First Quarterly Progress Report on CESDIS Contract S.C. 550-80

A Planetary Version of PC-McIDAS

Sanjay S. Limaye
Principal Investigator

Space Science & Engineering Center
University of Wisconsin-Madison
1225 West Dayton Street
Madison, Wisconsin 53706
(608)262-9541

15 October 1992
This is the first quarterly progress report on CESDIS (S.C. 550-80). Previous work performed in the first year of the three year program was performed under NASA contract Contract NAS5-31347.

The primary accomplishment in this quarter was the completion of the multispectral classification capability within the McIDAS-X environment that was begun in the previous year. This work also resulted in the completion of Master of Science Degree requirements for Mr. Gerald R. Peltzer in the Department of Civil and Environmental Engineering, University of Wisconsin-Madison.

A user manual to run the suite of classification programs and a copy of the M.S. thesis submitted by Mr. Gerald R. Peltzer describing the classification scheme and the source code form the core of this quarterly progress report, and are attached herewith.

Work is continuing on creating new or adapting existing applications programs for planetary use under the McIDAS-X environment. It is anticipated that the planetary software will soon be tested in the McIDAS-OS/2 environment while continuing work on the incorporation of the NAIF library routines in the McIDAS applications. These activities which have been initiated in this reporting period will be described in the next progress report.
ATTACHMENTS

1. **The McIDAS Unsupervised Classification System Operations Manual.** Also contains the source (which forms Appendix B of the MS Thesis by Mr. G.R. Peltzer, below).

2. **A Comprehensive Unsupervised Clustering Technique for the Classification of Remotely Sensed Data.** Copy of the M.S. Thesis by Mr. G.R. Peltzer submitted to the Civil and Environmental Engineering Department, University of Wisconsin-Madison, 51 pp with two appendices, 1992. (Appendix B contains source code which is also incorporated into the Operations Manual (above), and is not reproduced here so as to avoid duplication in the report as it is also included in the Operations Manual included above.)
1. **The McIDAS Unsupervised Classification System Operations Manual.**
The McIDAS Unsupervised Classification System

Operations Manual

Gerard Peltzer

UW-Madison

Space Science and Engineering Center
1.0 INTRODUCTION

An original procedure to classify multispectral images into spectrally similar categories in an automatic, or unsupervised, manner has been developed at the Space Science and Engineering Center of the University of Wisconsin-Madison. This procedure has been integrated into the Man-Computer Image Data Access System (McIDAS) package and provides an accurate, adjustable, and easy to use process for the extraction of thematic information from multispectral data. The package presented consists of routines for image classification and analysis of classification output. It also includes routines which aid in comparing output with that of other classification methods.

A full description of multispectral classification and details on the development and testing of the algorithms used in the McIDAS classification system can be found in:


**Contents of the system**

The McIDAS classification system developed by the author consists of a set of original routines and existing functions that allow the user to classify and analyze multispectral images within the McIDAS system. In this new system, there are essentially three classification routines (USCLAS, MDCLAS, and ISO.FOR) and a routine for visualizing the output classes of a classification (ELLIPS). In addition, the McIDAS system has been enhanced by adding a utility function (COMBIN) and provides routines for displaying classifications and changing the output enhancement table in order to highlight or merge classes.

USCLAS is the unsupervised classification routine which classifies a multi-band McIDAS area automatically. MDCLAS is a Minimum Distance to
Mean classification routine which uses cluster statistics developed from a previous classification to classify an area. A good application for MDCLAS is in classifying a time series of images using a set of class statistics derived by USCLAS. ISO.FOR contains standard ISODATA subroutines for cluster merging and splitting which can be inserted into USCLAS to make a standard ISODATA routine. COMBIN is a utility function which combines several one-band McIDAS areas into a multi-band area to be used by USCLAS or MDCLAS. ELLIPS is a cluster visualization utility which displays plots of class statistics graphically.

2.0 USCLAS

The USCLAS routine performs an unsupervised classification in a recursive process based on the ISODATA method which classifies an image repeatedly, using a changing set of spectral clusters which evolve to represent the statistical classes present in the data. Throughout the classification, oversized clusters are split using a method analogous to analyzing mass moments of inertia, while poorly separated clusters are merged using Transformed Divergence as a decision function for class separability. The advantage of these methods is that they give the user an absolute index for cluster separation. The final number of classes produced is a function of the degree of class separation and the class shape desired by the user.

Running USCLAS

To begin an image classification, a one-byte multi-band McIDAS area must be first obtained. (USCLAS will classify a single-band image, but this amounts to performing a level slicing operation.) If the image is in the form of several single-band areas, then the user must first run the McIDAS COMBIN routine, which was developed by the author to make a multi-band image from
single-band components.

The next step is to run USCLAS. A help menu for the program can be obtained through the `HELP USCLAS` command. When running this or any other classification routine, the best approach is to switch to a separate output window, start the classifier, then switch back to the first output window. The USCLAS output, which includes progress updates every iteration, can then be monitored separately while other McIDAS routines are run.

USCLAS is run as a command in the McIDAS system, with parameters entered through the command line. The only required parameters are for the input and output areas, though a number of adjustable parameters may be entered. When first viewing the USCLAS help menu, the assortment of parameters might seem confusing. This should not be disconcerting, as the program is designed for ease of use and should produce good classification results using defaults. The additional parameters, however, can be very useful in gaining added control over the output of the classifier. The following is a list of the command line parameters of USCLAS as seen through the help menu:

```
C===================================================================
C ? USCLAS -- SPECTRALLY CLASSIFY A MULTI-BAND IMAGE
C ? \--- UNSUPERVISED MULTISPECTRAL IMAGE CLASSIFICATION \---
C ?
C ? USCLAS INAREA OUTAREA
C ? PARAMETERS:
C ? INAREA - AREA TO BE CLASSIFIED
C ? OUTAREA - FINAL CLASSIFIED AREA WITH ~M CLASSES.
C ?
C ? KEYWORDS:
C ? BANDS: NUMBER OF AREA BANDS TO CLASSIFY,
C ? DEFAULT IS ALL BANDS IN AREA.
C ? USER DEFINES THE BANDS TO USE:
C ? BANDS= BAND1 BAND2 ... BANDN
C ? CLASSES: INITIAL # OF CLASSES PRESENTED.
C ? CLASSES= M \ DEFAULT IS 100
C ? SKIP: SKIP FACTOR FOR ITERATIVE
C ? CLASSIFICATION. \ NUMBER OF ROWS
C ? AND COLUMNS SKIPPED. MUST BE \< 8.
C ? \ SKIP= N \ DEFAULT IS 4
C ? ITER: MAXIMUM ALLOWED ITERATIONS
```
The first optional parameter is \textit{nbands}, the number of bands. This parameter allows the user to run a classification using either all of the bands of an image, or only selected bands. The maximum number of bands is currently set at seven. The selection of the first three bands used is important in that it will determine the output color scheme of the classification.

The next parameter, \textit{classes}, allows the user to choose the initial number of classes for the classifier to work with. The initial number of classes has an important and predictable effect on the classification output. Increasing the number of initial classes in USCLAS generally produces a larger number of output classes that meet all the other requirements of the classification. This effect is one of diminishing returns, however, and increasing the initial number of classes beyond a certain threshold ceases to change the output classification. A high value of around 200 initial classes has been found to be most effective for finding as many distinct classes as possible in the data; however, values greater than 100 generally work well.

The \textit{skip} parameter sets the classifier to analyze data only at every \textit{n}th row and column. This speeds up the clustering by a factor of \textit{n}^2, but at a cost of
a poorer classification. Unlike the ERDAS approach, the skip factor in USCLAS is used until convergence is reached, at which point the whole image (skip set at 1) is classified. A good approach to getting to a desired type of output quickly is to run USCLAS classifications with a large skip factor until the most useful program settings are found, then run USCLAS on the whole image to produce the final output. Increasing the skip factor generally reduces the output number of classes slightly as there is less information used in the analysis.

The parameter *iter*, the maximum number of iterations allowed, has a default of 35. This should be enough in most cases, except for data where the image bands are not registered well to each other. In general, changing the other parameters in USCLAS makes the classifier either more or less demanding, which is reflected in an increase or decrease in the required iterations. If USCLAS goes to the maximum number of iterations, it terminates and classifies the image using the last set of classes.

The *merge* and *split* parameters work as described earlier, and are developed in detail in the thesis. For merging, a transformed divergence value is entered. This value is the minimum transformed divergence allowed between cluster pairs. Although values of 1700 or greater are considered good theoretically, practical values of around 1400 seem to show better results. Increasing the the *merge* parameter in USCLAS makes for a more demanding classifier which reduces the output number of classes and increases the iterations required. Because of the predictable nature of this criterion, adjusting the merging parameter is the best way to fine-tune the number and characteristics of the output classes desired.

For cluster splitting, a value for the ratio of the largest to smallest cluster dimensions is entered. The default ratio of three allows clusters to be essentially up to three times as long as they are wide. In general, entering a
large `split` parameter produces fewer output classes. Due to the nature of the criterion however, the splitting factor is not recommended as a means to adjust the number of output classes, but rather as a check to ensure that class sizes are well behaved. For cluster splitting as well as merging, an input of zero disables the routine.

The `null` parameter tells the MDM classifier how to handle data in which there are areas of no data for some or all of the bands. In the case where some bands are zero, USCLAS is generally forced to develop an entirely new set of classes for that area which have a zero mean in the empty bands. As this can be good or bad depending on the application, the null parameter can be set to classify pixels as zero if any one of the bands contains drop-outs, or only if all bands are empty. The classifier reserves the special class of zero for data drop out.

The `minpix` parameter is the key in defining the minimum number of pixels allowed in each cluster. In USCLAS, a cluster is defined as empty and is removed if there are fewer than the minimum number of pixels in that cluster. As many images contain noise pixels, setting the minimum number of pixels to a value greater than zero reduces the presence of small noise classes. Setting this parameter to a value too high can eliminate small classes of useful information, however.

The `stretch` parameter in USCLAS allows the user to change the output appearance of the classification. In USCLAS, the output classes are given colors (described in the classification's enhancement table file) which try to match the 'colors' of the class mean vectors in the first three bands of the data. To make a more pleasing display, the colors are obtained from a histogram stretch of the mean vectors. The `stretch` parameter defines the type of stretch used: either linear or histogram equalization may be used.
The speed of USCLAS on an IBM RISC workstation is very fast relative to an ISODATA routine running on a PC-compatible computer, but still requires a significant amount of time. As an example of the time required, a full USCLAS classification of a 512 by 512 image containing six bands takes from twenty to forty minutes to classify, depending on the number of iterations required. The same scene processed through the ERDAS ISODATA on a 33MHz 80486 based PC-compatible computer can require over five hours to complete. The large difference in speed is due mainly to the differences in processing power and disk access time between the two computers, but there are also some speed enhancements incorporated into USCLAS. The standard ISODATA routine is slowed considerably by its need to access the disk repeatedly. For every iteration, it must read in the image, read in the old classification for comparison, then write out the new classification. This heavy disk access creates a scenario where the classifier works at the speed of the disk drive, no matter how fast the processor. In USCLAS, the output classification is kept entirely in memory until the final write, thus reducing the three disk operations to only one disk read each iteration.

**USCLAS output**

The USCLAS program produces four separate output files which contain the information from the classification. The display part of the output consists of a McIDAS area containing classified values for the image classified and an .ET enhancement table to provide colors for these classes. The other two files are .SIG and .LOG files which contain the statistics of the classification. The .LOG file is a log of the classification's operations through every iteration, including detailed descriptions of changing cluster means and statistics. All these files consist of the McIDAS area number of the classified image plus the relevant extension.
3.0 MDCLAS

After processing, the classification output may also be used as input for another McIDAS program. The MDCLAS program can use the class statistics to classify other multi-band McIDAS areas quickly, using a spectral minimum distance to mean algorithm. Its premier use is to classify a large number of images in a time series with the classes obtained from a control image of the series.

MDCLAS is run similarly to USCLAS, with parameters entered through the command line. The required parameters are for the input and output areas, plus the area number for the class statistics to be used.

4.0 ELLIPS

The .SIG file may be used to visualize the positions of the output classes in the pattern space of the classification through the use of the cluster visualization program named ELLIPS. ELLIPS is so named because it plots elliptical plots of the statistical distribution of the data in each cluster. This program also plots only the locations of the mean vectors of the data if requested, but the elliptical plots give a much better indication of the spread and overlap of classes.

ELLIPS draws a series of elliptical plots, going through all the possible combinations of axes two at a time. In the ELLIPS program, a plot connecting the four points along the principal axes of the ellipse is performed for every class in the output. ELLIPS uses two standard deviations as a default value, but will also accept any other value as an input parameter. The only input Ellipse requires is the area number of the classification. A standalone version of ELLIPS which does not run under McIDAS is available, and is called ELLIPSE.
5.0 VISUALIZING A CLASSIFICATION

Once USCLAS or MDCLAS has finished running, the user may display the output area in the image window, using the command \textit{EU REST nnnn} to color the classification image. (nnnn is the output area number; all the USCLAS output files are written as this number plus an extension.) Class colors may be modified with the \textit{EU} command. Classes may also be merged by making component clusters of an information class the same color. The output class numbers visible are doubled from their original values used in analysis. This is an unfortunate necessity because McIDAS-X has a seven bit color plane which limits the visualization of output colors. Output classes thus go as 0, 2, 4, ... and odd numbers are not used.

To aid in the analysis of output classes, the ELLIPS program may be called to create a window containing a plot of statistical ellipses or class means. Several ELLIPS plots may be run at once; the only limitation is with available screen area for viewing. ELLIPS plots do not present doubled classes, and the user must remember to multiply by two when comparing with screen classes.
Some trial McIDAS areas which may be classified by USCLAS:
Gerry Peltzer, 11/20/1992

<table>
<thead>
<tr>
<th>area</th>
<th>ss</th>
<th>yd</th>
<th>d</th>
<th>hh</th>
<th>mm</th>
<th>ss</th>
<th>lcor</th>
<th>ecor</th>
<th>lr</th>
<th>er</th>
<th>zr</th>
<th>lsiz</th>
<th>esiz</th>
<th>z</th>
<th>bands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>----</td>
<td>----</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>------</td>
<td>------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>------</td>
<td>------</td>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>Thesis AVHRR scene of Kuwait, smoke:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2230</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1234</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2231</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2232</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2233</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2234</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example classifications of 2230, see &quot;2230classes&quot; for details:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3063</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3064</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3065</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3066</td>
<td>60</td>
<td>91054</td>
<td>43100</td>
<td>661</td>
<td>1178</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classifications produced by ERDAS ISODATA:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3098</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3099</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>640</td>
<td>552</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVHRR of Kuwait, smoke:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2351</td>
<td>61</td>
<td>91235</td>
<td>104500</td>
<td>2401</td>
<td>1051</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>580</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2352</td>
<td>61</td>
<td>91235</td>
<td>104500</td>
<td>2401</td>
<td>1051</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>580</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2353</td>
<td>61</td>
<td>91235</td>
<td>104500</td>
<td>2401</td>
<td>1051</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>580</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2354</td>
<td>61</td>
<td>91235</td>
<td>104500</td>
<td>2401</td>
<td>1051</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>580</td>
<td>800</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVHRR of Kuwait, smoke:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2501</td>
<td>62</td>
<td>91316</td>
<td>43600</td>
<td>501</td>
<td>751</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>480</td>
<td>640</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2502</td>
<td>62</td>
<td>91316</td>
<td>43600</td>
<td>501</td>
<td>751</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>480</td>
<td>640</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2503</td>
<td>62</td>
<td>91316</td>
<td>43600</td>
<td>501</td>
<td>751</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>480</td>
<td>640</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2504</td>
<td>62</td>
<td>91316</td>
<td>43600</td>
<td>501</td>
<td>751</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>480</td>
<td>640</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVHRR of Wisconsin, Iowa, and Minnesota together:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2701</td>
<td>61</td>
<td>92295</td>
<td>204700</td>
<td>3628</td>
<td>9400</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1500</td>
<td>1200</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2702</td>
<td>61</td>
<td>92295</td>
<td>204700</td>
<td>3628</td>
<td>9400</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1500</td>
<td>1200</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2703</td>
<td>61</td>
<td>92295</td>
<td>204700</td>
<td>3628</td>
<td>9400</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1500</td>
<td>1200</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2704</td>
<td>61</td>
<td>92295</td>
<td>204700</td>
<td>3628</td>
<td>9400</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1500</td>
<td>1200</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Madison TM scene, 3 bands:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3120</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>600</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3121</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>600</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wisconsin AVHRR scene:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3131</td>
<td>62</td>
<td>92232</td>
<td>135900</td>
<td>1221</td>
<td>816</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>500</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3132</td>
<td>62</td>
<td>92232</td>
<td>135900</td>
<td>1221</td>
<td>816</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>500</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3133</td>
<td>62</td>
<td>92232</td>
<td>135900</td>
<td>1221</td>
<td>816</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>500</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3134</td>
<td>62</td>
<td>92232</td>
<td>135900</td>
<td>1221</td>
<td>816</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>500</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spot scene of Jakarta, Indonesia:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3150</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>601</td>
<td>801</td>
<td>1</td>
<td>1234</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3151</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>601</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3152</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>601</td>
<td>800</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Landsat agricultural scene of Wisconsin:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3400</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>600</td>
<td>600</td>
<td>1</td>
<td>1234</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example classifications, see &quot;3400classes&quot; for details:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3401</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>600</td>
<td>600</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3402</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>600</td>
<td>600</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3403</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>600</td>
<td>600</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3414</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>600</td>
<td>600</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3415</td>
<td>0</td>
<td>92000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>600</td>
<td>600</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
McIDAS utilities:

COMBIN - Combines several one-band areas into a multi-band area with one byte data. The navigation of the first area is copied.

LISTHEAD - Lists the contents of the header of an area in integer form.

FDEST - Subroutine to print output to a file (a mainframe routine).

ELLIPS - Draws signature ellipses of classes generated by a classification routine. Inputs can be in .SIG or .SBD (ERDAS) format.

McIDAS classifiers:


MDCLAS - Uses classes developed by another classifier to classify a multi-band area. Uses minimum distance to mean.

ISO.FOR - Contains cluster splitting and merging routines for the standard (Tou and Gonzalez, 1974) ISODATA.

McIDAS to ERDAS file translation programs:

ATOERD - Converts McIDAS areas to ERDAS .lan or .gis files, converting McIDAS .ET color look-up table files to ERDAS .trl color look-up files if a .gis file is desired.

ERDTOA - Converts ERDAS .lan or .gis files to McIDAS areas, converting ERDAS .trl files to McIDAS .ET files if a .gis file is converted.

SBDSIG - Translates an ERDAS .sbd binary classification signatures file into a text readable .SIG file for use in McIDAS classifiers. Also translates .SIG files into .SBD format for use with SIGMAN in ERDAS.
SUBROUTINE main

C COMBIN -- MAKE A MULTI-BAND AREA FROM SEVERAL SINGLE BAND AREAS.
C MAY ALSO BE USED TO REDUCE 2-BYTE DATA TO 1-BYTE DATA.

C USAGE --
C COMBIN OUTAREA BANDS-AREA1 AREA2...

C PARAMETERS:
C OUTAREA - AREA WITH N BANDS.
C BANDS: AREA BANDS TO REGROUP.
C USER DEFINES THE BANDS TO USE;
C AREA1 WILL BE BAND1 OF OUTAREA, AREA2
C WILL BE BAND2 OF OUTAREA, ETC.
C CUTOFF: PERCENT OF DATA TO CUTOFF IN 2BYTE COMPRESSION.
C DEFAULT = 1.0% OF DATA
C BANDWIDTH: OUTPUT RANGE FOR 2 BYTE RAW DATA, FROM 2 TO 255.
C DEFAULT = 255
C DATA: INPUT FORMAT OF DATA: BRIT, RAW
C DEFAULT = RAW

Written by Gerard Peltzer
UM SSEC
Ver 10/29/1992

MXBANDS sets max number of bands allowed in input image:
MAXRC sets maximum number of rows and cols in image:

IMPLICIT CHARACTER*12 (C)
IMPLICIT REAL*8 (D)
PARAMETER (MXBANDS=5)
PARAMETER (MAXRC=3584)
PARAMETER (MAXCUL=3584)
PARAMETER (MAXROW=1024)
PARAMETER (STBANDS=4)
PARAMETER (MAXOUT=17920)
PARAMETER (MAXX=32768)
PARAMETER (MAXB=255)
PARAMETER (XMCUTOF=1.0)
CHARACTER*3 CDDATA
PARAMETER (CDDATA='RAW')

CHARACTER*80 CMES
CHARACTER*4 CTYPE(2), CDATA
INTEGER*2 BUFFER(0:MAXRC), OUTDAT(0:MAXOUT), DATT(0:MAXOUT)
INTEGER IDIR(64), JDIR(64), ABANDS, NBANDS, NRWS, NCOLS
INTEGER I, J, ROW, COL, BAND, PACKING, DATATYPE
INTEGER CHECK, MRM1, NCM1, NBM1, IBANDWIDTH
INTEGER VAL, DOG, CAL, LEV, START1, START0
INTEGER*4 ITYPE(2), INAREA(MXBANDS+1), OTAREA
INTEGER*4 IBUF(10), PBUF(100), ICAL(128), IBAND
INTEGER*4 MAX(MXBANDS), MIN(MXBANDS), NAVOFF, NAVARR(256)
REAL*8 CDCAVG(MXBANDS), SCALE(MXBANDS), HIST(0:MAXBANDS,0:255)
REAL*8 SMX(MXBANDS), SMM(MXBANDS)

CALL SDEST('BEGIN':',0)
OTAREA=IP0(1,0)
IF (OTAREA .LT. 1) THEN
CALL SDEST('BAD OUTPUT AREA SPECIFIED',0)
RETURN
ENDIF
WRITE(CMES,'(1X,A12,I4)') 'Output Area: ', OTAREA
CALL SDEST(CMES,0)

NBANDS=NBANDS('BANDS')
IF (NBANDS.EQ.0) THEN
WRITE(CMES,'(1X,A10)')
C 'USER SPECIFIED ILLEGAL NUMBER OF BANDS'
CALL SDEST(CMES,0)
RETURN
ENDIF
DO 10 I = 1, NBANDS
INAREA(I) = IKBW('BANDS',I,1)
10 CONTINUE
WRITE(CMES,'(A12,I5)') 'Bands to combine: ', NBANDS
CALL SDEST(CMES,0)
IF (NBANDS.GT.0) THEN
WRITE(CMES,'(1X,A40)')
C 'USER SPECIFIED ILLEGAL NUMBER OF BANDS'
CALL SDEST(CMES,0)
RETURN
ENDIF

C Read in data cutoff:
DCUTOF=DKW('CUTOFF',1,XMCUTOF)/100.
C Check lter less than maximum:
IF (DCUTOF.GT.1.0.OR. DCUTOF.LT.0.0) THEN
WRITE(CMES,'(A40)') ' BAD CUTOFF VALUE, USING DEFAULT' CALL SDEST(CMES,0)
DCUTOF = XMCUTOF
ENDIF
WRITE(CMES,'(A16,F7.2)') ' Cutoff value = ', DCUTOF*100.
CALL SDEST(CMES,0)

C Read in data bandwidth:
IBWIDTH=IKBW('BANDWIDTH',1,MAXB)
C Check lter less than maximum:
IF (IBWIDTH.GT.MAXB) THEN
WRITE(CMES,'(A40)') ' BAD BANDWIDTH VALUE, USING DEFAULT' CALL SDEST(CMES,0)
IBWIDTH = MAXB
ENDIF
WRITE(CMES,'(A26,I4)') ' Output bandwidth value = ', IBWIDTH CALL SDEST(CMES,0)

C Read in data type read:
CDDATA=CKBW('DATA',1,CDDATA)
C Check data type:
19 CONTINUE
IF (CDDATA.EQ.'RAW' .OR. CDDATA.EQ. 'raw') THEN
DATATYPE = RAW
GOTO 29
ENDIF
IF (CDDATA.EQ.'BRIT' .OR. CDDATA.EQ. 'brit') THEN
DATATYPE = BRIT
GOTO 29
ENDIF
WRITE(CMES,'(A40)') ' USER SPECIFIED BAD DATA TYPE'
CALL SDEST(CMES,0)
CDDATA = CDDATA
WRITE(CMES,'(A23,A31)') 'USING DEFAULT = ',CDDATA CALL SDEST(CMES,0)
GOTO 19
29 CONTINUE
WRITE(CMES,'(1X,A11,A6,A6)') 'Reading in ',CDDATA, ' data.' CALL SDEST(CMES,0)

C Open area to see how big, number of bands:
CALL SDEST(')',0)
WRITE(CMES,'(1X,A20)') '---Image Contents---'
CALL SDST(CMES,0)
DO 20 BAND = 1, NBANDS
   CALL HHOBIG(INAREA(BAND),NROWS,NCOLS)
   WRITE(CMES,'(1X,A7,15)') 'Area: ', INAREA(BAND)
   CALL SDST(CMES,0)
   WRITE(CMES,'(1X,A7,15)') 'Rows ', NROWS
   CALL SDST(CMES,0)
   WRITE(CMES,'(1X,A7,15)') 'Cols ', NCOLS
   CALL SDST(CMES,0)
   IF (NROWS .GT. MAXROW) .OR. (NCOLS .GT. MAXCOL) THEN
      CALL SDST('AREA TOO BIG TO PROCESS - EXITING!',0)
      RETURN
   ENDIF
   CALL ONPRA(INAREA(BAND))
   CTYPE(1) = 'UNIT'
   ICTYPE(1) = LIT(CDATA)
   CTYPE(2) = 'SPAC'
   ICTYPE(2) = 2
   CALL ARAOPT(INAREA(BAND),2,CTYPE,ITYPE)
   CALL READI(INAREA(BAND),IDIR)
   IMBAND=WINT(ALOGIC(REAL(IDIR(19)))/ALOGIC(2.1)+1)
   WRITE(CMES,'(A12,13,A11,13)') 'Input band',IMBAND,
   C
   ABANDS= JDIR(14)
   IF (ABANDS .GT. IBUF) THEN
      WRITE(CMES,'(1X,A20,15)') 'Output to',BAND
   ELSE
      CALL SDST(CMES,0)
      CALL SDST('PROGRAM WILL USE 1 BAND ONLY',0)
   ENDIF
   CALL SDST('NOT ENOUGH BANDS TO PROCESS - EXITING!',0)
   RETURN
ENDIF

