('Number of classes >',4,1,64)
      nsample=intin('Number of samples >',1000,1,1000000)
c
c ------------------------------------------------------------------------
c read in the data :
c
      call open('Data file >','data','data',1,'FR')
      do 1,sample=1,nsample,1
         read(1,*) (data(dimn,sample),dimn=1,ndimn)
 1    continue
      close(1)
c
c ------------------------------------------------------------------------
c classify the data, and find the class mean vectors :
c
      call classify(data,nsample,ndimn,nclass,mean)
c
c ------------------------------------------------------------------------
c write out the mean vectors :
c
      call open('Means file >','data','means',1,'FN')
      do 2,class=1,nclass,1
         write(1,*) (mean(dimn,class),dimn=1,ndimn)
 2    continue
      close(1)
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c the end of the program :
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
```

```
c ######################################################################
c ----------------------------------------------------------------------
c
```

CLASSIFY (Forgy's method) :

```
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
      subroutine classify(data,nsample,ndimn,nclass,mean)
c
c ***********************************************************************
c *                                                                     *
c *       a subroutine to implement a version of Forgy's method         *
c *                      for custer analysis.                           *
c *                                                                     *
c ***********************************************************************
c
c =======================================================================
c >>>>>>>>>>>>          variable declarations               <<<<<<<<<<<<
c -----------------------------------------------------------------------
c integer variables :
c
c class        - class number.
c dimn         - dimension number.
c nclass       - number of classes.
c ndimn        - number of dimensions.
c nearclass    - class number of class closest to a cluster centre.
c nsample      - number of samples.
c sample       - sample number.
c seed         - seed for random number generator.
c
      integer   sample,nsample,ndimn,dimn,nclass,class,seed,nearclass
c
c -----------------------------------------------------------------------
c integer arrays :
c
c classif      - classification of the data points.
c data         - the data.
c
      integer   data(16,256000),classif(256000)
c
c -----------------------------------------------------------------------
c real variables :
c
c dist         - Euclidean distance from a feature vector to a cluster
c                centre.
c error        - mean squared error of feature vectors compared with
c                class mean vectors.
c mindist      - the minimum distance from a feature vector to a class
```

```
c               mean vector, with respect to class number.
c olderror     - the value of error on the previous iteration.
c
      real      dist,mindist,error,olderror
c
c -----------------------------------------------------------------------
c real arrays :
c
c mean         - class mean vectors.
c size         - class sizes.
c
      real      mean(16,16),size(16)
c
c -----------------------------------------------------------------------
c =======================================================================
c #######################################################################
c -----------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c set the random number generator seed :
c
      seed=272
c
c -----------------------------------------------------------------------
c initialise the ‘old’ mean squared error :
c
      olderror=1.0e+32
c
c -----------------------------------------------------------------------
c choose a set of initial, random, class mean vectors :
c
      do 2,class=1,nclass,1
         do 3,dimn=1,ndimn,1
            mean(dimn,class)=ran(seed)*255
 3       continue
 2    continue
c
c -----------------------------------------------------------------------
c classify the data by minimum Euclidean distance :
c
 1000 do 8,sample=1,nsample,1
         mindist=1000000
         do 9,class=1,nclass,1
```

```
            dist=0.0
            do 10,dimn=1,ndimn,1
               dist=dist+((data(dimn,sample)-mean(dimn,class))**2.0)
 10         continue
            if (dist .lt. mindist) then
               mindist=dist
               nearclass=class
            end if
 9       continue
         classif(sample)=nearclass
 8    continue
c
c ----------------------------------------------------------------------
c recalculate the class mean vectors :
c
c initialise class means, and sizes :
c
      do 12,class=1,nclass,1
         size(class)=0.0
         do 13,dimn=1,ndimn,1
            mean(dimn,class)=0.0
 13      continue
 12   continue
c
c accumulate means :
c
      do 14,sample=1,nsample,1
         class=classif(sample)
         size(class)=size(class)+1.0
         do 15,dimn=1,ndimn,1
            mean(dimn,class)=mean(dimn,class)+data(dimn,sample)
 15      continue
 14   continue
c
c if a class has "vanished" set a new, random, class mean vector, and
c reset olderror, otherwise just divide by the class size :
c
      do 16,class=1,nclass,1
         if (size(class) .eq. 0) then
            do 28,dimn=1,ndimn,1
               mean(dimn,class)=ran(seed)*255
 28         continue
         olderr=1.0e+32
         else
            do 17,dimn=1,ndimn,1
               mean(dimn,class)=mean(dimn,class)/size(class)
 17         continue
```

```
         end if
 16   continue
c
c -----------------------------------------------------------------------
c calculate the mean squared error of the classification :
c
      error=0.0
      do 18,sample=1,nsample,1
         class=classif(sample)
         do 19,dimn=1,ndimn,1
            error=error+
     &            (((data(dimn,sample)-mean(dimn,class))**2.0)/nsample)
 19      continue
 18   continue
c
c -----------------------------------------------------------------------
c if the error is still decreasing reiterate, otherwise stop :
c
      if (int(error*1000) .lt. int(olderror*1000)) then
         olderror=error
         goto 1000
      else
         return
      end if
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c the end of the subroutine :
c
c -----------------------------------------------------------------------
c #######################################################################
c =======================================================================
c -----------------------------------------------------------------------
c
      end
c
c -----------------------------------------------------------------------
c #######################################################################
c -----------------------------------------------------------------------
c
```

CLASSIFY (MacQueen's *k*-means) :

```
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
      subroutine classify(data,nsample,ndimn,nclass,mean)
c
c ************************************************************************
c *                                                                      *
c *          a subroutine to implement a version of MacQueen's           *
c *                k-means method for cluster analysis.                  *
c *                                                                      *
c ************************************************************************
c
c ========================================================================
c >>>>>>>>>>>>             variable declarations              <<<<<<<<<<<<
c ------------------------------------------------------------------------
c integer variables :
c
c class        - class number.
c dimn         - dimension number.
c nclass       - number of classes.
c ndimn        - number of dimesnions.
c nearclass    - class number of class closest to a cluster centre.
c nsample      - number of samples.
c sample       - sample number.
c seed         - seed for random number generator.
c
      integer   sample,nsample,ndimn,dimn,nclass,class,seed,nearclass
c ------------------------------------------------------------------------
c integer arrays :
c
c classif      - classification of the data points.
c data         - the data.
c
      integer   data(16,256000),classif(256000)
c
c ------------------------------------------------------------------------
c real variables :
c
c dist         - Euclidean distance from a feature vector to a cluster
c                centre.
c mindist      - the minimum distance from a feature vector to a class
c                mean vector, with respect to class number.
c
      real      dist,mindist
```

```
c -----------------------------------------------------------------------
c real arrays :
c
c mean         - class mean vectors.
c size         - class sizes.
c
      real      mean(16,16),size(16)
c
c -----------------------------------------------------------------------
c =======================================================================
c #######################################################################
c -----------------------------------------------------------------------
c
c the start of the subroutine proper :
c
c -----------------------------------------------------------------------
c =======================================================================
c -----------------------------------------------------------------------
c set the random number generator seed :
c
      seed=272
c
c -----------------------------------------------------------------------
c pick a random set of k samples, and use them as the class mean vectors :
c
      do 1,class=1,nclass,1
         size(class)=0.0
         sample=nint(ran(seed)*nsample)
         do 2,dimn=1,ndimn,1
            mean(dimn,class)=data(dimn,sample)
 2       continue
 1    continue
c
c -----------------------------------------------------------------------
c classify the data by minimum Euclidean distance :
c
      do 8,sample=1,nsample,1
         mindist=1000000
         do 9,class=1,nclass,1
            dist=0.0
            do 10,dimn=1,ndimn,1
               dist=dist+((data(dimn,sample)-mean(dimn,class))**2.0)
 10         continue
            if (dist .lt. mindist) then
               mindist=dist
               nearclass=class
            end if
 9       continue
c
c increment the appropriate class size, and adjust the class mean vector :
c
         size(nearclass)=size(nearclass)+1.0
         do 11,dimn=1,ndimn,1
            mean(dimn,nearclass)=
     &              ((mean(dimn,nearclass)*(size(nearclass)-1.0))+
     &                data(dimn,sample))/size(nearclass)
 11      continue
 8    continue
c
c ----------------------------------------------------------------------
c reclassify the data :
c
      do 18,sample=1,nsample,1
         mindist=1000000
         do 19,class=1,nclass,1
            dist=0.0
            do 110,dimn=1,ndimn,1
               dist=dist+((data(dimn,sample)-mean(dimn,class))**2.0)
 110        continue
            if (dist .lt. mindist) then
               mindist=dist
               nearclass=class
            end if
 19      continue
         classif(sample)=nearclass
 18   continue
c
c ----------------------------------------------------------------------
c initialise the class mean vectors :
c
      do 12,class=1,nclass,1
         size(class)=0.0
         do 13,dimn=1,ndimn,1
            mean(dimn,class)=0.0
 13      continue
 12   continue
c
c ----------------------------------------------------------------------
c accumulate the means :
c
      do 14,sample=1,nsample,1
         class=classif(sample)
         size(class)=size(class)+1.0
         do 15,dimn=1,ndimn,1
```

```
            mean(dimn,class)=mean(dimn,class)+data(dimn,sample)
 15      continue
 14   continue
c
c ------------------------------------------------------------------------
c divide by the class sizes :
c
      do 16,class=1,nclass,1
         do 17,dimn=1,ndimn,1
            mean(dimn,class)=mean(dimn,class)/size(class)
 17      continue
 16   continue
c
c ------------------------------------------------------------------------
c ========================================================================
c ------------------------------------------------------------------------
c the end of the subroutine :
c
      return
c
c ------------------------------------------------------------------------
c ########################################################################
c ========================================================================
c ------------------------------------------------------------------------
c
      end
c
c ------------------------------------------------------------------------
c ########################################################################
c ------------------------------------------------------------------------
c
```

References

ALI M. & AGGARWAL J.K., *Automatic interpretation of infrared aerial color photographs of citrus orchards having infestations of insect pest and diseases.* IEEE Transactions on Geoscience and Electronics, **15** (3), 1977.