IF (BAND .EQ. 1) THEN
   CALL Makara(INAREA,IDIR)
   NAVOFF = IDIR(35)
   CALL MOVCM('AREA COMBINATION',IDIR(25))
   IDIR(14) = NBANDS
   IDIR(19) = 0
   DO 15 I = 1, NBANDS
      IDIR(19) = IDIR(19) + 2**(I-1)
   CONTINUE
   IF (IDIR(36) .NE. 0) THEN
      VAL = IDIR(15)-IDIR(49)-IDIR(50)-IDIR(51)
   ELSE
      VAL = 0
   ENDIF
   LEV = IDIR(51)
   DOC = IDIR(49)
   CALL = IDIR(50)
   WRITE(CMES,11) 'VAL',VAL,'DOC',DOC,'CAL',CAL,'LEV',LEV
   CALL SDST(CMES,0)
   CALL MAKARA(INAREA,IDIR)
   WRITE(CMES,11) 'FORMATE(4(1X,A3,',':,1X,15))
   C
   "Copy DOG, CAL sections to new data:
   STARTI = IDIR(15)
   STARTO = STARTI - LEV - VAL + 8
   STARTO = 8
   Modify LEV section to read bands in new area:

C
C  buf(1 and 2) contain LEV information:
   IDIR(31) = 8
   IDIR(50) = 0
   IDIR(49) = 0
   INFACKING = IDIR(11)
   IF (INFACKING .EQ. 2 .AND. CDATA .EQ. 'RAW') THEN
      PACKING = 2
   ELSE
      PACKING = 1
   ENDIF

C
C  JDIR(11) = 1
   IDIR(34) = 768
   IDIR(35) = 256
   IDIR(63) = 0
   IDIR(52) = LIT('VSR')
   IF (IDIR(3) .EQ. 55) IDIR(52) = LIT('MSAT')
   C
   C  IDIR(3) .EQ. 60 .OR. (IDIR(3) .EQ. 61) .OR.
   C
   C  IDIR(52) = LIT('TIRO')
   C
   CENDIF
   IDIR(53) = LIT('BRIT')
   DO 22 I = 1, 8, 1
      IF (I .LT. NBANDS) THEN
         IBUF(I) = I
      ELSE
         IBUF(I) = 0
      ENDIF
   CONTINUE
   CALL PACK8(IBUF,IBUF)
   IF (STARTI .LT. 0) STARTI = 0
   IF (STARTO .LT. 0) STARTO = 0
   Start position in int*4 is quarter of bytes:
   IDIR(15) = STARTO
   STARTI = STARTI/2
   STARTO = STARTO/2
   CALL MAKARA(INAREA,IDIR)
   CALL MAKARA(INAREA,IDIR)
   CALL MAKARA(INAREA,IDIR)
   ENDIF

C
C  Loop over all input images, develop averages for each band:
   SKIPP = 2
   SCALE(BAND) = MAXB*1./MAX2B
   MAX(BAND) = 0
   MIN(BAND) = 10000
   SMAX(BAND) = 0.
   SMIN(BAND) = 10000.
   CCDVAG(BAND) = 0.0
   DO 86 J = 0, 255
      HIST(BAND,J) = 0
   CONTINUE
   DO 86 ROW = 0, NRMI, SKIPP
   CALL REDARA(INAREA(BAND),ROW,0,MAXRC,IMBAND,BUFFER)
   CALL 8S COL = 0, NCOL, SKIPP
   CALL CCDVAG(BAND) = CCDVAG(BAND) + BUFFER(COL)
   IF (BUFFER(COL) .GT. MAX(BAND)) MAX(BAND) = BUFFER(COL)
   IF (BUFFER(COL) .LT. MIN(BAND)) MIN(BAND) = BUFFER(COL)
   IF (INFACKING .EQ. 2) THEN
      DSpot = BUFFER(COL)*SCALE(BAND)+0.5
      HIST(BAND,DSpot) = HIST(BAND,DSpot) + 1
   ENDIF
   C
   C  IF (INFACKING .EQ. 1) THEN
DSPOP=BUFFER(COL)+0.5  
HIST(BAND,DSPOP)=HIST(BAND,DSPOP)+1  
ENDF

CONTINUE

CCDAVG(BAND)=CCDAVG(BAND)/(NROWS*NCOLS/SKIPF**2)
WRITE(CMES,'(1X,A18,I2,A2,F8.1)')'Average value of ',BAND,
C
'=',CCDAVG(BAND)
CALL SDEST(CMES,0)

Check to see that the DATA BRIT option is working:
IF (CDA0A .EQ. 'BRIT' .AND. CCDAVG(BAND) .GT. MAXB1) THEN
PACKING = 2
WRITE(CMES,'(A41)')'Failed to get BRIT data, using raw data'
CALL SDEST(CMES,0)
ENDIF

Use histogram to find smin, smax for stretching:
IF (PACKING .EQ. 2) THEN
J = 0
SUM = 0
CONTINUE

SUM = SUM + HIST(BAND,J)
IF (J .GE. 255) GOTO 56
IF (SUM .LE. (NROWS*NCOLS)*DCUTOF/SKIPF**2) THEN
J = J + 1
GOTO 55
ENDIF

J = 255
SUM = 0

CONTINUE
SUM = SUM + HIST(BAND,J)
IF (J .GE. 0) GOTO 57
IF (SUM .LE. (NROWS*NCOLS)*DCUTOF/SKIPF**2) THEN
J = J - 1
GOTO 57
ENDIF

IF (SMAX(BAND) .LT. J) SMAX(BAND) = J
Smax and smin are now a 1 byte range:
IF (SMAX(BAND) .LT. MAXB1) SMAX(BAND) = SMAX(BAND)+1
IF (SMIN(BAND) .GT. 0) SMIN(BAND) = SMIN(BAND)-1
Expand smax and smin to full 2 byte range:
SMAX(BAND) = SMAX(BAND)/SCALE(BAND)
SMIN(BAND) = SMIN(BAND)/SCALE(BAND)
IF (SMAX(BAND) .LT. MAX(BAND)) SMAX(BAND) = MAX(BAND)
IF (SMIN(BAND) .LT. MIN(BAND)) SMIN(BAND) = MIN(BAND)
Scale to output bandwidth:
SCALE(BAND) = IBWIDTH*1./(SMAX(BAND)-SMIN(BAND))
WRITE(CMES,'(1X,A0,I9,F9.1)')'Statistic max = ',SMAX(BAND)
CALL SDEST(CMES,0)
WRITE(CMES,'(1X,A0,I9,F9.1)')'Statistic min = ',SMIN(BAND)
CALL SDEST(CMES,0)
ENDIF
WRITE(CMES,'(1X,A0,I9)')'Max data read = ',MAX(BAND)
CALL SDEST(CMES,0)
WRITE(CMES,'(1X,A0,I9)')'Min data read = ',MIN(BAND)
CALL SDEST(CMES,0)
CALL SDEST(' ',0)

DO 60 ROW = 0, NRM1, 1
DO 21 BAND = 1, NBANDS
C
C REDARA (AREA, LINE, START_ELEM, #ELEMENTS, BAND, IARRAY)
CALL REDARA(INAREA(BAND), ROW, 0, MAXCOL, IMBD, BUFFER)
M = BAND - 1
DO 75 COL=0, NCMI, 1
IF (PACKING .EQ. 2) DATT(M+STARO+2) =
C
(BUFFER(COL)-SMIN(BAND))*SCALE(BAND)
IF (PACKING .EQ. 1) DATT(M+STARO+2) = BUFFER(COL)
M = M + NBANDS
75 CONTINUE
21 CONTINUE
C
C Write out to output file:
CALL PACK2(MAXOUT, DATT, DATT)
C
Header items appended (new lev):
CALL MOVC(8, IBUF, 0, DATT, 0)
C
C Write file out to output file name
CALL WRATA(OTHAREA, ROW, DATT)
60 CONTINUE
DO 62 BAND = 1, NBANDS
CALL CISARA(INAREA(BAND))
62 CONTINUE
CALL CISARA(OTHAREA)
CALL STAMP(OTHAREA)
CALL SDEST('----Finished Combin----',0)
RETURN
END

//-------------------------------
// Loop over rows and columns, convert input to output:
//
CALL SDEST(' ',0)
SUBROUTINE MAINO
  LISTHEAD -- LIST THE HEADER OF A MULTI-BAND IMAGE
  LISTHEAD INAREA

PARAMETERS:
  INAREA - AREA TO BE LISTED
  OUTPUT LISTING OF HEADER SAVED TO FILE --> INAREA.HED

Written by Gerard Peltzer, UW SSEC 11/14/1991
For use on IBM RISC 6000 MCIDAS-AIX

---------------------------------------------------------------------
INTEGER IDIR(64)
IMPLICIT REAL*8 (D)
IMPLICIT CHARACTER*12 (C)
INTEGER*4 INAREA, NROWS, NCOLS, I
CHARACTER*8 TABFIL
CHARACTER*70 CMES

INAREA=IP(1,0)
WRITE(CMES,'(1X,A12,4x)') ' AREA: ', INAREA
CALL SDEST(CMES,0)
IF (INAREA .LE. 0 .OR. INAREA .GT. 50000) THEN
  WRITE(CMES,'(1X,A15,1X,16x)') 'INVALID AREA: ', INAREA
ENDIF
RETURN
CALL LWD(TABFIL)
CNEWOUT = CFI(INAREA)
TABFIL(1:16) = CNEWOUT(9:12)
TABFIL(5:8) = '.HED'
WRITE(CMES,'(A5,1x,A8)') 'HEADER FILE: ', TABFIL
CALL SDEST(CMES,0)
CALL FDEST(TABFIL,CMES)
CALL OPNA(INAREA)
CALL AROAPT(INAREA,1,'SPAC',4)
CALL READD(INAREA,IDIR)
NROWS = IDIR(9)
NCOLS = IDIR(10)
WRITE(CMES,'(2A10,5,3x)') ' ROWS: ', NROWS, ' COLUMNS: ', NCOLS
CALL SDEST(CMES,0)
CALL FDEST(TABFIL,CMES)
DO 6 I = 1, 64
  WRITE(CMES,'(1X,I2,1x,I15)') I, IDIR(I)
  CALL SDEST(CMES,0)
  CALL FDEST(TABFIL,CMES)
CONTINUE
CALL SDEST('-----DONE-----',0)
CALL FDEST(TABFIL,'-----DONE-----')
RETURN
END
#include <stdio.h>

#if defined(sun)
  void fdest_(char cfile[8], char ctext[70])
#else
  void fdest(char cfile[8], char ctext[70])
#endif

/*
 * FDEST - Sends text output to a disk file.
 * Emulates the FDEST function on other McIDAS versions.
 * Compile with "fx fdest cli" in McIDAS src directory.
 * For use on IBM RISC 6000 MCIDAS-AIX.
 *----------------------------------------------------------*/

/*
 * Print a line of text (CTEXT) to file (CFILE): 
 *----------------------------------------------------------*/

char fname[9];
FILE *outfil;

strcpy(fname, cfile, 8);
fname[8] = '\0';
while ((outfil = fopen(fname, "a+")) == NULL)
  printf("\n Error opening FDEST output file \s \s \", fname, ctext);
  return;

fprintf(outfil, "%s\n", ctext);
fclose(outfil);
return;
SUBROUTINE MAINO

? ELLIPS - Draw signature ellipses from a USCLAS classification

? ELLIPS INAREA

? Parameters:

? 

? LCLASS - lowest class # to display
? HCLASS - highest class # to display
? MEAN - 1 to display class means only
? STD - 2 is default.
? 

? Defaults are 0 to largest class present, ellipses displayed.

Compilation information:

This program uses ellip.c and graphics libraries contained
in glib.o. To compile, use:

xlc ellip.c $2 -c -gsource -D_POSIX_SOURCE
ar r /u/mcidas/lib/libmcidas.a ellip.o
rm ellip.o
ar r /u/mcidas/lib/libmcidas.a glib.o
fx ellips 1

Written by Gerard Peltzer

UW SSEC 3/6/1992

PARAMETER (MMEAN=0)
PARAMETER (MLCLAS=0)
PARAMETER (MHCLASS=127)
PARAMETER (MSTD=2.0)

IMPLIED CHARACTER*12 (C)
IMPLIED REAL*8 (D)

CHARACTER*70 CMES
CHARACTER*9 INFILE
CHARACTER*12 CTEMP
INTEGER*4 INAREA, LCLASS4, HCLASS4, MEAN4
INTEGER*4 LCLASS, HCLASS, MEAN
REAL*4 STD
COMMON/TFILE/ LOGFIL

CALL SDEST('BEGIN', '0')

C Open input text file:

INAREA=IPF(1,0)

IF (INAREA .LE. 0 .OR. INAREA .GT. 50000) THEN
WRITE(CMES,'(1X,A15,1X,16)') 'INVALID AREA: ', INAREA
CALL SDEST(CMES,0)
RETURN
ENDIF

C Open output text file, use FDEST routine to print data to logfil:

CTEMP = CFI(INAREA)
INFILE(1:4) = CTEMP(9:12)
INFILE(5:9) = 'SIGO'
WRITE(CMES,'(A14,1X,A8)') 'INFILE ', INFILE
CALL SDEST(CMES,0)

C Read in LCLASS:

LCLASS4=IKWP('LCLASS',1,MLCLAS)
LCLASS = LCLASS4

C Check classes greater than 0, less than MXCLAS:

IF (LCLASS .LT. 0 .OR. LCLASS .GT. MHCLASS) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED POOR CHOICE'
CALL SDEST(CMES,0)
LCLASS = MCCLASS
WRITE(CMES,'(A25,15)') 'LOW CLASS TO BE USED - ', LCLASS
CALL SDEST(CMES,0)
ELSE
WRITE(CMES,'(A25,15)') 'HIGH CLASS TO BE USED - ', LCLASS
CALL SDEST(CMES,0)
ENDIF

C Read in HCLASS:

HCLASS4=IKWP('HCLASS',1,MHCLASS)
HCLASS = HCLASS4

C Check classes greater than 0, less than MXCLAS:

IF (HCLASS .LT. 0 .OR. HCLASS .GT. MHCLASS) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED POOR CHOICE'
CALL SDEST(CMES,0)
HCLASS = MCCLASS
WRITE(CMES,'(A25,15)') 'HIGH CLASS TO BE USED - ', HCLASS
CALL SDEST(CMES,0)
ELSE
WRITE(CMES,'(A25,15)') 'HIGH CLASS TO BE USED - ', HCLASS
CALL SDEST(CMES,0)
ENDIF

C Read in MEAN:

MEAN4=IKWP('MEAN',1,MMEAN)
MEAN = MEAN4

IF (MEAN .NE. 0 .AND. MEAN .NE. 1) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED POOR MEAN CHOICE'
CALL SDEST(CMES,0)
MEAN = MMEAN
WRITE(CMES,'(A25)') 'ELLIPSES WILL BE PLotted'
CALL SDEST(CMES,0)
ENDIF

C Read in STD:

STD=IKWP('STD',1,MSTD)

IF (STD .LE. 0.01 .OR. STD .GT. 10) THEN
INAREA = MSTD
ENDIF
WRITE(CMES,'(A25,F5.2)') 'USING STD of ', STD
CALL SDEST(CMES,0)

C Call ellipse program:

CALL ELLIP(INFILE,LCLASS,HCLASS,MEAN,STD)

CALL SDEST('-----DONE-----',0)
RETURN
END
# ellip.c

Routine to draw scatterplot ellipses of class signatures generated by spectral classification programs.

Ellipses are generated from mean and covariance matrices read in from a .SIG file. The four 'corner' points of the ellipse at the ends of the minor and major axes are plotted, with lines connecting them.

Multiple X-Windows graphs are produced which may be sent to printer via these commands (in the printscn program):

```bash
xwd -xy -bitmap -out scrdump.xpr scrdump
lpr -Preq scrdump.xpr
rm scrdump
rm scrdump.xpr
```

A list of the important variables used:

- lclass: Lowest class number to graph
- hclass: Highest class number to graph
- mean: Option to plot only class means (1)
- infile: Names for the image input file

Written by Gerard Peltzer

4/27/1992

```c
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define INPUT WIDTH 40 /* Number of chars in inputs. */
#define MAXCLASS 127
#define MAXSBD 16
#define MKNBS 7
#define SUCCESS 1
#define FAILURE 0
#define YES 1
#define NO 0
#define SBD 8
#define SIG 9
#define HISTBYTES 512*MKNBS

typedef struct Sig{
    char signame[32];
    long count;
    char blank[92];
    int mlnvector[MAXSBD];
    int maxvector[MAXSBD];
    float meanvector[MAXSBD];
    float covar[32];
}/*char histo[HISTBYTES];*/

typedef struct Sighead{
    char lannname[100];
    int classes;
    int bands;
    char blank[24];
} /* Signature;

void ellip(char infile[9], int *lclass, int *hclass, int *mean, double *std)
{
    /* Declarations of the functions used in this program:
    Signature *class;
    Sbdheader *head;
    int readsig(char *infile, Sbdheader *head, Signature *class);
    int ellipspts(char *infile, Sbdheader *head, Signature *class, int *lowclass, int *highclass, int *mean, double *std);
    */
    int type;

    /* Allocate memory for class: */
    if ((class = (Signature *) malloc(MXCLAS*sizeof(Signature)))
        == NULL)
        { printf("\nNot enough memory to allocate classes\n"); return; }

    /* Read in data from infile: */
    if (readsig(infile, head, class) == FAILURE )
        { printf("Quitting due to sig read errors...\n"); return; }

    if (ellipspts(infile, &head, class, lclass, hclass, mean, std)
        == FAILURE)
        { printf("Quitting due to point errors...\n"); return; }

    printf("\n...Ending...\n");
    free(class);
    return;
}
```
int max[MASSDB], min[MASSDB], smean[MASSDB][MAXCLAS];
long count[MAXCLAS];
float fcount, fmean, cova[32];

/* float covarm[COV]; */
float *covarm;
FILE *fspin;
char *triptr, nameline[32];
char dataline[80];
char firstline[100];
char moreline[80];

/* Allocate memory for covarm: */
if ((covarm = (float*)malloc(sizeof(float))) == NULL)
    { printf("No enough memory to allocate covariance
          \n"); return(1); }
/*
if (fopen(infile,"rt") == NULL)
    { printf("Error opening file %s", infile); return FAILURE; }
seek(fpin, 0L, SEEK_SET);
1 = 0;
.eof = NO;
/* File search: */
while (eof == NO)
{
/* Header information: */
/* First line */
for (i = 0; i < 100; i++) firstline[i] = 0;
for (i = 0; i < 100; i++) (*head).lannme[i] = 0;
if(fgets(firstline,100,fspin) == NULL) eof = YES; /* 1 */
/*printf("\nHeader of data file says: ");
puts(firstline);
print("\n");
if ((triptr = strtol(firstline,"\", -1) != NULL)
  { strcpy(triptr, "\n"); }
strcpy((*head).lannme, firstline);

/* Classes */
for (i = 0; i < 80; i++) dataline[i] = 0;
strcpy(dataline, "\n");
if(fgets(dataline,80,fspin) == NULL) eof = YES; /* 2 */
/*
  if (j < 1 || j == EOF) eof = YES;
  (*head).classes = classes;

/* Bands */
for (i = 0; i < 80; i++) dataline[i] = 0;
strcpy(dataline, "\n");
if(fgets(dataline,80,fspin) == NULL) eof = YES; /* 3 */
/*
  j = sscanf(dataline, "%d ",&bands);
  if (j < 1 || j == EOF) eof = YES;
  (*head).bands = nbands;
  for(i = 0; i < 24; i++) (*head).blank[i] = 0;

/* Develop structures for each signature: */

if(fgets(dataline,80,fspin) == NULL) eof = YES; /* 4 */
for (i = 0; i < 80; i++) dataline[i] = 0;

strcpy(dataline, \"\0\);
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 5 */
for (i = 0; i < MXXNDS; i++) max[i] = 0;
/*
  j = sscanf(dataline, "%d %d %d %d %d %d ",&max[0], &max[1],
  &max[2], &max[3], &max[4], &max[5], &max[6]);
  if (j < 1 || j == EOF) eof = YES; */
/* 2 blank lines */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 6 */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 7 */

/* Get Mean Values */
for (clas=0; clas< classes; clas++)
{
for (band=0; band< MXXNDS; band++)
    { smean[band][clas] = 0;
for (i = 0; i < 80; i++)
  dataline[i] = 0;
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 8 */
/* B+Cl */
j = sscanf(dataline, "%d %d %d %d %d %d ",&i,
  &count, &mean[0][clas], &mean[1][clas],
  &mean[2][clas], &mean[3][clas], &mean[4][clas],
  &mean[5][clas], &mean[6][clas]);
count[clas] = (long)fcount;
if (j < 1 || j == EOF) eof = YES; 
/* 1 blank line */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 9 */
/* Get Covariances */
for (clas=0; clas< classes; clas++)
{
/* 1 blank line */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 10 */
k = 0;
for (band=0; band< nbands; band++)
{
for (i = 0; i < 80; i++)
  moreline[i] = 0;
strcpy(moreline, \"\0\);
if (fgets(moreline,80,fpin) == NULL) eof = YES; /* 11 */
/* B+C+ */
if (k < 32)
  { printf("\nError in covar computation!");
    return FAILURE; }
/* (i=band; i< nbands; i++) */
  { /* covarm[k+clas+32] = cova[i]; */
    class[clas].covar[k] = cova[i];
    k++; }
}
}
eof = YES;
/* Name */
for (i = 0; i < MXXNDS; i++)
  nameline[i] = 0;
for (i = 0; i < 32; i++)
  class[clas].signame[i] = 0;
printf("name, CLASS \d+-", clas);
int zero, q1, q2, p1, p2, set;
float f0, f1, f2, f3, f4, f5;
int maxn(MXPAIR), minn(MXPAIR);
FILE *fpoint, *npoint;
char string[4];
char *tripep;
float covar[MXBNDS][MXBNDS], sigmap1, sigmap2, theta;
float diff1, diff2, diff3, diff4;
printf("\n Starting graph routine\n");

nbands = (*head).nbands;
classes = (*head).classes;
if (nbands <= 0 || nbands > MXBNDS || classes > MXCLAS)
  printf("Error working signatures\n");
  return FAILURE;
}
npairs = 0;
for (j = 1; j < nbands; j++) 
  npairs = npairs + j;

if ((classpt = (Points *) malloc(MXPAIR*sizeof(Points))) == NULL) 
  printf("\nNot enough memory to allocate points\n");
  return(NULL);

printf("\n # bands, # classes\n", nbands, classes);
printf("From class %d to class %d", *class, *class+1);
if (*mean == YES) printf("Plotting means only\n");

for (set = 0; set < npairs; set++) 
  { maxn[set] = 1; 
  minn[set] = 255; 
  }

for (class = *class; class < *class + classes; class++)
  { k = 0;
    /* Unfold covariance */
    for (band = 0; band < nbands; band++)
      { if (k > 32)
        
        printf("\nError in covar computation!\n");
        return FAILURE;
        }
        for (i = 0; i < nbands; i++) 
          
          covar[band][i] = class[i].covar[k];
          covar[i][band] = covar[band][i];
          k++;
        }
        set = 0;
        for (p1 = 0; p1 < nbands; p1++)
          
          sigmap = sqrt((double) covar[p1][p1]);
          for (p2 = p1+1; p2 < nbands; p2++)
            
            classpt.set.x0 = class[p1].meanvector[p1];
            classpt.set.y0 = class[p1].meanvector[p2];
            classpt.set.band = p1;
            classpt.set.band2 = p2;
            sigmap = sqrt((double) covar[p2][p2]);
            theta = atan((double) (sigmap2/sigmap1));
            diff1 = (*std)*sigmap*cos((double)theta);
            diff2 = (*std)*sigmap2*sin((double)theta);
if (max(set) > 255) max(set) = 255;
if (min(set) < 0) min(set) = 0;
min(set) = (min(set)/10)*10;
f0 = min(set);
fq1 = max(set);
fq2 = max(set);
fq3 = 50.;
fq4 = 10.;
q0 = classpt[set].band1 + 1;
sprintf(string,"%d",q1);
q1 = 1;
gaxis(f0, f0, 4f0, 4f0, 1, string, "%2.0f");
q1 = classpt[set].band2 + 1;
sprintf(string,"%d",q1);
q1 = 1;
gaxis(f0, f0, 4f0, 4f0, 1, string, "%4.0f");
for (class=1; class=nclass; class++)
{
  if (*mean != YES)
  {
    gplot (&classpt[set].x[1][clas],&classpt[set].y[1][clas],
        &classpt[set].x[2][clas],&classpt[set].y[2][clas],&qli, &class);
    gplot (&classpt[set].x[3][clas],&classpt[set].y[3][clas],
        &classpt[set].x[4][clas],&classpt[set].y[4][clas],&qli, &class);
    gplot (&classpt[set].x[5][clas],&classpt[set].y[5][clas],
        &classpt[set].x[6][clas],&classpt[set].y[6][clas],&qli, &class);
    gplot (&classpt[set].x[7][clas],&classpt[set].y[7][clas],
        &classpt[set].x[8][clas],&classpt[set].y[8][clas],&qli, &class);
  }
  else
  {
    gplot (&classpt[set].x[0][clas],&classpt[set].y[0][clas],
        &classpt[set].x[0][clas],&classpt[set].y[0][clas],&qli, &class);
  }
  gshow();
gclear();
set++;
}
quilt();
return SUCCESS;
SUBROUTINE MAIN

USCLAS -- SPECTRALLY CLASSIFY A MULTI-BAND IMAGE
--- UNSUPERVISED MULTISPECTRAL IMAGE CLASSIFICATION ---

USCLAS INAREA OUTAREA

PARAMETERS:
INAREA - AREA TO BE CLASSIFIED
OUTAREA - FINAL CLASSIFIED AREA WITH - M CLASSES.