ANDERBERG M.R., *Cluster Analysis for Applications.* Academic Press, New York, 1973.

BALL G.H. & HALL D.J., *ISODATA, a novel method of data analysis and pattern classification.* Stanford Research Institute, Menlo Park, California, 1965.

BANNINGER C., *Remote sensing of a biogeochemical anomaly associated with a base-metal deposit in the Spanish Pyrite Belt.* Proceedings of "Remote Sensing - an operational technology for the mining and petroleum industries.", Institute of Mining and Metallurgy, London, 1990.

BARNSLEY M.J., BARR S.L., & SADLER G.J., *Spatial re-classification of remotely-sensed images for urban land-use monitoring.* Proceedings of a joint conference of the Photogrammetric Society, the Remote Sensing Society, & the American Society for Photogrammetry and Remote Sensing - "Spatial Data 2000.", Christ Church College, University of Oxford, 1991.

BEDELL R.L., ODINGA M., NIYONDEZO S., NKURIKIYE L., FERNANDEZ-ALONSO M., & TREFOIS Ph., *Turbidite-hosted Proterozoic gold exploration in the Kibaran Belt, Burundi, Central Africa.* Proceedings of "Remote Sensing - an operational technology for the mining and petroleum industries.", Institute of Mining and Metallurgy, London, 1990.

BESAG J., *On the statistical analysis of dirty pictures.* Journal of the Royal Statistical Society, **B 48**, 1986.

BIGGAR M.J. & CONSTANTINIDES A. G., *Thin line coding techniques.* Proceedings of an International Conference on Digital Signal Processing, Florence, Italy, 1987.

BOCCHI S., GALLI A., MORIONDO A., ROSSI B., & TOMASONI R., *Changes detection in agricultural and urban land use by remote sensing techniques : the metropolitan area of Milan.* Proceedings of the 15th Annual Conference of the Remote Sensing Society - "Remote Sensing for Operational Applications.", University of Bristol, 1989.

BRACEWELL R.N., *The Fourier Transform and its Applications, 2nd Edition.* McGraw-Hill, New York, 1978.

CANNY J.F., *A computational approach to edge detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, **8** (6), 1986.

CARBON G.E. & EBEL W.J., *Co-occurrence matrix modification for small region texture measurement and comparison.* Proceedings of the International Geoscience And Remote Sensing Symposium - "Remote Sensing : moving towards the 21st century.", University of Edinburgh, 1988.

CROSS A.M., *Deforestation assesment in the Amazon basin using NOAA/ AVHRR data.* Proceedings of the 15th Annual Conference of the Remote Sensing Society - "Remote Sensing for Operational Applications.", University of Bristol, 1989.

DANSON F.M., STEVEN M.D., MALTHUS T.J., & JAGGARD K.W., *Spectral response of sugar beet to water stress.* Proceedings of the 16th Annual Conference of the Remote Sensing Society - "Remote Sensing and Global Change.", University College of Swansea, 1990.

FOODY G.M. & COX D.P., *Estimation of sub-pixel land cover composition from spectral mixture models.* Proceedings of a joint conference of the Photogrammetric Society, the Remote Sensing Society, & the American Society for Photogrammetry and Remote Sensing - "Spatial Data 2000.", Christ Church College, University of Oxford, 1991.