KEYWORDS:
BANDS: NUMBER OF AREA BANDS TO CLASSIFY,
DEFAULT IS ALL BANDS IN AREA.
USER DEFINES THE BANDS TO USE;
BANDS= BAND1 BAND2 ... BANDN
CLASSES: INITIAL # OF CLASSES PRESENTED.
CLASSES = M DEFAULT IS 100
SKIP: SKIP FACTOR FOR ITERATIVE
CLASSIFICATION. NUMBER OF ROWS
AND COLUMNS SKIPPED. MUST BE < 8.
SKIP= N DEFAULT IS 4
ITER: MAXIMUM ALLOWED ITERATIONS
ITER= P DEFAULT IS 35
MERGE: CLUSTER Merging FACTOR, 0 to 2000,
IF A DIVERGENCE < MERGE THEN MERGE.
0 = NO MERGING
MERGE= Q DEFAULT IS 1400
SPLIT: SPLITTING FACTOR, 1 to 10,
IF BIG/SMALL DMS. > SPLIT = SPLITTING.
0 = NO SPLITTING
SPLIT= R DEFAULT IS 3.0
NULL: DATA DROP-OUT DECISION.
ANY: CLASSIFY AS ZERO IF ANY BAND HAS 0.
ALL: CLASSIFY AS ZERO IF ALL BANDS = 0.
DEFAULT IS ALL
MINPIX: SMALLEST PIXEL COUNT ALLOWABLE IN A CLASS
DEFAULT = 0.01% OF DATA
STRETCH: OUTPUT COLOR STRETCH TYPE.
LINE: LINEAR STRETCH FOR CLASS COLORS
HIST: HISTOGRAM EQUALIZATION STRETCH
DEFAULT = HIST

Written by Gerard Peltzer
UW SSEC 5/29/1992

MBXNDS sets max number of bands allowed in input image:
MAXRC sets maximum number of rows and cols in image:
MXCLAS sets maximum number of output data classes:
MKSKIP sets maximum skip factor:
MXLUMP sets maximum lumping (merging) factor:
MSXFLT sets maximum splitting factor:
MLUMP and MDSPLT set default lumping and splitting (none).
MDITER is default maximum number of iterations allowed.
ZMINP and ZMAXF are default and max % minimum pixel counts.
PARAMETER (MBXNDS=7)
PARAMETER (MAXRC=3584)
PARAMETER (MAXROW=1024)
PARAMETER (MAXCOL=3584)
PARAMETER (MXCLAS=300)
PARAMETER (MDXITER=100)
PARAMETER (MKSKIP=8)
PARAMETER (MDSPLT=4)

PARAMETER (EXMERO=2000.)
PARAMETER (EMERO=1400.)
PARAMETER (ZXSPLIT=10.)
PARAMETER (ZMSPRT=1.)
PARAMETER (ZDSPLIT=3.0)
PARAMETER (MDITER=55)
PARAMETER (MDITER=60)
PARAMETER (ZMINP=0.01)
PARAMETER (ZMAXF=10.0)

C

IMPLICIT CHARACTER*12 (C)
IMPLICIT REAL*8 (D)
CHARACTER*70 CMES, CBKLN
CHARACTER*8 LOGFIL
INTEGER*2 NAREA,NTAREA,TEST, MINPIX
INTEGER CLASSES, NBANDS, NDLOCLD(MBANDS)
REAL*8 LUMPF, SPLITF
INTEGER IDIR(64), ABANDS
INTEGER NROWS, NCOLS, SKIPF, ITERA, NULL, STRETCH
COMMON/TFILE/ LOGFIL
CALL SDEST('BEGIN', '0')
NAREA=1PP (1,0)
OTAREA=1PP (2,0)
IF (NAREA .LE. 0 OR, NAREA .GT. 50000) THEN
WRITE (CMES, '(1X,A15,1X,16)') 'INVALID AREA', ' INAREA
CALL SDEST(CMES,0)
RETURN
ENDIF
C Open output text file, use FDEST routine to print data to logfil:
CTEMP = CFRP(OTAREA)
LOGFIL(14) = CTEMP (9:12)
LOGFIL(5:8) = '.LOG'
CALL LWO (LOGFIL)
WRITE (CMES, '(70A1)') (' ', I = 1, 70)
CALL FDEST(LOGFIL,CMES)
CALL FDEST(LOGFIL,CMES)
WRITE (CMES, '(A14,1X,A12)') 'SUMMARY FILE', LOGFIL
CALL FDEST(LOGFIL,CMES)
CALL FDEST(CMES,0)
WRITE (CMES, '(1X,A34)') 'USCLAS UNSUPERVISED CLASSIFICATION'
CALL FDEST(CMES,0)
CALL FDEST(CMES,0)
WRITE (CMES, '(2X,A12,14)') 'INPUT AREA', ' INAREA
CALL SDEST(CMES,0)
CALL FDEST(CMES,0)
WRITE (CMES, '(1X,A19)') 'IMAGE CONTENTS'
CALL SDEST(CMES,0)
CALL FDEST(CMES,0)
C Open area to see how big, number of bands:
CALL ROWBIG(INAREA, NROWS, NCOLS)
WRITE (CMES, '(2X,A12,13)') ' ROWS:', ' NROWS
CALL SDEST(CMES,0)
CALL FDEST(CMES,0)
WRITE (CMES, '(2X,A12,13)') ' COLUMNS:', ' NCOLS
CALL SDEST(CMES,0)
CALL FDEST(CMES,0)
IF ((NROWS .GT. MAXROW) OR (NCOLS .GT. MAXCOL)) THEN

CALL SDEST('AREA TOO BIG TO PROCESS - EXITING!',0)
CALL FDEST(LOGFIL,CMES)
RETURN
ENDIF
CALL OPNA(INAREA)
CALL ARAOPT(INAREA,1,'SPAC',4)
CALL READD(INAREA,IDIR)
ABANDS=IDIR(14)
WRITE(CMES,'(I1,A19,15)') ' BANDS IN AREA: ', ABANDS
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
IF (ABANDS .LT. 1) THEN
CALL SDEST('NOT ENOUGH BANDS TO PROCESS - EXITING!',0)
CALL FDEST(LOGFIL,CMES)
RETURN
ENDIF
CALL CLSARA(INAREA)

Read in bands to use in classification:
NBANDS=KWP('BANDS')
IF (NBANDS .EQ. 0) THEN
NBANDS = ABANDS
DO 20 I=1,NBANDS
BNDLOC(I-1) = I
CONTINUE
ENDIF
DO 10 I = 1, NBANDS
BNDLOC(I-1) = KWP('BANDS',I,1)
CONTINUE
WRITE(CMES,'(I1,A20,15)') 'BANDS TO CLASSIFY: ', NBANDS
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
IF (NBANDS .GT. ABANDS) THEN
WRITE(CMES,'(I1,A40)')
C 'USER SPECIFIED MORE BANDS THAN EXIST IN AREA'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
RETURN
ENDIF

Read in classes:
CLASSES=KWP('CLASSES',1,MDCLAS)
Check number of classes less than maximum:
IF (CLASSES .GT. MDCLAS) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED TOO MANY CLASSES'
CALL SDEST(CMES,0)
WRITE(CMES,'(A25,15)') 'MAX CLASSES TO BE USED = ',CLASSES
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ENDIF

Read in skip factor:
SKIPP=KWP('SKIPP',1,MDSKIP)
Check skip less than maximum:
IF (SKIPP .GT. MDSKIP) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED TOO LARGE A SKIP'
CALL SDEST(CMES,0)
WRITE(CMES,'(A25,15)') 'MAX SKIP TO BE USED = ',SKIPP
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ENDIF
WRITE(CMES,'(A25,15)') 'MAX SKIP TO BE USED = ',SKIPP
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
C Read in merging (lump) factor:
DMERG = ZDMERG
LUMP+DWP('MERGE',1,DMERG)
C Check lump less than maximum:
IF (LUMP .GT. 2*DMERG) THEN
WRITE(CMES,'(A40)') 'USER GAVE TOO LARGE A MERGING FACTOR'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
LUMP = 2*LUMP
WRITE(CMES,'(A25,F9.1)') 'MAX T.DIV TO BE USED = ', LUMP
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ENDIF

C Read in split factor:
DSPLT = ZDSPLT
SPLIT+DWP('SPLIT',1,DSPLT)
C Check split less than maximum:
IF (SPLIT .GT. 2*DSPLT) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED TOO LARGE SPLIT FACTOR'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
SPLIT = 2*SPLIT
WRITE(CMES,'(A25,F6.3)') 'SPLIT TO BE USED = ', SPLIT
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ENDIF
IF (SPLIT .LT. 2*DSPLT .AND. SPLIT .LE. 0.0) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED BAD SPLIT FACTOR'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
SPLIT = DSPLT
WRITE(CMES,'(A25,F6.3)') 'SPLIT TO BE USED = ', SPLIT
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ENDIF

C Read in maximum number of iterations:
ITERA=KWP('ITER',1,MHITER)
C Check iter less than maximum:
IF (ITERA .GT. MHITER) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED TOO MANY ITERATIONS'
CALL SDEST(CMES,0)
ENDIF
C Read in maximum number of iterations:
ITERA=KWP('ITER',1,MHITER)
C Check iter less than maximum:
IF (ITERA .GT. MHITER) THEN
WRITE(CMES,'(A40)') 'USER SPECIFIED TOO MANY ITERATIONS'
CALL SDEST(CMES,0)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ITERA = MHITER
WRITE(CMES,'(A25,13)') 'MAX ITERATIONS TO BE USED = ',ITERA
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ENDIF
C Read in type of dropout handling desired in null:
CNUL=KWP('NULL',1,'ALL')
C Check NULL:
IF (CNUL, .NE. 'ALL') .AND. CNUL .NE. 'ANY') THEN
WRITE (CMES, ' (A00)') 'USER SPECIFIED INVALID NULL PARAMETER'
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)
CNUL = 'ALL'
ENDIF
IF (CNUL .EQ. 'ALL') NULL = 0
IF (CNUL .EQ. 'ANY') NULL = 1
WRITE (CMES, ' (A31, A12)') 'DATA DROP OUT HANDLING USED = ', CNUL
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)

Read in minimum size of clusters:
IPIX = 2MINF*NRows*Ncols/100
MINPIX = IPXF ('MINPIX', 1, IPIX)
Check iter less than maximum:
IF (MINPIX .GT. SMAXF*NRows*Ncols/100) THEN
WRITE (CMES, ' (A40)') 'MINIMUM PIXEL COUNT TOO LARGE A VALUE'
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)
MINPIX = SMAXF*NRows*Ncols/100
ENDIF
WRITE (CMES, ' (A32, A17)') 'MINIMUM PIXELS IN A CLUSTER = ', MINPIX
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)

Read in type of output stretch desired:
COMM = CMPS ('STRETCH', 1, 'HIST')
Check STRETCH:
IF (COMM .NE. 'LINE') .AND. COMM .NE. 'HIST') THEN
WRITE (CMES, ' (A41)') 'USER SPECIFIED INVALID STRETCH PARAMETER'
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)
COMM = 'HIST'
ENDIF
IF (COMM .EQ. 'HIST') STRET = 0
IF (COMM .EQ. 'LINE') STRET = 1
WRITE (CMES, ' (A24, A12)') 'OUTPUT STRETCH USED = ', COMM
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)

WRITE (CMES, ' (A1)')
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)
WRITE (CMES, ' (2X, A13, I4)') 'OUTPUT AREA = ', OAREA
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)
WRITE (CMES, ' (A1)')
CALL DSEST (CMES, 0)
CALL DFDST (LOGFIL, CMES)
WRITE (CMES, ' (A20)') 'ENTERING USCLAS....'
CALL DSEST (CMES, 0)

Classify the area:
CALL USCLAS (INAREA, OAREA, NBANDS, NRows, Ncols, BNDLOC, CLASSES, SKIPF,
LUMPY, SPLITF, ITERA, NULL, MINPIX, STRET)
CALL DSEST ('-----DONE-----', 0)
CALL DFDST (LOGFIL, '-----DONE-----')
END
SUBROUTINE USCLAS(INAREA, OTAREA, NBANDS, NROWS, NCOLS, BNDLOC, CLASSES, C SKIPP, LUMPF, SPLITF, MEXTER, NULL, MINPIN, STRET, TRESH)  

MAXBD determines the maximum number of bands allowed in the input image.  
MAXRC determines the maximum number of rows and cols in the image.  
MCOL determines the maximum number of output data classes.  
TRESH sets the default percent of pixels allowed.  
MEXTER sets the maximum number of allowed iterations.  
DCUTOFF is the percent of data for practical lower/upper bounds.  
IMPLICIT CHARACTER*12 (C)  
PARAMETER(MAXBD=7)  
PARAMETER(MAXRC=3584)  
PARAMETER(MAXROW=1024)  
PARAMETER(MAXCOL=3584)  
PARAMETER(MCLOL=300)  
PARAMETER(TRESH=4)  
PARAMETER(DCUTOFF=0.006)  

CHARACTER*70 CMES  
CHARACTER*8 TRDIL  
INTEGER*4 INAREA, OTAREA, MM, DATAM, NAV2, OUTDAT(0:MAXRC)  
INTEGER*2 BUF(0:MAXRC,0:MAXBDNS), NEWIN(0:MAXRC)  
INTEGER*2 ODLOUT(0:MAXCOL,0:MAXROW)  
INTEGER*4 I,J,K,L, IDIR(64), ROW, COL, BAND, CLASSES, DESCLAS, NROW, NROWS, NCOLS, BNDLOC(0:MAXBDNS), NULL, STRET, SNEAN(0:MAXBDNS,0:MCOL), SMIRN(0:MAXBDNS)  
INTEGER*2 MEXTER, SKIPF, CSKIP, CHECK, NRMI, NCMI, NRMI  
INTEGER*4 VAL, LEV, START, STARTO, I, J, ITERM, STOP, CLASSF, ITERM(0:MAXBDNS), MAX(0:MAXBDNS), SMAX(0:MAXBDNS), SUM(0:MAXBDNS), NWMEAN(0:MAXBDNS), NSMEAN(0:MAXBDNS,0:MCOL)  
REAL*8 HIST(0:MAXBDNS,0:255), LUMPF, SPLITF  
COMMON/TITLE, LOGFIL  

Lumping/splitting variables:  
INTG(0:MAXBDNS,0:MCOL), MINI(0:MAXBDNS,0:MCOL), MAXKMP  
REAL*8 STDJ(0:MAXBDNS,0:MCOL), STDMJ(0:MAXBDNS,0:MCOL), STDMX(0:MCOL)  
REAL*8 STDAG, STDIM, STDINJ(0:MCOL), STDAXJ(0:MCOL)  
REAL*8 DBRJ(0:MCOL), DNET(0:MAXBDNS,0:MCOL), DBAR  
REAL*8 DSAXJ(0:MCOL), XMIND, MAXSD  
REAL*8 COVAR(0:MAXBDNS,0:MCOL)  
REAL*8 COR(0:MCOL,0:MAXBDNS,0:MCOL)  
REAL*8 DIV(0:MCOL,0:MCOL)  
REAL*8 TIDV(0:MCOL,0:MCOL)  

--- Data File Headers ---  
Open McIDAS areas, set header values for the output file:  
CALL OMPARA(INAREA)  
CALL ARAQTP(INAREA,1,'SPAC',2)  
CALL READA(INAREA, IDIR)  
CALL MVCM('MIDCLASSIFICATION',IDIR(25))  
IDIR(11)=1  
IDIR(14)=1  
IDIR(19)=1  
IDIR(52)=LIT('VISIR')  
IDIR(53)=LIT('BTR')  
DATAM=IDIR(11)  
NAV2=IDIR(35)  
LEV=IDIR(51)  
IF (IDIR(16).NE.0) THEN  
VAL=IDIR(15)-IDIR(49)-IDIR(50)-IDIR(51)  
ELSE  
VAL=0  
ENDIF  
IDIR(36)=0  
C Define where the start of data is in input, in bytes:  
START = IDIR(15)  
C Do not copy DOC, CAL to output:  
STARTO = 0  
IDIR(50)=0  
IDIR(49)=0  
C Modify LEF. option to read just one band in new area:  
IDIR(51)=0  
IDIR(15)=STARTO  
IF (START_LT.0) STARTO=0  
C Start position in int^2 is half of bytes:  
START=START/2  
C Copy navigation from input area to the output area:  
CALL MAYARA(INAREA, IDIR)  
IF (NAV2_LT.0) THEN  
CALL ARAGET(INAREA, NAV2, 256, NAVARR)  
CALL ARAPUT(INAREA, NAV2, 256, NAVARR)  
ENDIF  
C CALL OMPARA(IN AREA)  

--- Initial Statistics Extraction ---  
C Find the mean and maximum values for each data band:  
C Find image histogram, get max, min stretching values (smln, xmax):  
DO 10 I=0, NBANDS, 1  
   MEAN(I)=0  
   MAX(I)=0  
   MIN(I)=0  
   SMLN(I)=0  
   NMMEAN(I)=0  
   DO 15 J=0, 255  
      HIST(I,J)=0  
   CONTINUE  
15  
10  
CONTINUE  
NCMI=NCOLS-1  
NRMI=NBNADS-1  
NRMI=NROWS-1  
DO 20 ROW=0, NROW, SKIPF  
DO 30 BAND=0, NBANDS-1, 1  
   RARAAR(INAREA, ROW,0, NCOLS, BNDLOC(BAND), NEWIN)  
   DO 35 COL=0, NROW, SKIPF  
      NWMEAN(BAND)=NWMEAN(BAND)+NEWIN(COL)  
      IF (MAX(BAND).LT. NEWIN(COL)) THEN  
         MAX(BAND)=NEWIN(COL)  
      ENDIF  
   HIST(BAND, NEWIN/COL) /(255** (DATAM-1))) =  
   HIST(BAND, NEWIN/COL)/(255** (DATAM-1)) + 1  
35  
30  
CONTINUE  
20  
CONTINUE  
51  
FORMAT(1X,A14,12,1X,A1,11,A4,2X,17)  
DO 50 BAND=0, NBANDS, 1  
   MEAN(BAND)=SKIPF**2*(NWMEAN(BAND)/(NCOLS*NROWS))  
   WRITE(C,MES,51) 'MEAN OF BAND', 'BAND'+1,'(',  
   C BNDLOC(BAND),'=', 'MEAN(BAND)'  
   CALL SDEST(CMES,0)  
   CALL FDEST(LOGFIL,CMES)  
   WRITE(C,MES,51) 'MAX OF BAND', 'BAND'+1,'(',  
   C BNDLOC(BAND),'=', 'MAX(BAND)'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

Use histogram to find a minimum for stretching:
J = 0
SUM = 0
CONTINUE
SUM = SUM + HIST(BAND,J)
IF (J .GE. 255) THEN
SMIN(BAND) = 0
GOTO 56
ENDIF

IF (SUM .LE. (NROWS*NCOLS)*DCUTOF/SKIP**2) THEN
J = J + 1
GOTO 55
ENDIF
SMIN(BAND) = J

Use histogram to find a maximum for stretching:
J = 255
SUM = 0
CONTINUE
SUM = SUM + HIST(BAND,J)
IF (J .LE. 0) THEN
SMAX(BAND) = 0
GOTO 57
ENDIF

IF (SUM .LE. (NROWS*NCOLS)*DCUTOF/SKIP**2) THEN
J = J - 1
GOTO 57
ENDIF
SMAX(BAND) = J
 WRITE(CMES,'(5X,F6.2,A21,1X,15)') DCUTOF*100,
' % data lower bound:',SMIN(BAND)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

WRITE(CMES,'(5X,F6.2,A21,1X,15)') (1-DCUTOF)*100,
' % data upper bound:',SMAX(BAND)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

CONTINUE

Call the initial clustering algorithm:
CALL ICLUST(NROWS,NCOLS,NBANDS,MAX,SMAX,MIN,NSMEAN,CLASSES)

Note the desired number of classes, set up for the loop:

DISCAS = CLASSES
STOP = 0
ITER = 1
SKIP = /
WRITE(CMES,'(A25,F6.3)') 
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A25,F7.1)') 'Merging factor used = ', LUMPF
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A25,F6.3)') 'Split factor used = ', SPLITF
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A25,F6.14)') 'Max Iterations allowed = ', MXITER
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

C Begin iterative classification loop:
99 CONTINUE

WRITE(CMES,'(A1)') 
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A30,1X,14)') 'PROCESSING ITERATION #',ITER
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(6X,13,1X,A12)') CLASSES, 'Classes used'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A31,1X,I3)') 'Row&column skip used = ', CSKIP
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

C Clear counts and means of classes:
DO 95 I=0, CLASSES, 1
SCOUNT(I)=0
DO 105 BAND=0, NBM1, 1
NSMEAN(BAND, I) = 0
105 CONTINUE
95 CONTINUE
PERCH = 0.0

C Start loop over rows and columns, convert input to output:
DO 60 ROW=0, NBM1, CSKIP
DO 70 BAND=0, NBM1, CSKIP
C function readar(AREA,LIN,STARTELEM,ELEMENTS,BAND,ARRAY)
CALL REDAR(INAREA,ROW,0,NCOLS,ENDBLOC,BAND,FINM1)
DO 75 COL=0, NCM1, CSKIP
BUF(COL,BAND) = NEWIN(COL)
75 CONTINUE
70 CONTINUE

C Classify:
DO 80 COL=0, NCM1, CSKIP
OUTDAT(COL) = MDM(CLASSES,ROW,COL,NBANDS,BUF,NSMEAN,NULL)
80 CONTINUE

C Develop new means and point count for data:
DO 110 COL=0, NCM1, CSKIP
DO 120 BAND=0, NBM1, 1
NSMEAN(BAND,OUTDAT(COL)) = NSMEAN(BAND,OUTDAT(COL)) +
BUF(COL,BAND)
120 CONTINUE
SCOUNT(OUTDAT(COL)) = SCOUNT(OUTDAT(COL)) +1
110 CONTINUE

C Get in previous classification, save to array:
IF (STOP .EQ. 0) THEN
DO 130 COL=0, NCM1, CSKIP
IF (ITER .GT. 1) THEN
IF (OLDOUT(COL,ROW) .NE. OUTDAT(COL)) THEN
PERCH=PERCH+1
ENDIF
ENDIF
.OLDOUT(COL,ROW) = OUTDAT(COL)
130 CONTINUE
135 CONTINUE
ENDIF
ENDIF
IF (STOP .EQ. 1 .AND. CSKIP .EQ. 1) THEN
DO 135 COL=0, NCM1, 1
.OLDOUT(COL,ROW) = OUTDAT(COL)
135 CONTINUE
ENDIF

CONTINUE

IF (STOP .EQ. 1) THEN
  WRITE(CMES,'(A33)') ' -- Processing final iteration. --'
  CALL SDST(CMES,0)
  CALL FDST(LOGFIL,CMES)
ENDIF

Write out means used in classification to screen:
IF (STOP .EQ. 0) THEN
  WRITE(CMES,'(A35)') 'Cluster means used this iteration: '
  CALL FDST(LOGFIL,CMES)
  WRITE(CMES,'(A36)') '----Band Means-----
  CALL FDST(LOGFIL,CMES)
  WRITE(CMES,'(A13,5X,7(A1,1,1,1,1,3X))') 'Class Counts',
  '('(','BNACL,(I,1)'),',I=0,NMM1)
  CALL FDST(LOGFIL,CMES)
  DO 137 I=0, NCLASSES, 1
    WRITE(CMES,'(13,2X,F8.1,1X,7(I1,15,1))') I,SCOUNT(I)*(CSKIP**2),
    (SMEAN(BAND,1),BAND=0,NMM)
  CALL FDST(LOGFIL,CMES)
  CONTINUE
ENDIF

Calculate new means:
  DO 160 I=1, NCLASSES, 1
    DO 150 BAND=0, NMM, 1
        IF (SCOUNT(I) .GE. 1) THEN
            SMEAN(BAND,1) = NSMEAN(BAND,1)/SCOUNT(I)
        ELSE
            SMEAN(BAND,1) = 0
        ENDIF
      137 CONTINUE
    150 CONTINUE
  160 CONTINUE

Find percent of pixels that changed classes:
   IF (ITER .GT. 1 .AND. STOP .EQ. 0) THEN
      PERCH = PERCH*(CSKIP**2)/(NCLASSES*NMM)**2.0
      WRITE(CMES,'(8X,A10,13,A16,1X,F8.4)') 'Iteration ','ITER,
      PERCENT CHANGE: ','PERCH
      CALL SDST(CMES,0)
      CALL FDST(LOGFIL,CMES)
   IF (PERCH .LT. THRESH) STOP = STOP + 1
   ENDIF

Apply lumping (merging) and splitting of classes:
Variable names after Tou & Gonzalez, "Pattern Recognition Principles"
Some definitions:
STDIJ (BAND,CLASS) Std deviation of each class, for each band.
STMAXJ(CLASS) The maximum Std deviation for each class.
STMINJ(CLASS) The minimum Std deviation for each class.
STMAX The minimum Std deviation for all classes.
STMIN The minimum Std deviation for all classes.
STAVG The average Std deviation for all classes.
DNMTJ (BAND,CLASS) The average distance of samples in each class from
                 the class mean, for each band.
DBARJ(CLASS) The average distance of samples from their means.
DBAR Overall average distance of samples from their means.
SDIST(1,2) Spectral distance between means of classes 1 and 2.
MNSD, MAXSD The smallest and largest SDISTS.

----------------------------------------------
C MAXLMP The maximum number of lumpings allowed (classes/7).
C COVAR(CLASS,1,2) Set of CLASS covariance matrices.
C COVRE(CLASS,1,2) Set of CLASS correlation matrices.
C TDIV(CLASS,1,2) Transformed Divergence between classes 1 and 2.

C----------------------------------------------
C Extract the class statistics:
C Initialize variables:
OLDLMP - CLASSES
DBAR - 0
STAVG - 0
STMAX - 0
STMIN - 0
DO 225 I = 0, NCLASS, 1
  DBARJ(I) = 0
  STMAXJ(I) = 0
  STMINJ(I) = 0
  DO 220 J = 0, NCLASS, 1
    SDIST(I,J) = 0
  CONTINUE
  DO 225 J = 0, NCLASS, 1
    SDIST(J,I) = SDIST(I,J)
  CONTINUE
225 CONTINUE

C Find pairwise cluster distances:
DO 320 I=0, NCLASSES-1, 1
  MIND(I) = 0
  MIND(J) = 0
  DO 330 J=I+1, NCLASSES, 1
    SDIST(I,J) = SDIST(I,J) + (SMEAN(BAND,1) - SMEAN(BAND,1))**2
  CONTINUE
330 CONTINUE
320 CONTINUE

C Loop over rows and columns, compute dbar, stds;
C Develop average distance of samples from cluster centers:
DO 210 BAND=0, NMM, 1
  CALL REDATA(INAREA,ROW,0,NCOLS,BANDLOC(BAND),NEWIN)
  DO 230 COL=0, NMM, 1
    DBARJ(BAND,CLASS) = DBARJ(BAND,CLASS) + DBARJ(BAND,CLASS)
  CONTINUE
230 CONTINUE
210 CONTINUE

C Covariance calculations:
DO 350 COL=0, NMM, 1
  DO 360 ROW=0, NMM, 1
    IF (I .NE. J) THEN
      COVAR(OLDFUL,OLDFUL) = COVAR(OLDFUL,OLDFUL) + DBARJ(BAND,CLASS) +
      (DBARJ(BAND,CLASS)**2 - SMEAN(BAND,CLASS)**2)
    ELSE
      COVAR(OLDFUL,OLDFUL) = COVAR(OLDFUL,OLDFUL) + DBARJ(BAND,CLASS)
    ENDIF
  CONTINUE
360 CONTINUE
350 CONTINUE

570 CONTINUE
**usclass.pgm**

**C** Convert covariance to correlation:

```plaintext
DO 591 J = 1, CLASSES
DO 601 BAND = 0, NBM1
   IF (COVAR(I, BAND, J) .NE. 0, AND.
      C STDIJ(BAND, I) .NE. 0, AND.
      C COVAR(I, BAND, J)) THEN
      C STDIJ(BAND, I) * COVAR(I, BAND, J) / (STDIJ(BAND, I) * STDIJ(BAND, I))
      ELSE
      C COVAR(I, BAND, J) = 0
   ENDIF
   C Filler other half of symmetric correlation matrix:
   IF (J .NE. BAND) CORR(I, J, BAND) = CORR(I, BAND, J)
   C END the lumping/splitting of classes.
   C Write out mean array to screen:
   IF (STOP .EQ. 1) THEN
      WRITE(CMES,'(A21)') 'Final cluster means:
      CALL FDEST(LOGFIL, CMES)
      ELSE
      WRITE(CMES,'(4X,A23)') 'Adjusted cluster means:'
      CALL FDEST(LOGFIL, CMES)
   ENDIF
   C Convert correlation matrix, pairwise distances matrix:
   C Write out mean array to screen:
   IF (STOP .EQ. 1) THEN
      WRITE(CMES,'(A21)') 'Final cluster means:
      CALL FDEST(LOGFIL, CMES)
      ELSE
      WRITE(CMES,'(4X,A23)') 'Adjusted cluster means:'
      CALL FDEST(LOGFIL, CMES)
   ENDIF
   C Convert covariance to correlation:
   DO 591 J = 1, CLASSES
   DO 601 BAND = 0, NBM1
      IF (COVAR(I, BAND, J) .NE. 0, AND.
         C STDIJ(BAND, I) .NE. 0, AND.
         C COVAR(I, BAND, J)) THEN
         C STDIJ(BAND, I) * COVAR(I, BAND, J) / (STDIJ(BAND, I) * STDIJ(BAND, I))
         ELSE
         C COVAR(I, BAND, J) = 0
      ENDIF
      C Filler other half of symmetric correlation matrix:
      IF (J .NE. BAND) CORR(I, J, BAND) = CORR(I, BAND, J)
      C END the lumping/splitting of classes.
      C Write out mean array to screen:
      IF (STOP .EQ. 1) THEN
         WRITE(CMES,'(A21)') 'Final cluster means:
         CALL FDEST(LOGFIL, CMES)
         ELSE
         WRITE(CMES,'(4X,A23)') 'Adjusted cluster means:'
         CALL FDEST(LOGFIL, CMES)
      ENDIF
      C Convert correlation matrix, pairwise distances matrix:
      C Write out mean array to screen:
      IF (STOP .EQ. 1) THEN
         WRITE(CMES,'(A21)') 'Final cluster means:
         CALL FDEST(LOGFIL, CMES)
         ELSE
         WRITE(CMES,'(4X,A23)') 'Adjusted cluster means:'
         CALL FDEST(LOGFIL, CMES)
      ENDIF
      C Convert correlation matrix, pairwise distances matrix:
```
C Save an enhancement table along with the output image:

WRITE(CMES,'(A36)') 'OUTPUT COLOR ASSIGNMENTS:'
CALL SDSE(X(CMES,0))
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A24,2X,A1,1X,i1,1X,A1)') 'RED FROM BAND 1,'
C
(C', 'BDLOC(I,1)')
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A24,2X,A1,1X,i1,1X,A1)') 'GREEN FROM BAND 2,'
C
(C', 'BDLOC(I,1)')
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A24,2X,A1,1X,i1,1X,A1)') 'BLUE FROM BAND 3,'
C
(C', 'BDLOC(I,1)')
CALL FDSTEST(FOCOLOR,CMES)
CALL FDSTEST(FOCOLOR,CMES)
C Write McIDAS .ET color look-up table file:
CALL ENUNCE (ITAREA, NBANDS, MAX, SMAK, SMIN, SMAX, SMOAN, CLASSES, HIST, STRET)
WRITE(CMES,'(A1)') '
CALL SDSE(CMES,0)
CALL FDSTEST(FOCOLOR,CMES)
C Write class signatures data file:
CALL SIGOUT (ITAREA, NBANDS, MAX, SMAX, SMOAN, SCOUNT, DBARJ, STDI1J,
C
SMARK, CLASSES, BDLOC, COVAR)
WRITE(CMES,'(A1)') '
CALL SDSE(CMES,0)
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A19,1X,A7)') 'Log file: ', LOGFOL
CALL SDSE(CMES,0)
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A60)')
C 'Contains a detailed log of the procedure, including final
CALL SDSE(CMES,0)
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A60)')
C 'class statistics, mean distances, and Transformed divergence'
CALL SDSE(CMES,0)
CALL FDSTEST(FOCOLOR,CMES)
WRITE(CMES,'(A60)')
C 'matrices.'
CALL SDSE(CMES,0)
CALL FDSTEST(FOCOLOR,CMES)
RETURN
END

C-------------------------------------------------------------
C ICULST (INITIAL CLUSTER)
C Function to find initial cluster means.
C-------------------------------------------------------------
C SUBROUTINE ICULST (NRWNS, NCOLS, NBANDS, MAX, SMAX, SMIN, SMOAN, CLASSES)
PARAMETER (MXCLAS=300)
PARAMETER (MBNANDS=7)
CHARACTER*70 CMES
INTEGER NRWNS, NCOLS, NBANDS, CLASSES
INTEGER MAX(0:MBNANDS), SMAX(0:MBNANDS), SMIN(0:MBNANDS)
INTEGER SMOAN(0:MBNANDS, 0:MXCLAS)
INTEGER I, ROW, COL, BAND, SET
REAL*8 DELTA

Note: iterate set from 1 to nsets, save set=0 for unclassified. */
DO 10 BAND=0, NBANDS-1, 1
DELTA = MAX (BAND)/FLOAT (CLASSES)
IF (DELTA .LT. 1) DELTA = 1
DO 20 SET=0, CLASSES, 1
SMEAN (BAND, SET) = SET * DELTA
CONTINUE
RETURN
END

FUNCTION write out a triangular CLASSES by CLASSES matrix
Matrix elements divided by SIZE in output.
SCREEN toggle writes out to screen if a 1 is input.

SUBROUTINE CXXOUT (RMat, CLASSES, MXCLAS, SCREEN)
REAL*8 RMat (0:MXCLAS, 0:MXCLAS)
INTEGER CLASSES, J, I, II, JJ, CHECK, SIZE, SCREEN
INTEGER DMAT (1:20)
COMMON /TFILE/ LOGFIL
CHARACTER*70 CMES
C Check the maximum digit size of the matrix:
DO 468 II = 0, CLASSES -1
DO 469 JJ = 0, II
IF (RMat (II, JJ) .GT. MAX) MAX = RMat (II, JJ)
CONTINUE
469 C
CONTINUE
468 IF (MAX .GT. 9999) THEN
SIZE = 10
WRITE (CMES, ' (5X, A24, I3, Al) ') ' (Value shown - distance',
SIZE, ')
ELSE
IF (SCREEN .EQ. 1) CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
ELSE
SIZE = 1
ENDIF
C Printout:
DO 470 II = 0, CLASSES/13
WRITE (CMES, ' (AI) ') ' ',
IF (SCREEN .EQ. 1) CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
IF (I+I3+11 .LT. CLASSES) THEN
JJ = 12+13*II
ELSE
JJ = CLASSES
ENDIF
WRITE (CMES, ' (5X, A5, I3, 2X) ') ' Clusters', (I, I=I3+11, JJ)
IF (SCREEN .EQ. 1) CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
DO 460 II=I+II, CLASSES, 1
IF (I+II+13 .GT. 12) THEN
CHECK = I+II+12
ELSE
CHECK = I
ENDIF
DO 11 J = II+1, CHECK
DMAT (J) = RMat (J, J)/SIZE
CONTINUE
WRITE (CMES, ' (I3, 1X, I3, 14, IX) ') I, C
(C DMAT (J), J=II+1, CHECK)
IF (SCREEN .EQ. 1) CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
CONTINUE
460 C
CONTINUE
470 RETURN
END
C
SPLIT

Function to split apart clusters using Principal Axes of Inertia.

Calls EIGENS, a routine for finding the Eigenvalues and Eigenvectors for each class.

SUBROUTINE SPLITI(CLASSES, NBRANDS, DSCALAS, ITER, SPLITF, SKIPP,
                   COVAR, SMAX, SMAXI)

PARAMETER (MNBANDS=7)
PARAMETER (MCLASS=300)
PARAMETER (NEG1 = -1)
PARAMETER (YES = 1)
PARAMETER (NO = 0)

INTEGER SMAXI(0:MCLASS,SKIPP,0), SMAXI(0:MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

REAL*8 SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

REAL*8 SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

REAL*8 SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

INTEGER SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

INTEGER SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

INTEGER SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

INTEGER SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

INTEGER SMAXI(0:MCLASS,0,MCLASS, STDMIN, STDAGV, DBAR, SPLIT, SMAX, SMAXI)

COMMON TFIL, LOGFIL

CHARACTER*70 CMES

Exit if splitting operation is not enabled, or if odd iteration:
IF (SPLITF .LT. 0.001 .OR. MOD(ITER,2) .EQ. 0) RETURN

Tell log file splitting is being performed:
WRITE(CMES,'(2X,A34)') 'Running Cluster splitting routine.'
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

MAXSPL is maximum number of cluster pairs which can be split.
MAXSPL = CLASSES/3

OLDCLS = CLASSES

NBRM = NBRANDS - 1

1 = 1

CONTINUE

SPLIT = NO

ENDIF

Quit if we already have too many classes:
IF (CLASSES-OLDCLS .GT. MAXSPL) RETURN

Split clusters no matter what if there are fewer classes:
Find the class with the largest STD and split it:
IF ( (CLASSES .LE. DSCALAS/20) ) THEN
  MAX = 0
  DO 10 J = 1, CLASSES
    IF (STMAX(J) .GE. MAX) THEN
      MAX = STMAX(J)
      J = J
  ENDIF
  CONTINUE
  SMALL = YES
  SPLIT = YES
ELSE
  Move to next class if not enough pixels:
2   IF (SCOUNT(I)*SKIPP**2 .LT. 10*MINPIX) THEN
    I = I + 1
    GOTO 2
  ENDIF
  IF (I*GT. CLASSES) RETURN

  Run check to see if one or more of I's bands have mean 0:
  IF ((SMEAN(J),J) .EQ. 0) THEN
    IBNDS = NBRANDS
    DO 210 J = 0, NBRM
      IF (SMEAN(J,J) .EQ. 0) THEN
        IBNDS = IBNDS - 1
        BADD(J) = YES
      ELSE
        BADD(J) = NO
      ENDIF
    CONTINUE

Implement Inertial Matrix for multiple bands:

Build inertial matrix out of Covariance matrix:

NBRM = IBNDS - 1

DO 43 J = 0, NBRANDS - 1
  EVALU(J) = 0
  DO 43 K = 0, NBRANDS - 1
    IMAT(J,K) = 0
    EVEC(J,K) = 0
  CONTINUE

CONTINUE

K = 0

DO 46 J = 0, NBRANDS - 1
  IF (BADD(J,J) .EQ. NO) THEN
    IF (J-BADD(J,J) .EQ. 0) THEN
      IMAT(J,K) = NBRM*COVAR(I,J,J)
      H = H + 1
    IF (J-BADD(J,J) .EQ. NO) THEN
      IMAT(J,K) = IMAT(J,I) + COVAR(I,J,J)
    ENDIF
    CONTINUE

K = K + 1

CONTINUE

Find Eigenvalues and Eigenvectors for each IMAT:

If only one band, use ratio approach:

IF (IBNDS .EQ. 1) THEN
  WRITE(CMES,'(A20,1X,I3,1X,F9.3,F9.2)') 'Cluster: ',I,
  SMAX(A,I) STDAGV
  CALL SDEST(CMES,0)
  CALL FDEST(FDEST,CMES)
  IF (DBAR(J) .GT. DBAR .AND. (STMAX(J)/STDAGV .GT. SPLIT) .AND. (SCOUNT(I)*SKIPP**2 .GT. 10*MINPIX)
    THEN
      WRITE(CMES,'(A20,1X,I3,1X,F9.3,F9.2)') 'Split: ',I,
      SCOUNT(I)*SKIPP**2,10*MINPIX
      CALL SDEST(CMES,0)
      CALL FDEST(FDEST,CMES)
    SPLIT = YES
  ELSE
    SPLIT = YES
  ENDIF
ENDIF
ELSE
   I = I + 1
ENDIF
ELSE

------------

Information printouts:

WRITE (CMES, '(A15,1X,15)') 'COVAR matrix: ', I
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
DO 603 K = 0, NBANDS-1
   WRITE (CMES, '(10(F8.3,1X))') (COVAR(I,K), J=0,NBANDS-1)
   CALL SDEST (CMES, 0)
   CALL FDEST (LOGFIL, CMES)
CONTINUE
WRITE (CMES, '(A15,1X,15)') 'I matrix: ', I
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
DO 606 K = 0, NBANDM1
   WRITE (CMES, '(10(F8.3,1X))') (IMAT(K,J), J=0,NBANDM1)
   CALL SDEST (CMES, 0)
   CALL FDEST (LOGFIL, CMES)
CONTINUE

CALL EIGENS (IMAT, IBANDS, MXBANDS, EVECTR, EVALUE)

------------

Information printouts:

WRITE (CMES, '(A15,1X,15)') 'EVALUATE: ', I
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
WRITE (CMES, '(10(F8.3,1X))') (EVALUE(J), J=0,NBANDM1)
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
WRITE (CMES, '(A15,1X,15)') 'EVECTR: ', I
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
DO 609 K = 0, NBANDM1
   WRITE (CMES, '(10(F8.3,1X))') (EVECTR(K,J), J=0,NBANDM1)
   CALL SDEST (CMES, 0)
   CALL FDEST (LOGFIL, CMES)
CONTINUE

IF (EVALUE(NBANDM1) .NE. 0) THEN
   WRITE (CMES, '(A20,1X,13,1X,F9.3,F9.2,F9.3)') 'Cluster: ', I,
   EVALUE(0), EVALUE(NBANDM1), EVALUE(0)/EVALUE(NBANDM1)
   CALL SDEST (CMES, 0)
   CALL FDEST (LOGFIL, CMES)
ENDIF

IF (EVALUE(NBANDM1) .NE. 0) THEN
   IF ( (EVALUE(0)/EVALUE(NBANDM1) .GT. SPLIT**2) .AND.
       (SCOUNT(1)*SKIFF**2 .GT. 10.0*MINFIX) ) THEN
      SPLIT = YES
   ENDIF
ENDIF
IF (SPLIT .EQ. NO) I = I + 1
ENDIF

IF (SPLIT .EQ. YES) THEN
   CLASSES = CLASSES + 1
   WRITE (CMES, '(9X,A20,1X,13)') 'Splitting cluster: ', I
   CALL SDEST (CMES, 0)
ENDIF

CALL FDEST (LOGFIL, CMES)
STMINJ(K) = 0
STMAXJ(K) = 0
DO 290 K = CLASSES, I+1, -1
   SCOUNT(K) = SCOUNT(K-1)
   STMAXJ(K) = STMAXJ(K-1)
   STMINJ(K) = STMINJ(K-1)
   DBARJ(K) = DBARJ(K-1)
   DO 300 BAND = 0, NBANDM1, 1
      STDIJ(BAND,K) = STDIJ(BAND,K-1)
      SMAX(BAND,K) = SMAX(BAND,K-1)
      DO 300 J = 0, NBANDM1, 1
         COVAR(K,BAND,J) = COVAR(K-1,BAND,J)
      CONTINUE
   CONTINUE
   IF (NBANDS .EQ. 1) THEN
      SMAX(0,1) = SMAX(0,1) + STDIJ(0,1)/1.5
      SMAX(0,1) = SMAX(0,1) + STDIJ(0,1)/1.5
      IF (SMAX(0,1) .LT. 0) SMAX(0,1) = 0
   ELSE
      Split along principal axis:
      DO 310 BAND = 0, NBANDS-1, 1
      Expand EVECTR back into real space:
      DIFF = 0.
      IF (BAND .EQ. NO) THEN
         DIFF = - EVECTR(BAND,NBANDM1)*SORT(EVALUE(0))/2.0
         IF (DIFF .GT. 10.0 .AND. SMAX(BAND) .LT. 255.0) DIFF = 10
         IF (DIFF .GT. 2550.0 .AND. SMAX(BAND) .GT. 255.0) DIFF = 2550
      ELSE
         SMAX(BAND,1) = SMAX(BAND,1) + DIFF
         SMAX(BAND,1) = SMAX(BAND,1) - DIFF
         IF (SMAX(BAND,1) .LT. 0) SMAX(BAND,1) = 0
         IF (SMAX(BAND,1) .LT. 0) SMAX(BAND,1) = 0
      ELSE
      ENDIF
   ELSE
   ENDIF
CONTINUE
I = I + 2
ENDIF

888 CONTINUE
IF (FDEST .EQ. YES) I = 1
IF (I .LE. CLASSES) GOTO 1

999 CONTINUE
RETURN
END

C*****************************************************************************
C EIGENS
C*****************************************************************************

C Computes all eigenvectors and eigenvalues of a real symmetric
C matrix A(i,N,1:N). Non diagonal elements of A are destroyed.
C EVALUE(i,N) returns eigenvalues, EVECTR(i,N,1:N) is a matrix whose
C columns contain, on output, the normalized eigenvectors of A.
C Output is ordered in descending order.
C This routine is based on "Numerical Recipes" routines JACOBI
C and EIGEN.
C*****************************************************************************

SUBROUTINE EIGENS(IMAT, NBANDS, MXBANDS, EVECTR, EVALUE)
REAL*8 IMAT (1:MXBANDS, 1:NBANDS)

C*****************************************************************************
C*****************************************************************************

REAL*8 EVECTR(1:MBNDS, 1:MBNDS), EVALUE(1:MBNDS)
INTEGER CLASS, I, J, NROT

CALL JACOB J(IA, NBANDS, MBNDS, EVALUE, EVECTR, NROT)
CALL EIGSR (EVALUE, EVECTR, NBANDS, MBNDS)
RETURN
END

---------------------------------------------------------------------

SUBROUTINE JACOB J(A, N, NF, D, V, NROT)
PARAMETER (NMAX=100)
REAL*8 A(NF, N), D(N), V(NF, N), B(NMAX), Z(NMAX)
DO 12 IP=1, N
    DO 11 IQ=1, N
        V(IP, IQ) = 0.
        CONTINUE
    V(IP, IP) = 1.
    CONTINUE
DO 13 IP=1, N
    B(IP)= A(IP, IP)
    D(IP)= B(IP)
    Z(IP)= 0.
    CONTINUE
NROT=0
DO 24 I=1, 150
    SM=0.
    DO 15 IP=1, N-1
        SM=SM+ABS(A(IP, IQ))
    CONTINUE
    CONTINUE
    IF (SM.EQ.0.) RETURN
    IF (I.LT.4) THEN
        TRESH=0.2*SM/N**2
    ELSE
        TRESH=0.
    ENDIF
ENDIF
DO 22 IQ=1, N-1
    DO 21 IP=1, N-1
        G=100.*ABS(A(IP, IQ))
        IF (I.EQ.1) THEN
            A(IP, IP) = 1.
        ELSE IF (A(IP, IQ).LT.0.05) THEN
            A(IP, IQ) = 0.
        ENDIF
        D(IP)= D(IP)
        IF (ABS(A(IP, IQ)) .GT. TRESH) THEN
            D(IP)= D(IP)
        ELSE
            T=ABS(A(IP, IQ))
            IF (T.EQ.0.) THEN
                K=1
                RETURN
            ELSE
                T=ABS(A(IP, IQ))**2
                IF (T.TE 0.0) THEN
                    CONTINUE
                ELSE
                    CONTINUE
                    RETURN
                ENDIF
            ENDIF
        ENDIF
        CONTINUE
    ENDIF
    ENDIF
    CONTINUE
END

---------------------------------------------------------------------

SUBROUTINE EIGSR (D, V, N, NF)
REAL*8 D(NF), V(NF, N)
DO 13 I=1, N-1
    K=I
    P=D(I)
    DO 11 J=I+1, N
        IF (D(J).NE.P) THEN
            K=J
            P=D(J)
        ENDIF
    ENDIF
    CONTINUE
    IF (K.NE.1) THEN
        D(K)=D(I)
        D(I)=P
        DO 12 J=1, N
            V(J, I)=V(J, K)
            V(J, K)=P
        ENDIF
    ENDIF
    CONTINUE
RETURN
END

---------------------------------------------------------------------

C Function to merge together clusters by analysis of Transformed

C LUMP TD

C
Divergence.
LUMPF is the threshold transformed divergence between classes
to perform lumping. Pairs with a t. divergence less than the
merging factor will be merged. 1/3 of the cluster pairs is
the maximum number of mergings allowed.