FORGY E.W., *Cluster analysis of multivariate data : efficiency versus interpretability of classification.* Biometric Society Meetings, Riverside, California, (abstract in Biometrics, **21** (3)), 1965

FORSYTH D. & ZISSERMAN A., *Shape from shading in the light of mutual illumination.* Proceedings of the Fifth Alvey Vision Conference, University of Reading, 1989.

GOLDBERG M. & SHLIEN S., *A clustering scheme for multispectral images.* IEEE Transactions on Systems, Man, and Cybernetics, **8** (2), 1978.

GONZALEZ R.C. & WINTZ P., *Digital Image Processing, 2nd Edition.* Addison-Wesley, Reading, Massachusetts, 1987.

HADDON J.F., private correspondence, 1987.

HADDON J.F. & BOYCE J.F., *Image segmentation by unifying region and boundary information.* IEEE Transactions on Pattern Analysis and Machine Intelligence, **12** (10), 1990.

HARALICK R.M., SHANMUGAM K., & DINSTEIN I., *Textural features for image classification.* IEEE Transactions on Systems, Man, and Cybernetics, **3** (6), 1973.

HARTLEY H.O., *Maximum likelihood estimation from incomplete data.* Biometrics, **14**, 1958.

HELSTROM C.W., *Image restoration by the method of least squares.* Journal of the Optical Society of America, **57** (3), 1967.

HÖGBOM J.A., *Aperture synthesis with a non-regular distribution of interferometer baselines.* Astronomy and Astrophysics Supplements, **15**, 1974.

HORN B.K.P., *Robot Vision.* MIT Press, Cambridge, Massachusetts, 1986.

HOUGH P.V.C., *Method and means for recognising complex patterns.* US Patent 3069654, 1962.

IMAGE PROCESSING magazine (Reed Business Publishing), news item *Imaging tackles drug barons.* **4** (1), 1992.

ISO/JTC1/SC2/WG8 N800, International Organisation for Standardisation - *Initial draft for adaptive discrete cosine transform technique for still picture data compression standard.*

JANCEY R.C., *Multidimensional group analysis.* Australian Journal of Botany, **14** (1), 1966.

KAUTH R.J. & THOMAS G.S., *The tasseled cap - a graphic description of the spectral-temporal development of agricultural crops as seen by LANDSAT.* Proceedings of the LARS Symposium on Machine Processing of Remotely Sensed Data, Purdue University, 1976.

KIRKPATRICK S., GELATT C.D., & VECCHI M.P., *Optimization by simulated annealing.* Science, **220**, 1983.

KUNDU A., MITRA S.K., & VAIDYANATHAN P.P, *Application of two-dimensional generalized mean filtering for removal of impulsive noises from images.* IEEE Transactions on Acoustics, Speech, and Signal Processing, **32** (3), 1984.

LAND E.H., *The retinex theory of color vision.* Scientific American, **237** (6), 1977.

LEE Y.H. & KASSAM S.A., *Generalized median filtering and related nonlinear filtering techniques.* IEEE Transactions on Acoustics, Speech, and Signal Processing, **33** (3), 1985.

LEE Y.-H. & TANTARATANA S., *Decision-based order statistic filters.* IEEE Transactions on Acoustics, Speech, and Signal Processing, **38** (3), 1990.

LETTS P.A., *Unsupervised classification in the Aries image analysis system.* Proceedings of the 5th Canadian Symposium on Remote Sensing, 1978.

LOUGHLIN W.P., *Geological exploration in the western United States by use of airbourne scanner imagery.* Proceedings of "Remote Sensing - an operational technology for the mining and petroleum industries.", Institute of Mining and Metallurgy, London, 1990.

MACQUEEN J.B., *Some methods for classification and analysis of multivariate observations.* Proceedings of a Symposium on Mathematics, Statistics, and Probability, University of California at Berkeley, 1967.

MALTHUS T.J., ANDRIEU B., BARET F., CLARK J.A., DANSON F.M., JAGGARD K.W., & STEVEN M.D., *Candidate high spectral resolution infrared indices for the prediction of crop cover.* Proceedings of the 16th Annual Conference of the Remote Sensing Society - "Remote Sensing and Global Change.", University College of Swansea, 1990.

MORAN D. & MORRIS O.J., *Region and texture coding of TV pictures.* Proceedings of the IEE 3rd International Conference on Image Processing and its Applications, University of Warwick, 1989.

MUIRHEAD K. & CRACKNELL A.P., *Gas flares and forest fires - the potential of AVHRR band 3.* Proceedings of the 10th Anniversary Conference of the Remote Sensing Society - "Satellite Remote Sensing - Review and Preview.", University of Reading, 1984.