SUBROUTINE LUMPF(CLASSES,NBANDS,OLDCLS,ITER,LUMPF,SMEA,N,SCOUNT,
C
SDIST,SDT,J,STMAXJ,STMAJD,STMAKB,STMAJ,STDIV,DIV,DSCLAS
C
C
PARAMETER (MXBND=7)
PARAMETER (MXCLS=400)
REAL*4 SDTJ0(0:MXCLSJ,0:MXCLS), SDIST(0:MXCLS,0:MXCLS)
REAL*4 DBRJ0(0:MXCLS), SCOUNT(0:MXCLS), STMAJ0(0:MXCLS)
REAL*4 STMAJ, DBJ0, MNCTD, MAXTD, LUMPF
REAL*4 COVAR(0:MXCLS,0:MXBND), SDIST(0:MXBND)
REAL*4 DIV(0:MXCLS,0:MXCLS)
REAL*4 DIV(0:MXCLS,0:MXCLS)
INTEGER SMAX0(0:MXBND0,0:MXCLS), ROW, PREI, PREJ, I, JJ
INTEGER NBANDS,CLASSES, OLDCLS, ITER, K, BAND, DSCLAS
INTEGER MINJ0(0:MXCLS), MINJ(0:MXCLS), MAXLMP
COMMON/ITLE,LOGFIL/
CHARACTER*70 CMES

Exit if lumping operation is not enabled:
IF (LUMPF .LT. 0.01) RETURN

Sort, find lowest divergences for lumping:
ROW = 0
MAXTD = 0
MAXLMP is maximum number of cluster pairs which can be lumped.
MAXLMP = CLASSES/3

CONTINUE
MINCTD = 1000000
DO 360 I=1, CLASSES, 1
DO 370 J=1, I-1, 1
IF (CLASSES .LE. DSCLAS/20) THEN
WRITE(CMES,'(A50)') 'Too few classes for lumping'
CALL SDESF(CMES,0)
CALL DEST(LOGFIL,CMES)
RETURN
ENDIF
IF (ROW .EQ. 0) THEN
MAXTD = TDIV(I,J)
360 CONTINUE
C
IF (TDIV(J,J) .LT. 0.01) THEN
J = 0
DO 400 II=0, ROW
IF (J .EQ. MINII) THEN
We have already found this one, or one of these
classes to be lumped somewhere else.
J = 1
ENDIF
CONTINUE

CONTINUE
IF (JJ .EQ. 0) THEN
PREI = I
PREJ = J
MINCTD = TDIV(I,J)
ENDIF
CONTINUE

CONTINUE
If no distances are less than the minimum ,exit the loop!
IF (ROW .EQ. 0) THEN
WRITE(CMES,'(2X,A32)')
C 'Running Cluster merging routine.'
C (Histeq) Find total number of pixels:
IF (STRET .EQ. 0) THEN
  DO 9 I = 0, NBANDS-1
  HCOUNT (I) = 0
  DO 8 J = 1, 255
    HCOUNT (I) = HCOUNT (I) + HIST (I, J)
  8 CONTINUE
  9 CONTINUE
ENDIF
C-------------------
SUBROUTINE ENHANCE (IAA, NBANDS, MAX, SMAX, SMIN, SMEAN, CLASSES, HIST, C, STRET)
IMPLICIT CHARACTER(*) (C)
PARAMETER (MXBANDS=7)
PARAMETER (MXXCLAS=300)
PARAMETER (STRETCH=2)
CHARACTER*70 CMES
INTEGER NROWS, NCOLS, NBANDS, CLASSES, STRET
INTEGER MAX (0:MXBANDS), SMAX (0:MXBANDS), SMIN (0:MXBANDS)
INTEGER SMEAN (0:MXBANDS,0:MXXCLAS)
INTEGER MNRED (0:MXXCLAS), MNGRE (0:MXXCLAS), MNBLU (0:MXXCLAS)
INTEGER MXRED, MXGRE, MXBLU
INTEGER MNRED, MNGRE, MNBLU
INTEGER MNRED, MNGRE, MNBLU
INTEGER I, BAND, SET, TAB8 (816), SIZE
INTEGER*4 IAR
REAL*8 HIST (0:MXBANDS,0:255), HCOUNT (0:2), HISRED, HISGRE, HISBLU
REAL*4 REDST, GREENST, BLUEST
CHARACTER*7 TABFIL
COMMON/TFILE/ LOGFIL
Parameters for enhancement tables:
DIMENSION IRED (15), IGREEN (15), IBLUE (15)
DATA FULL / 816, GRE/256, BLU/512, GRE/768, GBLU/801/
DATA IRED / 255, 0, 255, 0, 255, 0, 255, 255, 255, 255, 227, 227, 227,
DATA IBLUE / 255, 0, 255, 0, 255, 255, 255, 255, 227, 227, 227, 227,
DATA IGREEN / 255, 255, 255, 227, 227, 227, 227, 227, 227, 227, 227,
DATA CNWOUT = CFL (IAR)
TABFIL (114) = CNWOUT (9:12)
TABFIL (517) = CTABFIL
CALL LWD (TABFIL)
WRITE (CMES, ' (A19,1X,A7)' ) 'Enhancement file: ', TABFIL
CALL SDEST (CMES, 0)
ELSE IF (NBANDS .EQ. 2) THEN
  IF (STRET .EQ. 1) THEN
    MNRED(I) = SMAX(I, J)
    MNGRE(I) = SMAX(J, 0)
    MNBLU(I) = SMAX(J, 0)
    MRED = SMIN(I)
    MGRE = SMIN(J)
    MBLU = SMIN(J)
    MRED = SMAX(I)
    MGRE = SMAX(J)
    MBLU = SMAX(J)
  ENDIF
  IF (STRET .EQ. 0) THEN
    HIRED = 0
    HIGRE = 0
    DO 16 K = 0, SMAX(I, J)
      MRED = HIRED * HIST(1, K)
      CONTINUE
    DO 14 J = 0, SMAX(0, J)
      MGRE = HIGRE * HIST(0, K)
      CONTINUE
    MRED = 255 * HIRED / HCOUNT(I)
    MGRE = 255 * HIGRE / HCOUNT(0)
    MNRED(I) = MNRED(I)
    MNGRE(I) = MNGRE(I)
    MNBLU(I) = MNBLU(I)
  ENDIF
ENDIF

ELSE IF (NBANDS .GE. 3) THEN
  IF (STRET .EQ. 0) THEN
    HIRED = 0
    HIGRE = 0
    DO 18 K = 1, SMAX(2, J)
      MRED = HIRED * HIST(2, K)
      CONTINUE
    DO 16 K = 1, SMAX(1, J)
      MGRE = HIGRE * HIST(1, K)
      CONTINUE
    DO 14 K = 1, SMAX(0, J)
      MBLU = HIBLU * HIST(0, K)
      CONTINUE
    MRED = 255 * HIRED / HCOUNT(2)
    MGRE = 255 * HIGRE / HCOUNT(1)
    MBLU = 255 * HIBLU / HCOUNT(0)
    MNRED(I) = MNRED(I)
    MNGRE(I) = MNGRE(I)
    MNBLU(I) = MNBLU(I)
  ENDIF
ENDIF

Alternate linear stretch:

IF (STRET .EQ. 1) THEN
  MNRED(I) = SMAX(2, J)
  MNGRE(I) = SMAX(1, J)
  MNBLU(I) = SMAX(0, J)
  MRED = SMAX(2)
  MGRE = SMAX(1)
  MBLU = SMAX(0)
  MRED = SMAX(2)
  MGRE = SMAX(1)
  MBLU = SMAX(0)
  MRED = SMIN(2)
  MGRE = SMIN(1)
  MBLU = SMIN(0)
ENDIF
PARAMETER (YES=1)
PARAMETER (NO=0)
INTEGER ROW, COL, NBANDS, NULL, NSETS, ZERO
INTEGER S MEAN (0:MXBND S, 0:MXCL AS)
INTEGER*2 BUF (0:MAXRC, 0:MXBND S)
/* class to be returned */
INTEGER*2 OUTCLS
INTEGER I, J, SET, BAND
/* distances to the mean of a set */
INTEGER D (MXBND S)
REAL*8 DISTSQ, MIN

For NULL = ALL option:
IF (NULL .EQ. 0) THEN
  IF pixel is zero in all elements, set class = 0, exit:
  ZERO = YES
  DO 5 BAND=0, NBANDS-1, 1
    IF (BUF (COL, BAND) .NE. 0) THEN
      ZERO = NO
    ENDIF
  CONTINUE
5
ENDIF
IF (ZERO .EQ. YES) THEN
  MM = 0
  RETURN
ENDIF

For NULL = ANY option:
IF (NULL .EQ. 1) THEN
  IF pixel is zero in any elements, set class = 0, exit:
  ZERO = NO
  DO 4 BAND=0, NBANDS-1, 1
    IF (BUF (COL, BAND) .EQ. 0) THEN
      ZERO = YES
    ENDIF
  CONTINUE
4
ENDIF
IF (ZERO .EQ. YES) THEN
  MM = 0
  RETURN
ENDIF

/* Set initially unclassified */
OUTCLS = 0
/* set initial minimum distance to something large */
MIN = 1.0E10
DO 10 SET=1, NSETS, 1
  DISTSQ = 0
  BAND = 0
  Simulated WHILE loop:
  IF (BAND .GE. NBANDS) THEN
    GOTO 15
  ELSE
    D (BAND) = SM EAN (BAND, SET) - BUF (COL, BAND)
    IF (D (BAND) .GT. MIN) THEN
      BAND = NBANDS
    ELSE
      DISTSQ = D (BAND) * D (BAND) + DISTSQ
      BAND = NBANDS
    ELSEIF (BAND .EQ. NBANDS) THEN
      /* set new min distance to be zero */
      MIN = DISTSQ
      OUTCLS = SET
  ENDIF
10

ENDIF
ENDIF
END
END
END
END
GOTO 20
CONTINUE
15
CONTINUE
10
MM = OUTCLS
RETURN
END

END

This routine calculates the divergence and Transformed Divergence for every possible combination of classes. The divergence is an indicator of separability between two classes.

Formula is adapted from "Introductory Digital Image Processing", Jensen, 1986.

Subroutines DIVERG uses:

MMULT - Multiply two matrices.
MINV - Invert matrix.
MADD - Add or subtract two matrices.
MTRACE - Trace of matrix.
MTRANS - Transpose matrix.
MDET - Determinant of matrix.

SUBROUTINE DIVERG(COVAR, S MEAN, SCOUNT, NBANDS, CLASSES, DIV, TDIV, CORRE)
PARAMETER (NBANDS=7)
PARAMETER (MAXRC=3584)
PARAMETER (MXCL AS=300)
PARAMETER (NEGB = -8.0)
PARAMETER (YES=1)
PARAMETER (NO=0)
IMPLICIT CHARACTER*12 (C)
IMPLICIT REAL*8 (T)
IMPLICIT REAL*8 (S)
INTEGER HI, J, K, B1, B2, BAND, NBANDS, NRM1, CLASSES, BCOMP
INTEGER S MEAN (0:MXBND S, 0:MXCL AS)
REAL*8 COVAR (0:MXCL AS, 0:MXBND S, 0:MXBND S)
REAL*8 CORRE (0:MXCL AS, 0:MXBND S, 0:MXBND S)
REAL*8 DIV (0:MXCL AS, 0:MXCL AS), SUM1, SUM2
REAL*8 TDIV (0:MXCL AS, 0:MXCL AS)
REAL*8 SCOUNT (0:MXCL AS), DEI, DET2
REAL*8 TB G1 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TB G2 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TB V1 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TB V2 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TEMP1 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TEMP2 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TEMP3 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TEMP4 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 SBI (0:MXBND S-1, 0:MXBND S-1)
REAL*8 STEM P1 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 STEM P2 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TCO R1 (0:MXBND S-1, 0:MXBND S-1)
REAL*8 TCO R2 (0:MXBND S-1, 0:MXBND S-1)
INTEGER BI NDS, B2NDS, BAD1 (0:MXBND S-1), BAD2 (0:MXBND S-1)
CHARACTER*7 TB G1
CHARACTER*7 TB G2
COMMON/TITLE/ LOGFIL

Simple calculation if only one band in analysis:
IF (NBANDS .EQ. 1) THEN
  DO 5 B1 = 1, CLASSES
    DO 15 B2 = 3, B1
      IF (COVAR(B1,0,0) .NE. 0 .AND.
          COVAR(B2,0,0) .NE. 0) THEN
        DIV(B1,B2) = (COVAR(B1,0,0)/COVAR(B2,0,0))**2
          ELSE
            DIV(B1,B2) = DIV(B1,B2)/2.
          ENDIF
        ENDIF
      TDIV(B1,B2) = 2000 - 2000. *(EXP(DIV(B1,B2)/NEGB8))
    IF (B2 .NE. B1) TDIV(B2,B1) = DIV(B1,B2)
    IF (B2 .NE. B1) TDIV(B2,B1) = TDIV(B1,B2)
  CONTINUE
  RETURN
ENDIF

Initialize Output matrices DIV and TDIV:
DO 10 I = 0, MXCLAS
  DO 20 J = 0, MXCLAS
    DIV(I,J) = 0
    TDIV(I,J) = 0
  CONTINUE
  CONTINUE

Loop over all possible band combinations, get elements of DIV:
DO 30 B1 = 1, CLASSES
  IF (NBANDS .NE. 1) THEN
    HANDLE data dropout in individual bands:
      Run check to see if one or more of B1's bands have mean 0:
      B2BANDS = NBANDS
      DO 200 I = 0, NRM1
        IF (SMEAN(B1,B1) .EQ. 0) THEN
          B2BANDS = B2BANDS - 1
          IF (SMEAN(B1,B1) .EQ. 0) THEN
            B2BANDS = B2BANDS - 1
            BADB1(I) = YES
          ELSE
            BADB1(I) = NO
          ENDIF
        ELSE
          HANDLE data dropout in individual bands:
          RUN check to see if one or more of B1's bands have mean 0:
          B2BANDS = NBANDS
          DO 200 I = 0, NRM1
            IF (SMEAN(B1,B1) .EQ. 0) THEN
              B2BANDS = B2BANDS - 1
              IF (SMEAN(B1,B1) .EQ. 0) THEN
                B2BANDS = B2BANDS - 1
                BADB1(I) = YES
              ELSE
                BADB1(I) = NO
              ENDIF
            ELSE
              HANDLE data dropout in individual bands:
            RUN check to see if one or more of B1's bands have mean 0:
            B2BANDS = NBANDS
            DO 200 I = 0, NRM1
              IF (SMEAN(B1,B1) .EQ. 0) THEN
                B2BANDS = B2BANDS - 1
                IF (SMEAN(B1,B1) .EQ. 0) THEN
                  B2BANDS = B2BANDS - 1
                  BADB1(I) = YES
                ELSE
                  BADB1(I) = NO
                ENDIF
              ELSE
                HANDLE data dropout in individual bands:
              RUN check to see if one or more of B1's bands have mean 0:
              B2BANDS = NBANDS
              DO 200 I = 0, NRM1
                IF (SMEAN(B1,B1) .EQ. 0) THEN
                  B2BANDS = B2BANDS - 1
                  IF (SMEAN(B1,B1) .EQ. 0) THEN
                    B2BANDS = B2BANDS - 1
                    BADB1(I) = YES
                  ELSE
                    BADB1(I) = NO
                  ENDIF
                ELSE
                  HANDLE data dropout in individual bands:
                RUN check to see if one or more of B1's bands have mean 0:
                B2BANDS = NBANDS
                DO 200 I = 0, NRM1
                  IF (SMEAN(B1,B1) .EQ. 0) THEN
                    B2BANDS = B2BANDS - 1
                    IF (SMEAN(B1,B1) .EQ. 0) THEN
                      B2BANDS = B2BANDS - 1
                      BADB1(I) = YES
                    ELSE
                      BADB1(I) = NO
                    ENDIF
                  ELSE
                    HANDLE data dropout in individual bands:
                  RUN check to see if one or more of B1's bands have mean 0:

      RETURN
    CONTINUE
  ENDIF
ELSE
  HANDLE data dropout in individual bands:
ENDIF
DO 40 B2 = 1, B1-1

C Information printouts:
C WRITE(CMES,'(A9,13,A6,13)') 'Comparing', B1, ' with ', B2
C CALL SDEST(CMES,0)
C CALL FDEST(LOOPFIL,CMES)
C------------------------------------------------------------------------
C IF (SCOUNT(B1) .GT. 0 .AND. SCOUNT(B2) .GT. 0) THEN
C Run check to see if one or more of B2's bands have mean 0:
B2BANDS = NBANDS
DO 210 I = 0, NRM1
  IF (SMEAN(B1,B2) .EQ. 0) THEN
    B2BANDS = B2BANDS - 1
    BADB2(I) = YES
  ELSE
    BADB2(I) = NO
  ENDIF
CONTINUE
RETURN
ENDIF

C Only compare classes if they have the same mean 0 bands:
IF (B2BANDS .EQ. B1BANDS) THEN
  BCOMPO = YES
  DO 220 I = 0, B1BANDS - 1
    IF (BADB1(I) .NE. BADB2(I)) BCOMPO = NO
  CONTINUE
ENDIF

C Copy class B2's mean and covariance matrix:
NRM1 = B1BANDS - 1
K = 0
DO 36 I = 0, NRM1
  IF (BADB2(I) .EQ. NO) THEN
    SMB2(K,0) = SMEAN(I,B2)
    H = 0
    DO 46 J = 0, NRM1
      IF (BADB2(J) .EQ. NO) THEN
        TCVB2(K,H) = COVAR(B2,I,J)
        TCCOR2(K,H) = CORRE(B2,I,J)
        H = H + 1
      ENDIF
    CONTINUE
    K = K + 1
  ENDIF
CONTINUE
ENDIF

C Simple calculation if only one non-zero band in analysis:
IF (B1BANDS .EQ. 1) THEN
  IF (TCVB1(0,0) .NE. 0 .AND. TCVB2(0,0) .NE. 0) THEN
    DIV(B1,B2) = (TCVB1(0,0) - TCVB2(0,0))**2
      ELSE
        DIV(B1,B2) = DIV(B1,B2)/2.
      ENDIF
    TDIV(B1,B2) = 2000 - 2000. *(EXP(DIV(B1,B2)/NEGB8))
  GOTO 999
ENDIF

C Assemble the 5 individual parts, combine:
CALL MDET(TCVB1,B1BANDS,DET1,MXBANDS)
CALL MDOT(TCVB2, B1BND, DET2, MXBND)

Check if there is no inverse to covar. matrices by checking if determinant is 0:

IF (DET1 .LT. 0.001 .AND. DET1 .GT. -0.001) THEN
WRITE(CMES,'(A48,1X,15,F9.5)') 'Found a singular covariance matrix, class ', B1, DET1
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMES)
DIV(BI, B2) = 0.
TDIV(BI, B2) = 0.
GOTO 999
ELSE
IF (DET1 .LT. 0.001 .AND. DET1 .GT. -0.001) THEN
WRITE(CMES,'(A48,1X,15,F9.5)') 'Found a singular covariance matrix, class ', B1, DET1
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMES)
DIV(BI, B2) = 0.
TDIV(BI, B2) = 0.
GOTO 999
ELSE
CALL MINV(TCVB1, B1BND, TCVB1, MXBND)
CALL MINV(TCVB2, B1BND, TCVB2, MXBND)
ENDIF
ENDIF

CALL MADD(TCVB1, TCVB2, TEMP1, B1BND, B1BND, -1, MXBND, MXBND)
CALL MMULT(TEMP1, TCVB1, TEMP2, B1BND, B1BND, 1, MXBND, MXBND)
CALL MTRACE(TEMP1, B1BND, SUM1, MXBND)
CALL MADD(TCVB1, TCVB2, TEMP1, B1BND, B1BND, -1, MXBND, MXBND)
CALL MMULT(STEMP1, STEMPI, STEMPI, B1BND, B1BND, 1, MXBND, MXBND)
CALL MMULT(STEMP1, STEMPI, STEMPI, B1BND, B1BND, 1, MXBND, MXBND)
CALL MTRACE(STEMP1, STEMPI, STEMPI, SUM2, MXBND)
DIV(BI, B2) = (SUM1+SUM2)/2
Develop transformed Divergence TDIV from DIV:
TDIV(BI, B2) = 2000 - 2000.*EXP(DIV(BI, B2)/MGB)
CONTINUE
INTEGER NR, NC, I, J, PR, PC
DO 10 I=0,NR-1
    DO 20 J=0,NC-1
        MAT2(J,1) = MAT1(I,J)
    CONTINUE
10 CONTINUE
RETURN
END

------------------------------------------------------------
MMULT
------------------------------------------------------------
Multiply(MA,MB,MC,n,m,p) Multiplies matrix MA (n rows by m cols)
by matrix MB (m rows by p cols) to get matrix MC (n rows by p cols).
Matrices are zero indexed.
Matrices are dimensioned FN, PM, and PP in size.

SUBROUTINE MMULT(MA,MB,MC,n,m,p)
    REAL*8 MA(M:FN,1:P:PP),MB(M:FPM,1:PP)
    INTEGER I,J,K,N,M,P,FN,PM,PP
    DO 10 I=0,N-1,1
        DO 20 J=0,P-1,1
            MC(I,J) = 0.0
            DO 30 K=0,M-1,1
                MC(I,J) = MC(I,J) + MA(I,K) * MB(K,J)
            CONTINUE
        CONTINUE
    10 CONTINUE
RETURN
END

------------------------------------------------------------
MINV
------------------------------------------------------------
Invert(A,n,n) Inverts matrix A (square, n row, n col, zero indexed)
into matrix B (same dimensions).

SUBROUTINE MINV(A,N,B,N,B)
    REAL*8 A(1:NP,1:PR),B(1:NP,1:PR)
    INTEGER I,J,K,N,B
    N is size of matrix (square).
    B starts as temporary matrix for putting A into.
    (because the input matrix is destroyed.)
    DO 10 I=1,N-1,1
        DO 20 K=1,N-1,1
            B(I,K) = A(I,K)
        CONTINUE
    10 CONTINUE
    CONTINUE
    DO 30 K=1,N,1
        DO 40 J=1,N,1
            IF (J .NE. K) B(K,J) = B(K,J)/B(K,K)
        CONTINUE
    30 CONTINUE
    CONTINUE
    DO 50 I=1,N,1
        IF (I .NE. K) THEN
            B(I,K) = -1.0/B(K,K)
        ELSE
            CONTINUE
        ENDIF
    50 CONTINUE
RETURN
END

------------------------------------------------------------
MDET
------------------------------------------------------------
Determinant of matrix A (square, n row, n col, zero indexed)
Returns B, the determinant of A.
Matrices are dimensioned PR and PR cols in size.
MDET calls LUDCMP routine.

SUBROUTINE MDET(A,N,B)
PARAMETER(MXBNDS=7)
REAL*8 A(0:PR-1,0:PR-1), B
REAL*8 C(MXBNDS,1:MXBNDS), INDX(MXBNDS)
INTEGER I,J,K,N,PR
N is size of matrix (square).
C starts as temporary matrix for putting A into.
C (because the input matrix is destroyed.)
MAX = 0.0
DO 10 I = 0, N-1, 1
    DO 20 K = 0, N-1, 1
        C[I+1,I+1] = A(I,K)
        IF (C[I,K] .GT. MAX) THEN
            MAX = C[I,K]
        ELSE
            CONTINUE
        ENDIF
    20 CONTINUE
10 CONTINUE
IF (MAX .EQ. 0.0) THEN
    B = 0
ELSE
    CALL LUDCMP(C,N,MXBNDS,INDX,B)
    DO 30 K = 1, N, 1
        B = B * C(K,K)
    30 CONTINUE
ENDIF
RETURN
END

------------------------------------------------------------
LUDCMP
------------------------------------------------------------
Computes LU decomposition of a matrix.
From "Numerical Recipes", Press, Flannery et al. Cambridge
University Press, Oxford.

SUBROUTINE LUDCMP(A,N,PP,INDX,D)
PARAMETER(MMAX=100, TINY=1.0E-20)
REAL*8 A(0:NP,1:PR), D, INDX(NP), VV(MMAX), D
D = 1.0
DO 10 I=1,N
    AAMAX = 0.0
    DO 11 J=1,N
        IF (ABS(A(I,J)) .GT. AAMAX) AAMAX = ABS(A(I,J))
    11 CONTINUE
    IF (AAMAX .EQ. 0.0) THEN
        VV(I) = 1/AAMAX
        CONTINUE
    ELSE
        DO 12 J=1,N
            IF (J .GT. I) THEN
                DO 13 K=1,J-1
                    SUM = SUM + A(I,K)*A(J,K)
                13 CONTINUE
                A(I,J) = SUM
            ENDIF
        12 CONTINUE
    ENDIF
10 RETURN
END
CONTINUE
ENDIF
AAMAX=A
DO 16 I=J,N
SUM=A(I,J)
IF (J.GT.I) THEN
DO 15 K=I,J-1
SUM=SUM-A(I,K)*A(K,J)
15 CONTINUE
A(I,J)=SUM
ENDIF
DUM=V(I)*ABS(SUM)
IF (DUM.GE.AAMAX) THEN
IMAX=I
AAMAX=DUM
ENDIF
CONTINUE
IF (J.NE.IMAX) THEN
DO 17 K=1,N
DUM=A(I,IMAX)
A(IMAX,K)=A(I,J)
A(J,K)=DUM
17 CONTINUE
D=D
V(V)=-V(V)
ENDIF
INDEX(I)=IMAX
IF (J.NE.N) THEN
IF (A(J,J).EQ.0.) A(J,J)=TINY
DUM=A(J,J)
DO 18 J=J+1,N
A(I,J)=A(I,J)*DUM
18 CONTINUE
ENDIF
CONTINUE
IF (A(N,N).EQ.0.) A(N,N)=TINY
RETURN
END

REAL*8 STDIJ(MXBNDS,0,MXCLASS), STMAXJ(0,MXCLASS)
REAL*8 SCOUNT(0,MXCLASS)
REAL*8 DBARJ(0,MXCLASS)
INTEGER NBANDS, CLASSES
INTEGER MAX(0,MXBNDS)
INTEGER SMEAN(0,MXBNDS,0,MXCLASS), BNLOC(0,MXBNDS)
INTEGER I, BAND, SET, SIZE, ISTRETCH
INTEGER*4 IAR
CHARACTER*8 TABFIL
COMMON/TFILE/ LOGFIL
C Parameters for enhancement tables:
DIMENSION IRED(15), IGREEN(15), IBLUE(15)
DATA IRED/255,0,255,0,255,0,255,255,255,255,237,0,227,255,0,255/
CNEWOUT = CFI(OTAREA)
NRM1 = NBANDS - 1
ISTRETCH = STRETCH
TABFIL(1:4) = CNEWOUT(9:12)
TABFIL(5:8) = 'SIG'
CALL LMD(TABFIL)
WRITE(CMES,'(A19,X,A8)') 'Signature file: ', TABFIL
CALL DRED(CMES,0)
CALL DSETD(TABFIL,CMES)
WRITE(CMES,'(A28,1X,A14)') 'Contains signatures for ',
C
CLASSES(1), ' Classes'
CALL DSETD(CMES,0)
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(A5)') 'Class values doubled for 7 bit display.'
CALL DFDSET(CMES,0)
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(A8,24X)') TABFIL
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(13,1X,A10)') CLASSES(1), 'Classes'
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(13,1X,A10)') NBANDS, 'Bands'
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(A29)') '--Max Data values per band--'
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(7,15,1X)') (MAX(I), I=0,NRM1)
CALL DFDSET(TABFIL,CMES)

C Write out mean array to file:
WRITE(CMES,'(25X,A17)') 'Band Means--'
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(A5,2X,A6,1X,7(3X,A11,11,A1))')
C 'Class', 'Counts', '(', 'BNLOC(I)', ')', I=0,NRM1)
CALL DFDSET(TABFIL,CMES)
DO 450 I=0, CLASSES, 1
WRITE(CMES,'(14,2X,F7.0,7(1X,15))')
C I*ISTRETCH, SCOUNT(I), (SMEAN(BAND,I), BAND=0,NRM1)
CALL DFDSET(TABFIL,CMES)
450 CONTINUE
WRITE(CMES,'(A28)') 'Class covariance matrices:
CALL DFDSET(TABFIL,CMES)
WRITE(CMES,'(14,11)') 0, ' (CLASS)'
CALL DFDSET(TABFIL,CMES)
DO 601 BAND = 0, NRM1
WRITE(CMES,'(7(F9.3,1X))') (0.0, J=0,NRM1)
CALL DFDSET(TABFIL,CMES)
601 CONTINUE
DO 595 I=1, CLASSES
WRITE(CMES,'(14,11)') I*ISTRETCH, ' (CLASS)'
CALL DFDSET(TABFIL,CMES)

SIGOUT
Function to save signature classes data with means of sets,
and covariances.

Note that a stretch factor of 2 has been used throughout this
program because of the 7 bit nature of the color display.

McIDAS LMFIL operations are not used due to their tendency
to append unwanted data at EOF. Unit 19 is used to write
the .SIG signature file.

SUBROUTINE SIGOUT(OTAREA, NBANDS, MAX, SMEAN, SCOUNT, DBARJ, STDIJ, C
STMAXJ, CLASSES, BNLOC, COVAR)
IMPLICIT CHARACTER*12 (C)
PARAMETER (MXBNDS=7)
PARAMETER (MXCLASS=300)
PARAMETER (STRETCH=2)
CHARACTER*70 CMES
REAL*8 COVAR(0,MXCLASS,0,MXBNDS,0,MXBNDS)

--- End of code ---
DO 606 BAND = 0, NBM1
   WRITE(CMES, '(7(F9.3,1X))') (COVAR(I,BAND,J), J=0,NBM1)
   CALL FDST(FTABFIL,CMES)
   CONTINUE
406
195 CONTINUE

     Write out stds to file:
   WRITE(CMES, '(27X,A17)') '---Band Stds---'
   CALL FDST(FTABFIL,CMES)
   WRITE(CMES, '(A5,2X,A6,5X,A4,7(3X,A1,11,A1))')
   C 'Class','Counts','dbar', ('+BNDLOC(I),'),I=0,NBM1)
   CALL FDST(FTABFIL,CMES)
   DO 455 I=0, CLASSES, 1
      WRITE(CMES, '(14,2X,F8.1,1X,F7.2,7(1X,F5.1))') I*ISTRETCH,
      C SCOUNT(I), DBARJ(I), (STDIJ(BAND,J),BAND=0,NBM1)
   CALL FDST(FTABFIL,CMES)
   CONTINUE
155
RETURN
END
SUBROUTINE MAIN

MDCLAS -- SPECTRA L CLASSIFY A MULTI-BAND IMAGE

PARAMETERS:
INAREA -- AREA TO BE CLASSIFIED
OUTAREA -- FINAL CLASSIFIED AREA WITH N CLASSES.
SIGAREA -- AREA WITH .SIG STATISTICS FILE

KEYWORDS:
BANDS: NUMBER OF AREA BANDS TO CLASSIFY,
DEFAULT IS ALL BANDS IN AREA.
USER DEFINES THE BANDS TO USE:
BANDS= BAND1 BAND2 ... BANDN

NULL: DATA DROP-OUT DECISION.
ANY: CLASSIFY AS ZERO IF ANY BAND IN IMAGE=0.
ALL: CLASSIFY AS ZERO IF ALL BANDS = 0.

This program must be linked with mdsgin.c when compiling.

Written by Gerard Pelitzer
UW SSEC 5/1/1992

C

IMPLICIT CHARACTER*12 (C)
IMPLICIT REAL*8 (D)

CHARACTER*70 CMES, CBLNK
CHARACTER*8 LOGFIL, SIGFIL
INTEGER*4 INAREA, OTAREA, SIGAREA, TEST
INTEGER CLASSES, NBANDS, BANDLOC (0:MXBNDNS)
INTEGER IDIR (4), ABANDS
INTEGER NROWS, NCOLS, NULL
COMMON/TFILE/ LOGFIL

CALL SDEST ('BEGIN', '0')

INAREA=IPF (1,0)
OTAREA=IPF (2,0)
SIGAREA=IPF (3,0)
IF (INAREA .LT. 0 .OR. INAREA .GT. 500000) THEN
WRITE (CMES, '(1X,A15,1X,15)') 'INVALID AREA:', INAREA
RETURN
ENDIF

C Open output text file, use FDEST routine to print data to logfils
CTEMP = CFI (OTAREA)
LOGFIL(1:4) = CTEMP (9:12)
LOGFIL(5:8) = 'LOG'
CALL LMD (LOGFIL)
WRITE (CMES, '(1X,A15,1X,12)') 'SUMMARY FILE', LOGFIL
CALL FDEST (LOGFIL, CMES)
CALL SDEST (CMES, 0)
WRITE (CMES, '(1X,A15,1X,12)') 'MCCLAS UNSUPERVISED CLASSIFICATION'
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
WRITE (CMES, '(1X,A15,1X,12)') 'INPUT AREA:', INAREA
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)

C Set up signature file:
CTEMP = CF1 (SIGAREA)
SIGFIL(1:4) = CTEMP (9:12)
SIGFIL(5:8) = 'SIG'
WRITE (CMES, '(1X,A15,1X,12)') 'STATISTICS FILE', SIGFIL
CALL FDEST (LOGFIL, CMES)
CALL SDEST (CMES, 0)
WRITE (CBLNK, '(A1)')
CALL SDEST (CBLNK, 0)
CALL FDEST (LOGFIL, CBLNK)
WRITE (CMES, '(1X,A20)') 'IMAGE CONTENTS'
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)

C Open area to see how big, number of bands:
CALL HOMBIG (INAREA, NROWS, NCOLS)
WRITE (CMES, '(1X,A15,15)') 'ROWS:', NROWS
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
WRITE (CMES, '(1X,A15,15)') 'COLUMNS:', NCOLS
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
IF ( (NROWS .GT. MAXROW) .OR. (NCOLS .GT. MAXCOL) ) THEN
CALL SDEST ('AREA TOO BIG TO PROCESS - EXITING!', 0)
CALL FDEST (LOGFIL, CMES)
RETURN
ENDIF

CALL OPHA (INAREA)
CALL ARAOPT (INAREA, 1, 'SPAC', 4)
CALL READD (INAREA, IDIR)

ABANDS = IDIR (14)
WRITE (CMES, '(1X,A20,15)') 'BANDS IN AREA:', ABANDS
CALL SDEST (CMES, 0)
CALL FDEST (LOGFIL, CMES)
IF (ABANDS .LT. 1) THEN
CALL SDEST ('NOT ENOUGH BANDS TO PROCESS - EXITING!', 0)
CALL FDEST (LOGFIL, CMES)
RETURN
ENDIF

CALL CLSARA (INAREA)

C Read in bands to use in classification:
NBANDS = INFO ('BANDS')
IF (NBANDS .EQ. 0) THEN
NBANDS = ABANDS
DO 20 I = 1, NBANDS
BANDLOC (I-1) = I
20 CONTINUE
ENDIF
DO 10 I = 1, NBANDS

MDCLA$i$ (MDM CLASSIFIER)

Function to classify image using clustering.

This routine reads in the bands of an image, does a classification, and writes the image to disk.

A list of the important variables used:

- $\text{buf}(N)$: The buffer of band $N$: a row of input data.
- $\text{outdata}$: The output data: a row of classifications.
- $\text{classes}$: The number of classes.

---

Routines usclas calls:

- Function MDM(set, row, col, buf, smear)
- Routine to perform Min. Dist. to Mean classification. Returns classification type as an int for pixel in column col.

Subroutine ENHANCE(TAB, Nbands, MAX, SMEAN, MCLASS)
- Routine to save enhancement tables with means of sets. Saves class means from bands 1, 2, and 3 as colors for class.

Subroutine CXXCTR(TDIV, CIASES, MCLASS)
- Routine to print out a square matrix.

Subroutine SIGIN(SIGCIL, CIASES, Sbands, SMEAN)
- Routine to read in class signatures.

Subroutine SIGOUT(TOTAREA, Nbands, MAX, SMEAN, SCOUNT, DBARJ, STD, SCLASS, BNDLOC, COVAR)
- Routine to write out class signatures.

---

MDCLA$i$ (INAREA, OTAREA, SAREA, NBANDS, NRUNS, NCOLS, BNDLOC, NULL)

- MXRBDS sets max number of bands allowed in input image.
- MXBANDS sets maximum number of rows and cols in image.
- MXCLAS sets maximum number of output data classes.
- THRSH sets default percent change of pixels allowed.
- MXITER set maximum number of allowed iterations.
- MCUTOFF is the percent of data for practical lower/upper bounds.

IMPLICIT CHARACTER*12 (C)
IMPLICIT REAL*8 (D)
PARAMETER (MXRBDS=7)
PARAMETER (MXCOL=3584)
PARAMETER (MAXROW=1024)
PARAMETER (MAXCOL=3584)
PARAMETER (MAXCLAS=27)
PARAMETER (THRESH=4)
PARAMETER (MIXTRIX=64)
PARAMETER (DCUTOFF=0.006)

CHARACTER*70 CMES

CHARACTER*8 TRILF, SIGCIL

INTEGER NBANDS, NRUNS, NCOLS, BNDLOC(0:MXRBDS), NULL
INTEGER*4 INAREA, OTAREA, SAREA, MDM, DATAW, NAV2
INTEGER*4 OUTDAT(0:MAXRC), OUDOUT(0:MAXCOL), OUTROW
INTEGER*2 BUF(0:MAXRC, 0:MAXRBDS), NLEM(0:MAXRC)
INTEGER SMEAN(0:MAXRBDS), SMIN(0:MAXRBDS)
INTEGER NSMEAN(0:MAXRBDS), SCLAS
INTEGER IDIR(64), I, J, K, L, ROW, COL, BAND, CLASSES
INTEGER MXITER, SKIP, CSTRIP, CHECK, NRM1, NCM1, NBM1
INTEGER VAL, LEV, STAR, START, SCLAS
INTEGER MEAN(0:MXRBDS), MAX(0:MXRBDS), SMAX(0:MXRBDS)
REAL*8 SNMEAN(0:MAXRBDS, 0:MXCLAS)
REAL*8 SNMEAN(0:MAXRBDS, 0:MXCLAS)
REAL*8 HIST(0:MAXRBDS, 0:255)
COMMON/TLT/L/ LOGFIL

C Statistics variables:

INTEGER*4 OCLCL, PREI, PREJ, II, JJ
INTEGER*4 MINI(0:MXCLAS), MINJ(0:MXCLAS)
REAL*8 STDJ(0:MXRBDS, 0:MXCLAS), STMAXJ(0:MXCLAS)
REAL*8 STDAVG, STDIN, STMINJ(0:MXCLAS)
REAL*8 DBARJ(0:MXCLAS), DWFTJ(0:MXRBDS, 0:MXCLAS), DBAR
REAL*8 SDIST(0:MXCLAS, 0:MXCLAS), MINSD, MAXSD
REAL*8 COVAR(0:MXCLAS, 0:MXCLAS)
REAL*8 CORR(0:MXCLAS, 0:MXRBDS, 0:MXCLAS)
CALL SIGN(SIGFIL, CLASSES, SBANDS, SMAX)
IF (NBANDS .GT. SBANDS) THEN
WRITE (CMS, '(A59)')
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMS)
RETURN
ENDIF
IF (CLASSES .LT. 1) THEN
WRITE (CMS, '(A39)') 'SIGNATURE READ ERROR (NO CLASSES)'
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMS)
RETURN
ENDIF

--- Initial Statistics Extraction ---
C Find the mean and maximum values for each data band:
C Find image histogram, get min, max, stretch values (smin, smax):
C
SKIP = 1
CSKIP = 1
DO 10 I = 0, NBANDS, 1
MEAN(I) = 0
MAX(I) = 0
SMIN(I) = 0
SMAX(I) = 0
NWMEAN(I) = 0
DO 15 J = 0, 255
HIST(I, J) = 0
10 CONTINUE
15 CONTINUE
NCMI = NCOLS - 1
NBMI = NBANDS - 1
NRMI = NROWS - 1
DO 20 ROW = 0, NRMI, SKIPF
DO 30 BAND = 0, NBANDS - 1, 1
C
REDARA(AREA, LINE, STARTLEM, #ELEMENTS, BAND, IARRAY)
CALL REDARA(INAREA, ROW, 0, NCOLS, BNDCALL, NEWIN)
DO 35 COL = 0, NCMI, SKIPF
NWMEAN(BAND) = NWMEAN(BAND) + NNEWIN(COL)
IF (MAX(BAND) .LT. NEWIN(COL)) THEN
MAX(BAND) = NEWIN(COL)
ENDIF
HIST(BAND, NEWIN(COL)) / (255**((DATAW-1))) =
HIST(BAND, NEWIN(COL)) / (255**((DATAW-1))) + 1
35 CONTINUE
30 CONTINUE
20 CONTINUE
C
DO 50 BAND = 0, NBANDS, 1
C
MEAN(BAND) = SKIP**2 * (NWMEAN(BAND) / (NCOLS * NROWS))
WRITE(CMES, 51) 'MEAN OF BAND ', BAND+1, ',', ' ('
BDNDCALL(BAND), ')', ' MEAN(BAND)
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMS)
WRITE(CMES, 51) 'MAX OF BAND ', BAND+1, ',', ' ('
BDNDCALL(BAND), ')', ' MAX(BAND)
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMS)
C Use histogram to find a minimum for stretching:
J = 0
SUM = 0
55 CONTINUE
SUM = SUM + HIST(BAND, J)

IF (J .GE. 255) THEN
  SMIN(BAND) = 0
GOTO 56
ENDIF
IF (SUM .LE. (NROWS*NCOLS)*DCUTOF/SKIPF) THEN
  J = J + 1
GOTO 55
ENDIF
SMIN(BAND) = J
Use histogram to find a maximum for stretching:
J = 255
SUM = 0
CONTINUE
SUM = SUM + HIST(BAND,J)
IF (J .GE. 0) THEN
  SMAX(BAND) = 0
GOTO 57
ENDIF
SMAX(BAND) = J
WRITE(CMES,'(F6.2,A21,1X,15)') DCUTOF*100,
C
  "data lower bound: ", SMIN(BAND)
CALL SDST(E,CMES,0)
CALL FDST(LOGFIL,CMES)
WRITE(CMES,'(F6.2,A21,1X,15)') (1-DCUTOF)*100,
C
  "data upper bound: ", SMAX(BAND)
CALL SDST(E,CMES,0)
CALL FDST(LOGFIL,CMES)
CONTINUE
FORMAT(1X,A14,I2,1X,A1,11,A1,12,2X,17)
Start loop over rows and columns, convert input to output:
DO 60 ROW=0, NROW, NSAMP
DO 70 BAND=0, NBM1, 1
  function reada(AREA,LINE,STARTL,R,ELEMENTS,BAND,IARRAY)
  CALL REDARA(INAREA,ROW,0,NCOLS,NBLOC(BAND),NEWWIN)
  CALL 75 COL=0, NCM1, CSSKIP
  BUF(COL,BAND) = NEWWIN(COL)
CONTINUE
Write output:
CLASSIFY:
DO 80 COL=0, NCM1, CSSKIP
OUTDAT(COL) = MDM(CMCLASS,ROW,COL,NBANDS,BUF,SMEAN,NULL)
CONTINUE
Develop new mean and point count for data:
DO 110 COL=0, NCM1, CSSKIP
DO 120 BAND=0, NBM1, 1
  NSMEAN(BAND,OUTDAT(COL)) = NSMEAN(BAND,OUTDAT(COL)) +
  BUF(COL,BAND)
C
  CONTINUE
  SCOUNT(OUTDAT(COL)) = SCOUNT(OUTDAT(COL)) +1
C
  CONTINUE
Save to array:
DO 135 COL=0, NCM1, 1
OLDOUT(COL,ROW) = OUTDAT(COL)
CONTINUE
DO 135 CONTINUE

---Band Means---
C
WRITE(CMES,'(A13,5X,T(1X,11,A1,11,A1,12,2X,17))') 1, SCOUNT(I)*(CSSKIP**2),
C
  (SMEAN(BAND)), BAND=0, NBM1
C
CALL SDST(E,CMES,0)
CALL FDST(LOGFIL,CMES)
DO 137 J=0, NCLASS, 1
WRITE(CMES,'(13,2X,F8.1,1X,7(1X,15))') 1, SCOUNT(I)*(CSSKIP**2),
C
  (SMEAN(BAND)), BAND=0, NBM1
C
CALL SDST(E,CMES,0)
CALL FDST(LOGFIL,CMES)
CONTINUE

CONTINUE

Find pairwise cluster distances:
DO 330 I=0, CLASSES-1, 1
  MINI(I) = 50
  MINJ(J) = 0
  DO 340 J=I+1, CLASSES, 1
    DO 345 BAND=0, NBM1, 1
      SDIST(I,J) = SDIST(I,J) + (SMEAN(BAND,I)-SMEAN(BAND,J))^2
  END DO 345
  CONTINUE
  SDIST(I,J) = SQRT(SDIST(I,J))
  CONTINUE

MINJ(CLASSES) = 0
Loop over rows and columns, compute dbar, stds:
DO 210 ROW=0, NRMI, 1
  CALL REDARA(INAREA, ROW, 0, NCOLS, BNDLOCS, BAND, NEWWIN)
  DO 220 COL=0, NCM1, 1
    BUF(COL, BAND) = NEWWIN(COL)
    DNETJ(BAND, OLDOUT(COL, ROW)) = DNETJ(BAND, OLDOUT(COL, ROW))
    STDIJ(BAND, OLDOUT(COL, ROW)) = STDIJ(BAND, OLDOUT(COL, ROW))
    CONTINUE
  END DO 220
  CONTINUE
  Covariance calculations:
  DO 550 COL=0, NCM1, 1
    DO 560 J=0, 1, 1
      IF (I .NE. J) THEN
        COVAR(OLDOUT(COL, ROW), I, J) = COVAR(OLDOUT(COL, ROW), I, J)
      ENDIF
    END DO 560
    CONTINUE
  END DO 550

Diagonal covariance elements:
DO 270 BAND=0, NBM1, 1
  IF (SCOUNT(I) .GT. 1) THEN
    COVAR(I, BAND, BAND) = STDIJ(BAND, I)
  ENDIF
  CONTINUE

Covariance elements:
DO 580 J=0, BAND, 1
  IF (SCOUNT(I) .GT. 1) THEN
    COVAR(I, BAND, J) = COVAR(I, BAND, J) / (SCOUNT(I)-1)
  ELSE
    COVAR(I, BAND, J) = 0
  ENDIF
  CONTINUE
END DO 580

C Write out correlation matrix, pairwise distances matrix:
C Convert covariance to correlation:
C Don't count the diagonal due to 1 pixel classes:
IF (STMIN .GT. STMINJ(I) .AND. STMINJ(I) .GT. 0.01)
  STMAX = STMINJ(I)
  IF (STMAX .LT. STMAXJ(I))
    STMAX = STMAXJ(I)
    DBARJ(I) = SQRT(DBARJ(I))
    DBAR = DBAR + DBARJ(I)*SCOUNT(I)
  ENDIF
  CONTINUE

WRITE (CMES, *(A19)*) 'For all classes:
  CALL DSEST(CMES,0)
  CALL DFDST(LOGFILE,CMES)
WRITE (CMES, *(A11,F9.2)*) 'STMIN = ',STMIN
  CALL DSEST(CMES,0)
  CALL DFDST(LOGFILE,CMES)
WRITE (CMES, *(A11,F9.2)*) 'STMAX = ',STMAX
  CALL DSEST(CMES,0)
  CALL DFDST(LOGFILE,CMES)
WRITE (CMES, *(A11,F9.2)*) 'STD AVG = ',STD AVG
  CALL DSEST(CMES,0)
  CALL DFDST(LOGFILE,CMES)
WRITE (CMES, *(A11,F9.2)*) 'DBAR = ',DBAR
  CALL DSEST(CMES,0)
  CALL DFDST(LOGFILE,CMES)
C Write out variance array to screen:
WRITE (CMES, *(A34)*) 'Adjusted cluster means:
  CALL DFSEST(LOGFILE,CMES)
WRITE (CMES, *(A34)*) 'Band means:
  CALL DFDSEST(LOGFILE,CMES)
WRITE (CMES, *(A5,2X,A6,2X,A6,2X,A4,2X,A7,2X(AI,11,2)))
C 'Class', 'Counts', 'Stdmax', 'Dbars', (',', '(BNDLOCS(I),1),', I=0, NBM1)
  CALL DFDST(LOGFILE,CMES)
DO 450 I=0, CLASSES, 1
  WRITE (CMES, *(A13,2X,F7.0,1X,F7.2,1X,F7.2,7X(I,15))) I,
    SCOUNT(I)*CSKIP**2, STMAXJ(I), DBARJ(I),
    SMEAN(BAND(I),1), BAND=0, NBM1
  CALL DFDST(LOGFILE,CMES)
END DO 450
C Write out correlation matrix, pairwise distances matrix:
C Convert covariance to correlation:
C Don't count the diagonal due to 1 pixel classes:
DO 590 I = 1 - CLASSES
  DO 600 BAND = 0, NBM1
    IF (SCOUNT(I) .NE. 0) THEN
      DO 610 J = 0, BAND
        IF (COVAR(I,BAND,J) .NE. 0 .AND.
          STDIJ(BAND,I) .NE. 0 .AND.
          STDIJ(J,BAND,I) .NE. 0) THEN
          CORRE(I,BAND,J) = COVAR(I,BAND,J) /
          (STDIJ(BAND,I)*STDIJ(J,BAND,I))
        ELSE
          CORRE(I,BAND,J) = 0
        ENDIF
      END DO 610
      FILL other half of symmetric correlation matrix:
      IF (J .NE. BAND) CORRE(I,J,BAND) = CORRE(I,BAND,J)
    ENDIF
  END DO 600
END DO 590
C Write out correlation matrix, pairwise distances matrix:
C Convert covariance to correlation:
C Don't count the diagonal due to 1 pixel classes:
WRITE(CMES,'(A11,13)') ' CLASS: ',I
CALL FDEST(LOGFIL,CMES)
DO 606 BAND = 0, NBM1
WRITE(CMES,'(10(F5.3,1X))') (CORRE(I,BAND,J), J=0,NBM1)
CALL FDEST(LOGFIL,CMES)
6 CONTINUE
5 CONTINUE
WRITE(CMES,'(A39)') 'Distances between Cluster means:'
CALL FDEST(LOGFIL,CMES)
CALL SDEST(CMES,0)
CALL CXCOUT(SDEST,CLES,S,MNCLS)

WRITE file out to output file name
DO 170 ROW = 0, NRM1, 1
DO 180 COL = 0, NCM1, 1
OUTDAT(COL) = ODLOUT(COL,ROW)**2
170 CONTINUE
CALL PACK(NCOLS,OUTDAT,OUTDAT)
CALL WRTARA(OTAREA,ROW,OUTDAT)
180 CONTINUE
CALL STAMP(OTAREA)
CALL CLSARA(OTAREA)
CALL CLSARA(INAREA)

Save an enhancement table along with the output image:
WRITE(CMES,'(A30)') 'OUTPUT COLOR ASSIGNMENTS: '
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A24,2X,A1,1X,13,1X,A1)') ' RED FROM BAND 1 ',
' (', BDLOC(0), ')',
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A24,2X,A1,1X,13,1X,A1)') ' GREEN FROM BAND 2 ',
' (', BDLOC(1), ')',
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A24,2X,A1,1X,13,1X,A1)') ' BLUE FROM BAND 3 ',
' (', BDLOC(2), ')',
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)

Write MCDAS .ET color look-up table file:
CALL ENHANCE(OTAREA,SGAREA,NBANDS,MAX,SMAX,SMIN,SMEAR,CLASSES)
RETURN
END

FUNCTION

Function to write out a triangular CLASSES by CLASSES matrix
Matrix elements divided by SIZE in output.

SUBROUTINE CXCOUT(RMAT,CLASSES,MNCLS)
REAL*8 RMAT(0:MXCLS,0:MXCLS)
INTEGER CLASSES,1,J,J,JJ,CHECK,SIZE
INTEGER DMAT(0:20)
COMMON/FILE/LOGFIL
CHARACTER*70 CMES

Check the maximum digit size of the matrix:
DO 468 II = 0, CLASSES-1
DO 469 JJ = 0, II
IF (RMAT(JJ,II) .GT. MAX) MAX = RMAT(JJ,II)
468 CONTINUE
469 CONTINUE
IF (MAX .GT. 9999) THEN
SIZE = 10
WRITE(CMES,'(5X,2A4,13,A1)') ' (Value shown = distance/'
C SIZE, ')',
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
ELSE
SIZE = 1
ENDIF
C Printout:
DO 470 JJ = 0, CLASSES
WRITE(CMES,'(A1)') ' '
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
IF (JJ .LT. CLASSES) THEN
J = JJ + 1
ELSE
J = CLASSES
ENDIF
WRITE(CMES,'(A5,13,(I3,2X))') ' Clusters (I,J = 13*I1,JJ)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
DO 460 II = JJ, CLASSES, 1
IF (II .GT. 13) .GT. 12) THEN
CHECK = II + 12
ELSE
CHECK = I
ENDIF
DO 11 J = II + 13, CHECK
DMAT(J,J) = RMAT(J,J)/SIZE
11 CONTINUE
WRITE(CMES,'((I3,1X,13,(I4,1X))') I,
C (DMAT(J,J),J=II+13,CHECK)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
460 CONTINUE
470 CONTINUE
RETURN
END

-------------------------------------------------------------------
ENHANCE

This routine copies the .SIG file .ET color scheme, if present. If not, it generates a new one.

Function will save enhancement tables with means of sets.
Saves class means from bands 1, 2, and 3 as colors for class.

Note that a stretch factor of 2 has been used throughout this program because of the 7 bit nature of the color display.

MCDAS LWFILE operations are not used due to their tendency to append unwanted data at EOF. Unit 19 is used to write the .ET enhancement table file.