MURTAGH F., *Multivariate analysis methods.* in "Pattern Recognition and Image Processing in Physics." - Proceedings of the 37th Scottish Universities Summer School in Physics, (edited by R.A. Vaughan), Adam Hilger, Bristol, 1990.

NAYLOR P.J., M.Sc. thesis - *Towards the easy processing of images produced by radio interferometers.* University of Manchester, 1988.

NRSC PAMPHLET G 06, *Applications of radar imagery to geological studies.* National Remote Sensing Centre, Farnborough, Hampshire, 1986.

PETROU M., *Optimal edges detectors for linear features with finite width.* One day meeting on Automated Techniques for Feature Extraction from Imagery, NERC Unit for Thematic Information Systems, University of Reading, 1989.

POLAK E., *Computational Methods in Optimization.* Academic Press, New York, 1971.

PRATT W.K., *Digital Image Processing, 2nd Edition.* John Wiley & Sons, New York, 1991.

PRESS W.H., FLANNERY B.P., TEUKOLSKY S.A., & VETTERLING W.T., *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, Cambridge, 1986.

RABE S., Ph.D. thesis - *The Ising model as a model for texture in image analysis.* University of London, 1991.

RICHARDS J.A., *Remote Sensing Digital Image Analysis, An Introduction.* Springer-Verlag, Berlin, 1986.

RUIZ R.M., ELLIOTT D.A., YAGI G.M., POMPHREY R.B., POWER M.A., FARRELL K.W., LORRE J.J., BENTON W.D., DEWAR R.E., & CULLEN L.E., *IPL processing of the Viking orbiter images of Mars.* Journal of Geophysical Research, **82** (28), 1977.

SABINS F.F., *Remote Sensing : Principles and Interpretation, 2nd Edition.* W.H. Freeman, New York, 1987.

SACKS O., *The Man Who Mistook his Wife for a Hat.* Picador (Pan books), 1986.

SEZAN M.I., *A peak detection algorithm and its application to histogram-based image data reduction.* Computer Vision, Graphics, and Image Processing, **49** (1), 1990.

SHARMAN M., *Monitoring European agriculture by remote sensing : Action 4 : "Rapid estimates" of the JRC's pilot project.* Proceedings of the 15th Annual Conference of the Remote Sensing Society - "Remote Sensing for Operational Applications.", University of Bristol, 1989.

SINGH A., *Detecting changes in tropical forest cover due to shifting cultivation using LANDSAT MSS data.* Proceedings of the 10th Anniversary Conference of the Remote Sensing Society - "Satellite Remote Sensing - Review and Preview.", University of Reading, 1984.

SMITH G.M. & VAUGHAN R.A., *A heat source monitoring system and its application to strawburning in the UK.* Proceedings of a joint conference of the Photogrammetric Society, the Remote Sensing Society, & the American Society for Photogrammetry and Remote Sensing - "Spatial Data 2000.", Christ Church College, University of Oxford, 1991.

SPACEK L.A., *Edge detection and motion detection.* Image and Vision Computing, **4**, 1986.

STAVTSER A.L. & KARASEV O.I., *New methods and technologies for forecasting onshore and offshore oil and gas fields.* Proceedings of "Remote Sensing - an operational technology for the mining and petroleum industries.", Institute of Mining and Metallurgy, London, 1990.

STEFOULI M. & OSMASTON H.A., *The remote sensing of geological linear features using LANDSAT : matching analytical approaches to practical applications.* Proceedings of the 10th Anniversary Conference of the Remote Sensing Society - "Satellite Remote Sensing - Review and Preview.", University of Reading, 1984.

STONE R.J., *Application of remote sensing to material resource location in developing countries.* Proceedings of the 15th Annual Conference of the Remote Sensing Society - "Remote Sensing for Operational Applications.", University of Bristol, 1989.

SWAIN P.H. & DAVIS S.M. (editors), *Remote Sensing : The Quantitative Approach.* McGraw-Hill, New York, 1978.

WRIGHT W.D., *The Measurement of Colour, 4th Edition*, Adam Hilger, London, 1969.

XU R., XU H., YE S., LU H., & ZHANG L., *Geobotanical remote sensing in China.* Proceedings of “Remote Sensing - an operational technology for the mining and petroleum industries.”, Institute of Mining and Metallurgy, London, 1990.

ZHANG J. & MODESTINO J.W., *A model fitting approach to cluster validation with application to stochastic model-based image segmentation.* IEEE Pattern Analysis and Machine Intelligence, **12** (10), 1990.