-------------------------------------------------------------------
SUBROUTINE ENHANCE(IAR, SGAREA, NBANDS, MAX, SMAX, SMIN, SMEAR, CLASSES)
IMPLICIT CHARACTER*12 (C)
PARAMETER (MXBANDS=7)
PARAMETER (MXCLAS=127)
PARAMETER (STRETCH=2)
CHARACTER*70 CMES
INTEGER NROWS, NCOLS, NBANDS, CLASSES, SGAREA
INTEGER MAX(0:MXBANDS), SMAX(0:MXBANDS), SMIN(0:MXBANDS)
INTEGER SMAX(0:MXBANDS), SMIN(0:MXBANDS)
INTEGER MXRED, MXGRE, MXBLU
INTEGER MINRED, MINGRE, MINBLU
INTEGER I, BAND, SET, TTAB(816), SIZE
INTEGER IAR
REAL*4 REDSTR, GRESTR, BLUSTR
CHARACTER*7 TABFIL
COMMON/FILE/- LOGFIL

Parameters for enhancement tables:
DIMENSION IRED(15), IGREEN(15), IBLUE(15)
DATA FULL/816/, GRE/256/, BLU/512/, RED/769/, GGRE/785/, GBLU/801/
DATA IRED/255, 0, 255, 0, 255, 0, 255, 255, 255, 255, 255, 0, 255, 0, 255/
DATA IBLU/255, 0, 255, 0, 255, 0, 255, 255, 255, 255, 255, 255, 255, 0, 255/
DATA IGREEN/0, 255, 255, 0, 255, 255, 255, 0, 255, 127, 127, 127, 0, 127, 0, 127/

Attempt to read in old .ET file and copy it to output .ET file:
CTEMP = CFI(SGAREA)
TABFIL(1:4) = CTEMP(9:12)
TABFIL(5:7) = ' .ET'
OPEN(UNIT=39, FILE=TABFIL, ACCESS='SEQUENTIAL', FORM='UNFORMATTED',
C STATUS='OLD', ERR=99)
READ(UNIT=39, ERR=99, END=99) (TTAB(J), J=1, 816)
CLOSE(UNIT=39)
CONTINUE
IF (J .GE. GBLU+13) THEN
CNUMOUT = CFI(IAR)
TABFIL(1:4) = CNUMOUT(9:12)
TABFIL(5:7) = ' .ET'
CALL LMD(TABFIL)
READ(UNIT=19, FILE=TABFIL, ACCESS='SEQUENTIAL', FORM='UNFORMATTED',
C ERR=99)
WRITE(UNIT=19, ERR=99) (TTAB(J), J=1, GBLU+1)
CLOSE(UNIT=19)
WRITE(CMES,'(A16, A12)') 'STATISTICS FILE ', TABFIL
CALL FDEST(LOGFIL, CMES)
CALL SDEST(CMES, 0)
RETURN
ENDIF

If here, than transferring previous .ET didn't work. Develop new one:
CONTINUE
CNUMOUT = CFI(IAR)
TABFIL(1:4) = CNUMOUT(9:12)
TABFIL(5:7) = ' .ET'
CALL LMD(TABFIL)
WRITE(CMES,'(A31,A9)') 'Building new enhancement file:', TABFIL
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMES)
WRITE(CMES,'(A28, A14, A9)') 'Contains look up table for ',
C CLASSES, ' Classes',
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMES)
WRITE(CMES,'(A45)') 'Class values doubled for 7 bit display,'
CALL SDEST(CMES, 0)
CALL FDEST(LOGFIL, CMES)
WRITE(CMES,'(A25)') 'CLASS RED GREEN BLUE '
CALL SDEST(CMES, 0)
INTEGER Semean(0:MAXBND$0:MXCLAS$)
INTEGER*2 Buf(0:MAXRC0:MAXBND$)
C/* class to be returned */
INTEGER*2 OUTCLS
INTEGER I, J, SET, BAND
C/* distances to the mean of a set */
INTEGER D(MXBND$)
REAL*8 DISTSQ, MIN
C For NULL = ALL option:
IF (NULL .EQ. 0) THEN
  IF pixel is zero in all elements, set class = 0, exit:
    ZERO = YES
    DO 5 BAND=0, NBANDS-1, 1
      IF (BUF(COL,BAND) .NE. 0) THEN
        ZERO = NO
      ENDIF
    END
  ENDIF
ELSE
  MNR(1) = 0
  MNGRE(1) = 0
  MNBLU(1) = 0
ENDIF
C For NULL = ANY option:
IF (NULL .EQ. 1) THEN
  IF pixel is zero in any elements, set class = 0, exit:
    ZERO = NO
    DO 4 BAND=0, NBANDS-1, 1
      IF (BUF(COL,BAND) .EQ. 0) THEN
        ZERO = YES
      ENDIF
    END
  ENDIF
ELSE
  D(MXBND$) = Semean(BAND,SET) = Buf(COL,BAND)
  IF (D(MXBND$) .GT. MIN) THEN
    BAND = NBANDS
    ELSEIF (D(MXBND$) .GT. DISTSQ) THEN
      DISTSQ = DISTSQ + D(MXBND$)
      BAND = NBANDS
    ELSEIF (D(MXBND$) .EQ. NBANDS-1) THEN
      /* set new min distance to beat */
      MIN = DISTSQ
      OUTCLS = SET
    ENDIF
ENDIF
BAND = BAND + 1
ENDIF
Function MDsigin

This routine reads data from McIDS's .SIG file.
To be used with MDCLASS.PGM (McIDS routine)
The arrays are all indexed from 0 to n-1.

Read in Data:
1) name of sig file
2) number of classes
3) number of bands
4) blank (says "--Max data values--")
5) max-B1 max-B2 ... max-BN
6) blank (says "-----BAND MEANS-----")
7) blank (says "Class Counts " then bands)
8) class# counts mean-B1 mean-B2 ... mean-BN
C of these...
8+C+1) blank (says "class covariance matrices")
8+C+2) class#
8+C+3) COVAR(0,0) COVAR(0,1) ... COVAR(0,N-1)
N of these...
C of these...

Compile with: "fx mdsigin.c"

Written by Gerard Peiter
UW SSEC, 5/1/1992

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>

#define INPUTWIDTH 80 /* Number of chars in inputs. */
#define MXCLASS 127 /* Number of classes. */
#define MAXSBND 16 /* Maximum number of bands. */
#define MXRBNDS 7 /* Maximum number of real bands. */
#define SUCCESS 0
#define FAILURE 0
#define YES 1
#define NO 0
#define SBD 8
#define SIG 9
#define HISTBYTES 51*MAXRBNDS
#define COV 32*MXCLASS

typedef struct Sig{
  char *signature[32];
  long *count;
  char *blank[92];
  int *minvector[MAXSBND];
  int *maxvector[MAXSBND];
  float *covar[32];
  } Sig;

typedef struct Sighead{
  char *lanname[100];
  int *classes;
  int *bands;
  char *blank[24];
  } Sbdheader;

int sigin(char *infile, int *Classes, int *Nbands, int *Smean)
{
  Sbdheader *head, *head2;
  Signature class[MXCLASS];
  int i,j,k;
  int class, classes, nbands, band, eof, *meanptr;
  int max[MAXSBND], min[MAXSBND], mean[MAXRBNDS][MXCLASS];
  long count[MXCLASS];
  float fcount, fmean, cov[32];
  float *covarm;
  FILE *fpin;
  char *triplr, nameline[32];
  char dataline[80];
  char firstline[100];
  char moreline[80];
  char line[70];
  char junkfil[8];
  head = &head2;
  if((fpin = fopen(infile, "rt")) == NULL)
  {
    printf("\nError opening infile\n\n");
    return FAILURE;
  }
  fseek(fpin, 0L, SEEK_SET);
  i = 0;
  eof = NO;
  /* File search */
  while (eof == NO)
  {
    /* Header information: */
    /* First line */
    for (i=0; i< 100 ; i++) firstline[i] = 0;
    for (i=0; i< 100 ; i++) (*head).lanname[i] = 0;
    if (fgets(firstline,100,fpin) == NULL) eof = YES; /* 1 */
    print("\nHeader of data file says: ");
    puts(firstline);
    printf("\n");
    if ((triplr = strrchr(firstline,".")) != NULL) {
      strcpy(triplr, ";" );
      strcpy(("head").lanname, firstline);
    }
    /* Classes */
    for (i=0; i< 80 ; i++) dataline[i] = 0;
    strcpy(dataline, ";0");
    if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 2 */
    j = sscanf(dataline, "%d ", &classes);
    if (j < 1 | j == EOF) eof = YES;
    (*head).classes = classes;
    *Classes = classes;
    /* Bands */
    for (i=0; i< 80 ; i++) dataline[i] = 0;
    strcpy(dataline, ";0");
    if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 3 */
    j = sscanf(dataline, "%d 
" , &nbands);
    if (j < 1 | j == EOF) eof = YES;
    (*head).bands = nbands;
    *Nbands = nbands;
    for (i=0; i< 24 ; i++) (*head).blank[i] = 0;
    /* Develop structures for each signature: */
  }
}
/* Get Maximum Values */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 4 */
for (i=0; i< 80 ; i++) dataline[i] = 0;
strcpy(dataline,"\0");
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 5 */
for (i=0; i< MXBNDS ; i++) max[i] = 0;
} = sscanf(dataline, "%d %d %d %d %d %d", &max[0], &max[1],
&max[2], &max[3], &max[4], &max[5], &max[6]);
if (j < 1 || } == EOF) eof = YES;
/* Blank lines */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 6 */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 7 */
/* Get Mean Values */
for (clas=0; clas< classes ; clas++)
{

for (band=0; band< MXBNDS; band++) smean[band][clas] = 0;
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 8+8+C */
for (i=0; i< 80 ; i++) dataline[i] = 0;
} = sscanf(dataline, "%d %d %d %d %d %d", &l,
&count, &smean[0][clas], &smean[1][clas],
&smean[2][clas], &smean[3][clas], &smean[4][clas],
&smean[5][clas], &smean[6][clas]);
meanptr = smean;
for (clas=0; clas< classes ; clas++)
{

for (i=0; i< MXBNDS+1 ; i++)
{meanptr[i] = smean[i][clas];
meanptr++;}

/* 1 blank line */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 8+C+1 */
/* Get Covariances */
for (clas=0; clas< classes ; clas++)
{

/* 1 blank line */
if (fgets(dataline,80,fpin) == NULL) eof = YES; /* 8+C+2 */
k = 0;
for (band=0; band< nbands ; band++)
{
for (i=0; i< 80 ; i++) moreline[i] = 0;
strcpy(moreline,"\0");
if (fgets(moreline,80,fpin) == NULL) eof = YES; /* 8+C+3 */
} = sscanf(moreline, "%f %f %f %f %f %f", &cova[0],
&cova[1], &cova[2], &cova[3], &cova[4], &cova[5], &cova[6]);
if (j < 1 || } == EOF) eof = YES;
if (k > 32) printf("\nError in covar computation!");
return FAILURE;

for (i=band; i< nbands ; i++)
{ /* covar[k+clas*32] = cova[i];*/
clas = clas.covar[k] = cova[i];
k++;}

fclose(fpin);
return SUCCESS;
ISO.FOR: Standard Routines for Cluster Splitting and Merging

Input parameters from main program are used as:

MERGE: MERGING FACTOR, 0 TO 1,
0 = NO MERGING
MERGE = Q
DEFAULT IS 0.0

SPLIT: SPLITTING FACTOR, 0 TO 10,
0 = NO SPLITTING
SPLIT = R
DEFAULT IS 0.0

Written by Gerard Peltzer
UW SSEC 3/6/1992

Apply lumping (merging) and splitting of classes:
Variable names after Tou & Gonzalez, "Pattern Recognition Principles"

Some definitions:

LUMPF The parameter for merging classes together.
SPLITF The parameter for splitting classes apart.
STDIJ(BAND,CLASS) Std deviation of each class, for each band.
STMAXJ(CLASS) The maximum Std deviation for each class.
STMINJ(CLASS) The minimum Std deviation for each class.
STMAX The maximum Std deviation for all classes.
STMIN The minimum Std deviation for all classes.
STDavg The average Std deviation for all classes.
DNETJ(BAND,CLASS) The average distance of samples in each class from
the class mean, for each band.
DBARJ(CLASS) The average distance of samples from their means.
DBAR Overall average distance of samples from their means.
SDIST(1,2) Spectral distance between means of classes 1 and 2.
MINSD, MAXSD The smallest and largest SDISTs.
MAXLMP The maximum number of lumpings allowed (classes/7).
COVAR(CLASS,1,2) Set of CLASS covariance matrices.
CORRE(CLASS,1,2) Set of CLASS correlation matrices.

SPLITR Ratio Split

Function to split apart clusters as per the ISODATA
definition in "Pattern Recognition Principles".

The threshold is essentially STDMAX/STDMIN>SPLITF for
bands in each class. A SPLITF of 0 gives no splitting.
SPLITF should range from 1 to 10, with 1 splitting all
classes. Note that only 1/3 of the classes will be split
at any iteration.

SUBROUTINE SPLITR(CLASSES,NBANDS,DSCLAS,ITER, SPLITF, SKIPF,
C SRA, SCOUNT, STDIJ, STMAXJ, STMINJ, STDMAX,
C STDMIN, STDAVG, DBARJ, DBAR)
PARAMETER (MAXBANDS=7)
PARAMETER (MAXCLAS=127)
PARAMETER (MINPIX=64)
PARAMETER (YES=1)
PARAMETER (NO=0)
INTEGER SRA, SCOUNT, DSCLAS, CLASSES, I, J, KBAND,
REAL*8 STDIJ, STMAXJ, STMINJ, STDMAX,
REAL*8 DBARJ, DBAR, SPLITF, SKIPF, DSCLAS
INTEGER NCLASS, NCLASS, NBANDS, MINPIX, MAXLMP
INTEGER SPLIT, SMALL
COMMON/TFILE/ LOGFIL
CHARACTER*70 CMES

Exit if splitting operation is not enabled:
IF (SPLITF .LT. 0.001) RETURN

MAXLMP is maximum number of cluster pairs which can be split.

MAXLMP = CLASSES/3
OLDCLS = CLASES
I = 1

1 CONTINUE
SPLIT = NO
SMALL = NO

C Quit if we already have too many classes:
IF ( CLASSES-OLDCLS .GT. MAXSPL) GOTO 999

C Split clusters no matter what if there are few classes:
C Find the class with the largest STD and split it:
IF ( CLASSES .LE. DSCLAS/2 ) THEN
SMAX = 0
DO 10 J = 1, CLASSES
IF ( STMAXJ(J) .GT. SMAX) THEN
SMAX = STMAXJ(J)
I = J
ENDIF
10 CONTINUE
SPLIT = YES
SMALL = YES
ELSE

C Split clusters if this is an odd iteration:
IF ( MOD(ITER,2) .NE. 0 .AND. CLASSES .LT. 2*DSCLAS ) THEN
IF ( NBBANDS .EQ. 1 ) STMINJ(I) = STDAVG
WRITE (CMES,'(A20,1X,I3,1X,F9.3,F9.2)') 'Cluster: ','I,
C STMAXJ(I),STMINJ(I)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
IF ( STMINJ(I) .GT. 0 .AND. DBARJ(I) .GT. DBAR ) THEN
IF ( ( STMAXJ(I)/STMINJ(I) .GT. SPLITF ) .AND. 
C (SCOUNT(I)*SKIPF**2 .GT. 10*MINPIX) ) THEN
SPLIT = YES
WRITE (CMES,'(A20,1X,I3,1X,F9.3,F9.2)') 'Split: ','I,
C SCOUNT(I)*SKIPF**2,10.*(MINPIX+1)
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
SPLIT = YES
ENDIF
ENDIF
ENDIF
ENDIF

IF ( SPLIT .EQ. NO ) I = I + 1

IF ( SPLIT .EQ. YES ) THEN
CLASSES = CLASSES + 1
WRITE (CMES,'(A20,1X,I3)') 'Splitting cluster: ','I
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
STMINJ(I) = 0
STMAXJ(I) = 0
DO 290 K= CLASSES, I+1, -1
SCOUNT(K) = SCOUNT(K-1)
STMAXJ(K) = STMAXJ(K-1)
STMINJ(K) = STMINJ(K-1)
DBARJ(K) = DBARJ(K-1)
DO 300 BAND=0, NBBANDS-1, 1
STDIJ(BAND,K) = STDIJ(BAND,K-1)
SMEAN(BAND,K) = SMEAN(BAND,K-1)
300 CONTINUE
290 CONTINUE
DO 310 BAND=0, NBBANDS-1, 1
SMEAN(BAND,I+1) = SMEAN(BAND,I) + STDIJ(BAND,I)/1.5
SMEAN(BAND,I) = SMEAN(BAND,I) - STDIJ(BAND,I)/1.5
IF ( SMEAN(BAND,I) .LT. 0 ) SMEAN(BAND,I) = 0
310 CONTINUE
I = I + 2
ENDIF
IF (SMALL .EQ. YES) I = 1
IF (I .LE. CLASSES) GOTO 1
999 CONTINUE
RETURN
END

C LUMP
C Function to lump together clusters as per the ISODATA
C definition in "Pattern Recognition Principles".
C The merging threshold is a percentage of the maximum
C spectral distance found between cluster pairs.
C
SUBROUTINE LUMP (CLASSES,NBANDS,OLDCLS,ITER,LUMPF,SMEAN,
C SCOUNT,SDIST,STDIJ,STMAXJ,STMAX,DBARJ,DBAR)
PARAMETER (MXBND=7)
PARAMETER (MXCLAS=127)
PARAMETER (MINPIX=12)
REAL*8 STDIJ(0:MXBND,0:MXCLAS), SDIST(0:MXCLAS,0:MXCLAS)
REAL*8 DBARJ(0:MXCLAS), SCOUNT(0:MXCLAS), STMAXJ(0:MXCLAS)
REAL*8 STMAX, DBAR, MINS, MAXS, LUMPF
INTEGER SMEAN(0:MXBND,0:MXCLAS), ROW, PREI, PREJ, II, JJ
INTEGER NBANDS, CLASSES, OLDCLS, ITER, I, K, BAND
INTEGER MINI(0:MXCLAS), MINJ(0:MXCLAS), MAXLMP
COMMON/TFILE/, LOGFIL
CHARACTER*70 CMES
C Exit if lumping operation is not enabled:
C IF (LUMPF .EQ. 0) RETURN
C Merging and splitting are not allowed together in same iteration:
C IF (CLASSES .EQ. OLDCLS) THEN
C Sort, find lowest distances for merging:
C ROW = 0
C MAXSD = 0
C MAXLMP is maximum number of cluster pairs which can be merged.
C MAXLMP = CLASSES/3
380 CONTINUE
MINS = 1000000
DO 360 I=1, CLASSES-1, 1
DO 370 J=I+1, CLASSES, 1
C IF (ROW .EQ. 0 .AND. SDIST(I,J) .GT. MAXSD)
C MAXSD = SDIST(I,J)
C IF (SDIST(I,J) .LT. MINS) THEN
C JJ = 0
C DO 400 II=0, ROW
C IF (I .EQ. MINI(II) .OR. J .EQ. MINJ(II)) THEN
C We have already found this one, or one of these
C classes is to be merged somewhere else.
C JJ = 1
C ENDIF
400 CONTINUE
C IF (JJ .EQ. 0) THEN
C PREI = I
C PREJ = J
C MINS = SDIST(I,J)
C ENDIF
400 CONTINUE
C If no distances are less than the minimum ,exit the loop!
C IF (ROW .EQ. 0) THEN
WRITE(CMES,'(A31)') 'Distances between classes:
CALL SDEST(CMES,0)
CALL FDEST(LOGFIL,CMES)
WRITE(CMES,'(A19,1X,F6.1)') 'Smallest = ',MINS
CALL SDEST(CMES,0)
CALL FDST(LOGFIL,CMES)
WRITE(CMES,'(A19,1X,F6.1)') 'Largest = ', MAXSD
CALL SDEST(CMES,0)
CALL FDST(LOGFIL,CMES)
WRITE(CMES,'(A32,1X,F6.1)') 'Threshold for merging = '
  ,MAXSD*LUMPFF
ENDIF
MINI (ROW) = PREI
MINJ (ROW) = PREJ
IF (SDIST(MINI(0),MINJ(0)) .GT. MAXSD*LUMPFF) GOTO 390
IF (ROW .GE. 2*MAXLMP) GOTO 350
ROW = ROW + 1
GOTO 380
350 CONTINUE
MINSD = SDIST(MINI(0),MINJ(0))
C Write out the MAXLMP smallest distances:
WRITE(CMES,'(14,1X,A21)') MAXLMP,' Smallest Distances:
CALL SDEST(CMES,0)
CALL FDST(LOGFIL,CMES)
WRITE(CMES,'(A25)') 'Class1 Class2 Distance'
CALL SDEST(CMES,0)
CALL FDST(LOGFIL,CMES)
DO 435 II= 0, MAXLMP-1, 1
WRITE(CMES,'(14,4X,I4,4X,F6.1)') MINI(II), MINJ(II),
  SDIST(MINI(II),MINJ(II))
435 CONTINUE
C Do the lumping, replacing two classes with an average:
DO 440 II= 0, MAXLMP-1, 1
C IF (SDIST(MINI(II),MINJ(II)) .LE. MAXSD*LUMPFF) THEN
C Change the means of the lumped cluster:
WRITE(CMES,'(A9,13,1X,A1,1X,I3)') 'Merging: '
  ,MINI(II), '+', MINJ(II)
CALL SDEST(CMES,0)
CALL FDST(LOGFIL,CMES)
DO 420 BAND=0, NBANDS-1, 1
  SMAXJ(K) = STMAXJ(K)
  STMAXJ(K) = STMAXJ(K+1)
  DBARJ(K) = DBARJ(K+1)
DO 425 BAND=0, NBANDS-1, 1
  SMAXJ(K) = STMAXJ(K+1)
  STMAXJ(K) = STMAXJ(K+1)
  DBARJ(K) = DBARJ(K+1)
420 CONTINUE
C Wipe out the old second cluster:
SCOUNT(MINI(II)) = SCOUNT(MINI(II)) + SCOUNT(MINJ(II))
DO 430 K= MINJ(II)+1, CLASES-1, 1
  SCOUNT(K) = SCOUNT(K+1)
  SMAXJ(K) = STMAXJ(K)
  STMAXJ(K) = STMAXJ(K+1)
  DBARJ(K) = DBARJ(K+1)
DO 425 BAND=0, NBANDS-1, 1
  SMAXJ(K) = STMAXJ(K+1)
  STMAXJ(K) = STMAXJ(K+1)
  DBARJ(K) = DBARJ(K+1)
425 CONTINUE
430 CONTINUE
CLASES = CLASES - 1
ENDIF
440 CONTINUE
390 CONTINUE
IF (OLDCLS .GT. CLASES) THEN
WRITE(CMES,'(A20,14,1X,I4)') 'Merging total: ', OLDCLS, CLASES
CALL SDEST(CMES,0)
CALL FDST(LOGFIL,CMES)
ENDIF
ENDIF
RETURN
END
/* ATOERD.C:  Converts McIDAS AREA files to ERDAS format. 
   Handles files from PC-McIDAS downloads and 
   McIDAS-AIX transfers. 
   Creates ERDAS .LAN and .GIS format files, 
   and will convert the .ET enhancement file 
   to a .TRL ERDAS trailer file if the output 
   file is specified as .GIS. 

A list of the important variables used: 

infile      Names for the image input files. 
nbands      Number of input areas, each a 
one band McIDAS area, to combine 
            into an ERDAS image file. 
outfile     Output ERDAS format image file 
            with nbands bands. 
trlfile     Output ERDAS .TRL trailer file. 
            (for making ERDAS .GIS files.) 
etfile      Input McIDAS enhancement file. 

Copyright Gerard Peltzer, 1992. 
Last modified 3/5/1992 */

#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

#define FILECHARS 70   /* Max number of chars in filename. */
#define INPUTWIDTH 2   /* Number of chars in inputs. */
#define MAXROWCOL 4000 /* Maximum # of rows or columns. */
#define MAXBANDS 7     /* Most bands to put in Erdas file */
#define SUCCESS -1 
#define FAILURE 0 
#define YES 1 
#define NO 0 
#define GIS 3 
#define LAN 4 

typedef unsigned char byte;

main() 
{
  byte buf[MAXROWCOL], null[2000];
  long temp, head, head2, header[2000], *longspot;
  char infile[MAXBANDS][FILECHARS], outfile[FILECHARS];
  char lbuf[80], c;
  int i, j, dx, row, col, length, reversal, *intspot;
  FILE *fil[MAXBANDS], *fpout, *fpstuf;
  char *stuf = "header.txt";
  double dcol;

  /* Enhancement table (GIS) variables: */
  FILE *etfil, *trlfil;
  char etfile[FILECHARS], trlfile[FILECHARS], *etptr, *trlptr;
  byte erhd[128];
  long xstart, ystart;
  int maptyp, lautyp, nclass, clases;

  /* Subroutines used: */
  int Reverse4(long *word);
  int Reverse2(int *word);
  int maketrlfile(char etfile[FILECHARS], char trlfile[FILECHARS]);

  /* Image specifications used: */
  /* Type long is really a long int (*4); */
  long ncols;       /* The number of columns in the input image */
  long nrows;       /* The number of rows in the input image */
# atoe.rd.c

```c
int imagetype; /* Holds image type of input (GIS or LAN) */
int packtype; /* The pack type of input image (4, 8, 16 bit) */
int nbands; /* The number of bands in the input image */
long prefix; /* The line prefix before data */
int abands;
int satellite;
int packing;

textbackground(1);
textcolor(6);
c1rscreen();
window(1,1,80,15);
gotoxy(10,1);
printf("McIDAS (AREAxxx) to ERDAS (.LAN or .GIS) file conversion\n");
printf("For .GIS output, color table .ET file converted to .TRL.\n");
window(1,17,80,25);
gotoxy(5,19);
c1rscreen();
printf("Enter the number of image files (n <= 7) to put\n");
printf("into a single ERDAS format image file with n bands: ");
window(1,1,80,15);
gotoxy(1,6);
c1rel();
printf("Number of areas (bands): ");
c1rel();
gets(lnbuf);
sscanf(lnbuf, "%d", &nbands);
if(nbands < 1 || nbands > MAXBANDS)
{
    printf("\nThis program uses 1 to %d bands! ",& MAXBANDS);
    printf("\nTaking default of 1 band. ");
nbands = 1;
}
for(i=0; i<nbands; i++)
{
    strcpy(infile[i], "\0");
    while((window[i]=fopen(infile[i], "rb")) == NULL)
    {
        window(1,17,80,25);
gotoxy(5,19);
c1rscreen();
printf("File should be a single band McIDAS area\n");
printf("This will become band %d in the ERDAS file\n", i+1);
window(1,1,80,15);
gotoxy(1,7);
c1rel();
printf("File should be a single band McIDAS area\n");
printf("This will become band %d in the ERDAS file\n", i+1);
window(1,1,80,15);
gotoxy(1,6);
c1rel();
printf("Enter File Name for Channel %d: ",& i+1);
gets(infile[i]);
}
/* Read the header for each file, get parameters: */
reversal = NO;
head = 768;
fread(header+i,1,head,fil[i]);
/* Rows and columns */
 nrows = header[9];
 ncols = header[10];
 head = header[34];
c1rel();
if(nrows <= 1 || ncols <= 1 || (head < 256 || head > 1999))
{
    printf("\n Header is not standard information...\n");
    printf("\n Attempting a reversed byte order. ");
    Reverse4(&header[4]);
    Reverse4(&header[9]);
    Reverse4(&header[10]);
    Reverse4(&header[34]);
    Reverse4(&header[35]);
    Reverse4(&header[3]);
    Reverse4(&header[11]);
```
Reverse4(&header[14]);
Reverse4(&header[63]);
Reverse4(&header[49]);
Reverse4(&header[50]);
Reverse4(&header[51]);
reversal = YES;
}
nrows = header[9];
ncols = header[10];
head = header[34];
prefix = header[49]+header[50]+header[51];
printf("\nLine Prefix = %ld ",prefix);
if (prefix < 0 || prefix > 200)
  printf("\nNo read of prefix size (%ld) - assuming 0."
 ,prefix);
  prefix = 0;
if (head < 256 || head > 1999)
  printf("\nNo read of header size (%ld) - assuming 768 bytes."
 ,head);
  head = 768;
else
  printf("\nHeader size = %ld ", head);
printf("\nEnter header size if different: ");
c1reol();
gets(lnbuf);
sscanf(lnbuf,"%ld",&head2);
if (head2 <= 0 || head2 > 1999)
{  head2 = head; }
else
{  head = head2;
    printf("\nAssumed header size: %ld",head);
}
printf("\nArea Date Read: %ld",header[4]);
printf("\nArea rows (lines): %ld  Area columns (elements): %ld"
 ,nrows,ncols);
/* only allow MAXROWCOL rows or cols */
if (nrows > MAXROWCOL || ncols > MAXROWCOL)
{  printf("\nToo many rows or columns to handle!\n");
    printf("\nnrows: %ld ncols: %ld\n",nrows,ncols);
    exit(1);
}
if (nrows <= 1 || ncols <= 1)
{  printf("\nProblem reading header rows or columns!\n");
    printf("\nnrows: %ld ncols: %ld\n",nrows,ncols);
    exit(1);
}
satellite = header[3];
packing = header[11];
abands = header[14];
c1reol();
printf("\nHeader size: %ld",head);
printf("\nSatellite: %d, Bytes per pixel: %d, bands in area: %d"
 ,satellite,packing,abands);

/* Option to print out header information to file 'header' */
/* fptstuf=fopen(stuf,"w");
printf(fptstuf,"Word: Header(int) Header(long)\n");
for(j=1;j<head;j++)
{  fprintf(fptstuf,"%d, %d \ld\n",j,header[j],header[j]);
  }
*/

/* Open output image file: */
window(1,17,80,25);
gotoxy(5,19);
clsrcl();
printf("Enter the new output file here\n");
printf("Data will be saved to this file\n");
printf("A .lan or .gis extension is recommended.\n");
window(1,1,80,15);
gotoxy(1,6);
clear();
printf("Output File name: ");
guts(outfile);
while((fpout=fopen(outfile,"wb")) == NULL)
{
    window(1,17,80,25);
gotoxy(5,19);
clsrcl();
printf("Error opening output ERDAS file \n");
printf("Please try again: \n");
window(1,1,80,15);
gotoxy(1,6);
clear();
printf("Enter Output File Name: ");
guts(outfile);
}
window(1,1,80,25);
clsrcl();
printf("succeeded opening %s\n",outfile);
if (packing == 1 && packing == 2)
{
    printf("nData must be one or 2 bytes in size!\n");
    printf("n Data packing = %d ",packing);
    exit(1);
}

/* Find if GIS or LAN file. If GIS, search for trailer: */
strcpy(trfile,outfile);
strdup(trlfile);
if ((trlptr=strstr(trlfile,".GIS")) != NULL)
{
    classes = 0;
imagetype = GIS;
    if ((trlptr=strstr(infile,"REA")) != NULL)
    {
        strcpy(etfile,trlptr+3);
    }
else
    {
        strcpy(etfile,infile[0]);
    }
strup(trfile);
strcat(trfile,".ET");
printf("Searching for enhancement file: %s\n",etfile);
if (etfil=fopen(etfile,"rb")) == NULL)
{
    printf("ERDAS enhancement file %s not found. \n",etfile);
    printf("Enter enhancement table filename, if any: ");
clear();
guts(etfile);
printf("\n");
if (etfil=fopen(etfile,"rb")) == NULL)
    printf("No enhancement table used \n");
}
if(etfil != NULL)
{
    fclose(etfil);
    trlptr=strstr(trlfile,".GIS");
    sprintf(trlptr,".trl\0n");
    if (maketrfile(etfile,trlfile) == FAILURE )
        printf("Error making .TRL enhancement file! \n");
else
    printf("Created .TRL enhancement file %s \n",trlfile);
}
else
    imagetype = LAN;

/* Set up ERDAS LAN and GIS file specs: */
ERDAS uses 128 byte header at the beginning of LAN and GIS files. */
if (packing == 1) packtype = 0;
if (packing == 2) packtype = 2;
xstart = 0;
ystart = 0;
maptyp = 0;
ialtyp = 0;
if (imagetype == GIS) {nclass = classes;
if (imagetype == LAN) {nclass = 0;
for (i=0; i<128; i++){
   printf("erhd[1] = 0; ");
}rength[1] = 0;
}
printf("HEAD74");
intspot = (int*)&erhd[6];
*intspot = packtype;
intspot = (int*)&erhd[8];
*intspot = nbands;
longspot = (long*)&erhd[16];
*longspot = ncols;
longspot = (long*)&erhd[20];
*longspot = nrows;
longspot = (long*)&erhd[24];
*longspot = xstart;
longspot = (long*)&erhd[28];
*longspot = ystart;
intspot = (int*)&erhd[88];
*intspot = maptyp;
intspot = (int*)&erhd[90];
*intspot = nclass;
intspot = (int*)&erhd[107];
*intspot = iatyp;

/*
 * Loop over rows and columns, convert input to output:
 */
printf("Header data for file %s:\n", outfile);
printf("\n nrows: %ld ncols %ld \n", nrows, ncols);
fwrite(fpout, 01, SEEK_SET);
fwrite(erhd, 1, 128, fpout);
fwrite(fpout, 1281, SEEK_SET);
for (i=0; i<nbands; i++)
{
   fseek(fil[i], 01, SEEK_SET);
   length = fread(null, 1, head, fil[i]);
   gotoxy(1, 22);
clear();
   printf("Processing row: ");
for (row=0; row < nrows; row++)
{
   for (i=0; i<nbands; i++)
   {
      if (prefix > 0)
         { length = fread(buf, packing, prefix, fil[i]);
         }
      length = fread(buf, packing, ncols, fil[i]);
      j = 10*(row/10);
dx = row - j;
      if (row > 0 && dx == 0)
         { gotoxy(17, 22);
clear();
printf("%d", row);
      }
      if (length != ncols)
         { printf("\nA read error at row: %d", row);
            printf(" ESC to exit, any key to continue");
            if ((c=getc(stdin)) == 27) exit(1);
         }
      if (packing == 2)
      {
         for (col=0; col<nrows; col++)
            { dcol = col;
               if (reversal == YES && fmod(dcol, 2.0) == 0)
               {
int Reverse4(long *word)
{
    typedef unsigned char byte;
    byte *point1, *pointr, temp;
    /* Reverse the AIX 4 byte words: */
    point1 = word;
    pointr = point1+3;
    temp = *point1;
    *point1 = *pointr;
    *pointr = temp;
    point1++;
    pointr--;
    temp = *point1;
    *point1 = *pointr;
    *pointr = temp;
    return SUCCESS;
}

int Reverse2(int *word)
{
    typedef unsigned char byte;
    byte *point1, *pointr, temp;
    /* Reverse the AIX 4 byte words: */
    point1 = word;
    pointr = point1+1;
    temp = *point1;
    *point1 = *pointr;
    *pointr = temp;
    return SUCCESS;
}

/****************************************************************************
/*
* Function to save enhancement tables with means of sets from
* a McIDAS format .ET file in an ERDAS .TRL format trailer file.
* This file is used to define colors in ERDAS .GIS image classification files.
* Note that a stretch factor of 2 has been used throughout this
* program because of the 7 bit nature of the McIDAS color display.
* For use on IBM PC, compiled with Borland C++. */
/****************************************************************************
int maketrfile(char etfile[FILECHARS], char trlfile[FILECHARS])
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

#define MXBNDS 5
#define MXCLAS 256
#define STRETCH 2

{  
  byte output[128];
  FILE *etfil, *trfil;
  int BAND, SET, SIZE, NROWS, NCOLS, CLASSES;
  int i, j, reverse;
  unsigned long longi, MNRED[MXCLAS], MNGRE[MXCLAS], MNBLU[MXCLAS];
  byte BMRED[MXCLAS], BMGRE[MXCLAS], BMBLU[MXCLAS];

  /* Subroutines used: */
  int Reverse4(long *word);
  int Reverse2(int *word);

  /* Open the McIDAS format file, read information: */
  while ((etfil = fopen(etfile, "rb")) == NULL)
    { printf("\n Error opening .ET file %s ", etfile);
      return FAILURE;
    }

  /* Read in color look-up tables: */
  fseek(etfil, 41, SEEK_SET);
  fread(MNRED, 4, CLASSES*STRETCH, etfil);
  fseek(etfil, 41+4*2561, SEEK_SET);
  fread(MNGRE, 4, CLASSES*STRETCH, etfil);
  fseek(etfil, 41+4*5121, SEEK_SET);
  fread(MNBLU, 4, CLASSES*STRETCH, etfil);

  /* Switch bytes, if bad values read in: */
  reverse = NO;
  for(i = 0; i < CLASSES*STRETCH; i++)
    { if(MNRED[i] > 255 | | MNRED[i] < 0)
      { reverse = YES; }
    }  
  if (reverse == YES)
    { 
      for(i = 0; i < CLASSES*STRETCH; i++)
        { Reverse4(&MNRED[i]);
          Reverse4(&MNGRE[i]);
          Reverse4(&MNBLU[i]);
        }
    }

  /* Open the Erdas format file, write information: */
  while ((trfil = fopen(trfile, "wb")) == NULL)
    { printf("\n Error opening .TRL output file %s ", trfile);
      return FAILURE;
    }

  /* Clear output buffer, Write header: */
  for (i = 0; i < 128; i++) { output[i] = 0; }
  fseek(trfil, 0L, SEEK_SET);
  for (i = 0; i < 17; i++)
    { fwrite(output, i, 128, trfil); }

  printf(output,"TRAIL74");
  printf(output+73,"McIDAS ISODATA IMAGE CLASSIFICATION~");
fseek(trlfil,0L,0L);
fwrite(output,1,128,trlfil);

/* Write out class names, which are numbers: */
/*
 for (i = 0; i< STRETCH*CLASES; i+=4)
  {
    for (j = 0; j< 128; j++) { sprintf(output[j]," "); }
    for (j = 0; j< 3; j++)
    {
      if (fmod((i+j),2) == 0)
        sprintf(output+32+j, "%3d", (i+j)/2);
      else
        sprintf(output+32+j, "-");
    }
    fseek(trlfil, (128*(17+i/4)), SEEK_SET);
    fwrite(output,1,128,trlfil);
  }
*/

/* Write out class means, integers changed to bytes: */
for (i = 0; i< MXCLAS; i++)
{
    BMRED[i] = MNRED[i];
    BMGRE[i] = MNGRE[i];
    BMBLU[i] = MNBLU[i];
}

fseek(trlfil,128L,SEEK_SET);
fwrite(BMGRE,1,CLASES*STRETCH,trlfil);
fseek(trlfil,384L,SEEK_SET);
fwrite(BMRED,1,CLASES*STRETCH,trlfil);
fseek(trlfil,6401L,SEEK_SET);
fwrite(BMBLU,1,CLASES*STRETCH,trlfil);

fclose(trlfil);
fclose(etrfil);
return SUCCESS;
}
ERDAS format is band-sequential:
    Pixels 1 through x of line 1, band 1
    Pixels 1 through x of line 1, band 2 ...
    Pixels 1 through x of line 1, band n
    Pixels 1 through x of line 2, band 1 ...

McIDAS format is band-interleaved:
    Element1: band1,band2,band3 Element2: band1,band2,band3 ...

This program converts an n band ERDAS file to an n band McIDAS area.

If the ERDAS file is a .GIS classification, then the corresponding .TRL file will be converted to a McIDAS-X .ET color look-up table file. To use the .ET file on PC McIDAS, a conversion program must be used to switch the byte positions in each 4-byte word.

The areas and .ET files created by this program are ready to be sent to McIDAS-X machines via binary FTP.

Copyright Gerard Peltzer, 1992.
Last modified 5/28/1992

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
define INPUTWIDTH 2     /* Number of chars in inputs. */
define MAXROWCOL 3000   /* Maximum # of rows or columns. */
define MAXBANDS 7       /* Most bands from Erdas file */
define SUCCESS -1
#define FAILURE 0
#define FILECHARS 70       /* Max number of chars in filename. */
define INPUTWIDTH 2     /* Number of chars in inputs. */
define YES 1
define NO 0
define GIS 3
define LAN 4
define MAXELINT MAXROWCOL*MAXBANDS

typedef unsigned char byte;

void main(int argc, char *argv[])
{
    byte outbuf[21000];
    byte inbuf[MAXBANDS][MAXROWCOL];
    byte prefix[40];
    byte erdhed[128];
    long header[2000];
    byte color;
    char infil[FILECHARS], outfile[FILECHARS];
    char lnbuf[80], *c;
    FILE *infil, *outfil;
    byte null[2000];
    long temp, head, head2, *longspot;
    int i, j, band, dx, row, col,length, reversal, *intspot;
    int headread, stretch;
    double dcol;

    /* Enhancement table (GIS) variables: */
    FILE *etfil, *trlfil;
    char etfile[FILECHARS], trlfile[FILECHARS], *etptr, *trlptr;
    long xstart, ystart;
int maTyp, iaTyp, clases;

/* Image specifications used: */
/* Type long is really a long int (4); */
    long nCols;  /* The number of columns in the input image */
    long nRows;  /* The number of rows in the input image */
    int imgType; /* Holds image type of input (GIS or LAN) */
    int packType; /* The pack type of input image (4,8,16 bit) */
    int nBands; /* The number of bands in the input image */
    int nClasses; /* The number of classes (GIS) */
    int abands;
    int satellite;
    int packing;

/* Subroutines used: */
    int Reverse4(long *word);
    int makeetfile(char etfile[FILECHARS], char trlfile[FILECHARS]);
    int Reverse2(int *word);

    textcolor(6);
    clrscr();
    window(1,1,80,15);
    gotoxy(10,2);
    printf("ERDAS .LAN to McIDAS AREA file conversion utility\n");

    window(1,17,80,25);
    gotoxy(5,17);
    clrscr();
    printf("Enter the ERDAS file to convert into a McIDAS area: ");
    window(1,1,80,15);
    gotoxy(1,6);
    clrleol();
    printf("Enter ERDAS File Name: ");
    gets(infile);
    while((infile=fopen(infile,"rb")) == NULL)
    {
        clrscr();
        printf("File should be an ERDAS .LAN or .GIS file \n");
        clrleol();
        printf("Enter Erdas input file name: ");
        gets(infile);
    }

    /* Stretch for McIDAS if file is a classification: */
    strupr(infile);
    if (!strstr(infile,".GIS")) != NULL)
    { stretch = 2; }
    else
    { stretch = 1; }

    fseek(infile,01,SEEK_SET);
    length=fread(erdhed,128,1,infile);
    headread = 0;
    printf("Header data for file %s:\n",infile);
    while (headread != SUCCESS)
    {
        int *spot = (int *)&erdhed[6];
        packType = *intspot;
        int *spot = (int *)&erdhed[8];
        nBands = *intspot;
        longspot = (int *)&erdhed[16];
        nCols = *longspot;
        longspot = (int *)&erdhed[20];
        nRows = *longspot;
        int *spot = (int *)&erdhed[90];
        nClasses = *intspot;
        printf("imgType:%d packType:%d nbands:%d", imgType, packType, nBands);
printf("\n nrows: %ld ncols %ld \n", nrows, ncols);
/* only allow MAXROWCOL rows or cols */
if (nrows > MAXROWCOL || ncols > MAXROWCOL || nrows<=0 || ncols<=0
    || nbands < 1 || nbands > MAXBANDS)
{
    if (headread == 0) /* call reverse for PC to WS convert */
    {
        printf("Now Attempting reversed byte order: \n");
        Reverse4(&erdhed[16]);
        Reverse4(&erdhed[20]);
        Reverse2(&erdhed[90]);
        Reverse2(&erdhed[6]);
        Reverse2(&erdhed[8]);
        headread++;
    }
    else
    {
        printf("\n Invalid parameters in data, exiting! \n");
        exit(1);
    }
}
else if (headread!=SUCCESS)
{
    printf("succeeded opening %s\n", infile);
    window(1,17,80,25);
    gotoxy(5,23);
    clrscr();
    printf("Enter the output McIDAS area, convention is AREAxxxx ");
    window(1,1,80,17);
    gotoxy(1,14);
    clrscr();
    strcpy(outfile,"\0");
    printf("Enter McIDAS area file Name: ");
    gets(outfile);
    while((outfile=fopen(outfile,"wb")) == NULL)
    {
        window(1,17,80,25);
        gotoxy(5,23);
        clrscr();
        printf("File should be a new single band McIDAS area\n");
        window(1,1,80,15);
        gotoxy(1,14);
        clrscr();
        printf("Enter McIDAS area file Name: ");
        gets(outfile);
    }
    /* If GIS file, search for trailer: */
    strcpy(trlfile,infile);
   strupr(trlfile);
   strupr(outfile);
    if (((trlptr=strstr(trlfile,".GIS")) != NULL)
    {
        strcpy(trlptr,".TRL");
        if (((trlptr=strstr(outfile,"AREA")) != NULL)
        {
            strcat(etfile,trlptr+4);
            strcat(etfile,".ET");
           strupr(etfile);
        }
    }
    else
    { strcpy(etfile,outfile);
       strupr(etfile);
        if ((trlptr=strstr(etfile,".")) !=NULL) strcat(etfile,".ET");
        else strcat(etfile,".ET");
    }
    printf("Searching for ERDAS .TRL file: %s\n",trlfile);
    if((trlfil=fopen(trlfile,"rb")) == NULL)
    {
        printf("ERDAS enhancement file ->%s< not found. \n",trlfile);
        printf("Enter trailer table filename, if any: ");
        clrscr();
        gets(trlfile);
        printf("\n");
if((trfil=fopen(trfil,"rb")) == NULL)
printf("No color table used \n");
if((etfil=fopen(etfil,"wb")) == NULL)
printf("No color table used \n");
}
if(trfil != NULL && etfil != NULL)
{
fclose(trfil);
fclose(etfil);
if (makeetfile(etfile, etfil) == FAILURE )
printf("Error making .ET enhancement file! \n");
else
printf("Created .ET enhancement file %s \n", etfile);
}

/* Set the header for McIDAS file, set parameters: */
head = 768/sizeof(long);
for (i=0; i<head; i++) header[i] = 0;
header[2] = 4;
header[3] = 3;
header[4] = 92000;
header[8] = 1;
header[9] = nrows;
header[10] = ncols;
header[3] = 0; /* satellite */
header[11] = 1; /* packing, in bytes */
header[12] = 1; /* line resolution */
header[13] = 1; /* element resolution */
header[14] = nbands; /* # of bands */
header[17] = 92000;

/* Filter map-prefix (bit-for-band, right most bit is band 1) */
header[19] = 1;
if (nbands > 1)
{
 for (i=2; i< nbands; i++)
  {
   dcol = i-1;
   header[19] = header[19] + pow(2, dcol);
  }
}

header[34] = 768; /* area data offset */
header[35] = 256;
header[48] = 1; /* starting scan */
if (nbands > 1)
{
 header[51] = 8;
 header[55] = 8;
 /* LBV section of prefix */
 header[52] = 1381189974;
 header[53] = 1414091330;
 header[55] = 1;
 header[56] = 2;

/* Debugging printouts: */
printf("viel= %d, %d, %d, %d, %ld\n",
header[52], header[52],
header[53], header[53]);
  for (i=1; i< 65; i++) printf(" header[%d] = %ld, i, header[i]);

/* Line prefix setup */
for (i=1; i< 9; i++)
  {
  if (i <= nbands) prefix[i-1]=i;
  else prefix[i-1]=0;
  }

/* Loop over rows and columns, convert input to output: */
seek(outfil, 01, SEEK_SET);
wrote(header+1, sizeof(long), head, outfil);
seek(infil, 1281, SEEK_SET);
gotoxy(1, 22);
crelol();
printf("Processing row: ");
for(row=0; row < nrows; row++)
{ for(band=0; band<nbands; band++) fread(inbuf[band],1,ncols,infil);
    j = 10*(row/10);
    dx = row - j;
    if (row > 0 && dx == 0) {
    gotoxy(18,22);
    clreol();
    printf(" %d",row);
    }
    col = 0;
    j = 0;
    while(col < ncols)
    {
        for(band=0; band < nbands; band++)
        {
            outbuf[j] = inbuf[band][col]*stretch;
            j ++;
        }
        col++;
    }

    /* Write out data to outimg, ncolsnbands: */
    if (nbands >1) {length = fwrite(prefix,1,8,outfil);}
    length = fwrite(outbuf,1,ncols*nbands,outfil);
    }
fclose(infil);
fclose(outfil);
gotoxy(5,24);
clreol();
printf("\nFinished...Have a nice day. \n");
exit(0);


/* Reverse the AIX 4 byte words: */
int Reverse4(long *word)
{
    typedef unsigned char byte;
    byte *point1, *pointr, temp;
    point1 = (byte *)word;
    pointr = (byte *)point1+3;
    temp = *point1;
    *point1 = *pointr;
    *pointr = temp;
    point1++;
    pointr--;
    temp = *point1;
    *point1 = *pointr;
    *pointr = temp;
    return SUCCESS;
}

int Reverse2(int *word)
{
    typedef unsigned char byte;
    byte *point1, *pointr, temp;

    /* Reverse the AIX 4 byte words: */
    point1 = word;
    pointr = point1+1;
    temp = *point1;
    *point1 = *pointr;
    *pointr = temp;
    return SUCCESS;
}

/--------------------------------------------*/
/*
Function to save enhancement tables with means of sets from
an ERDAS .TRL format trailer file.
This file is used to define colors in ERDAS .GIS classification files.

Note that a stretch factor of 2 has been used throughout this program because of the 7 bit nature of the McIDAS color display.

Written by Gerard Feitiner
For use on IBM PC, compiled with Borland C++.

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define MXBND5 5
#define MXCLAS 256
#define STRETCH 2

{ FILE *etfil, *trlfil;
  int BAND, SET, SIZE, NROWS, NCOLS, CLASSES;
  int i, j, reverse;
  unsigned long longi, MNRED[MXCLAS], MNGRE[MXCLAS], MNBLU[MXCLAS];
  byte BMRED[MXCLAS], BMGRE[MXCLAS], BMBLU[MXCLAS];

  /* Subroutines used:
       int Reverse4(long *word);
       int Reverse2(int *word);
  */

  /* Open the Erdas format file, read information: */
  while ((trlfil = fopen(trlfil, "rb")). close
  { printf("\n Error opening .TRL input file %s ", trlfil);
    return FAILURE;
  }

  /* read in class means, integers changed to bytes: */
  fseek(trlfil, 1281, SEEK_SET);
  fread(BMRED, 1, MXCLAS, trlfil);
  fseek(trlfil, 3841, SEEK_SET);
  fread(BMRED, 1, MXCLAS, trlfil);
  fseek(trlfil, 6401, SEEK_SET);
  fread(BMRED, 1, MXCLAS, trlfil);
  for (i = 0; i < MXCLAS/STRETCH; i++)
  {
    j = i * STRETCH;
    MNRED[j] = BMRED[i];
    MNGRE[j] = BMRED[i];
    MNBLU[j] = BMRED[i];
    MNRED[j+1] = BMRED[i];
    MNGRE[j+1] = BMRED[i];
    MNBLU[j+1] = BMRED[i];
  }

  /* Debugging printouts:
       printf("LUT classes: \%d", MXCLAS);
       for(i=0;i<MXCLAS;i++)
         printf("r: \%d g: \%d b: \%d \n", MNRED[i], MNGRE[i], MNBLU[i]); */

  reverse = YES;
  if (reverse == YES)
  {
    for(i=0;i<MXCLAS;i++)
    { Reverse4(&MNRED[i]);
      Reverse4(&MNGRE[i]);
      Reverse4(&MNBLU[i]);
    }
ere the McIDAS format file: */
   while ((etfil = fopen(etfile,"wb")) == NULL)
      printf("\n Error opening .ET file \%s ",etfile);
   return FAILURE;
}

/* Write out color look-up tables: */
   fwrite(MNRED,4,MXCLAS,etfil);
   fwrite(MNGRE,4,MXCLAS,etfil);
   fwrite(MNBLU,4,MXCLAS,etfil);
fclose(trlfil);
fclose(etfil);
return SUCCESS;
SBDSIG.C

This program converts ERDAS .SBD files to McIDAS .SIG files or vice-versa, depending on input.

The program takes input parameters through a user friendly menu as well as through a command line of arguments.

A list of the important variables used:
infile  Name for the .SIG or .SBD input file.
outfile Name for the output file.

Copyright Gerard Peltzer, 1992.
Last modified: 4/29/1992

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#define INPUTWIDTH 80 /* Number of chars in inputs. */
#define MXCLAS 255
#define MAXSBD 16
#define MXBNDSD 7
#define SUCCESS 1
#define FAILURE 0
#define YES 1
#define NO 0
#define SBD 8
#define SIG 9
#define HISTBYTES 512*MXBNDSD

typedef struct Sig{
    char signame[32];
    long scount;
    char blank[92];
    int minvector[MAXSBD];
    int maxvector[MAXSBD];
    float meanvector[MAXSBD];
    float covar[32];
    /*char histo[HISTBYTES];*/
} Signature;

typedef struct Sighead{
    char lanname[100];
    int classes;
    int bands;
    char blank[24];
} Sbdheader;

int main(int argc, char *argv[])
{
    /* Declarations of the functions used in this program: */
    Signature *class;
    Sbdheader head;
    void commandline(int argc, char *argv[], char *infile,
    char *outfile, int *type);
    int readsig(char *sigfile, Sbdheader *head, Signature *class);
    int writesig(char *sigfile, Sbdheader *head, Signature *class);
    int readsbd(char *sbdfile, Sbdheader *head, Signature *class);
    int writesbd(char *sbdfile, Sbdheader *head, Signature *class);
    int type, mean, lclass, hclass;
/* Variable declarations: */
char sigfile[INPUTWIDTH], sbdfile[INPUTWIDTH];
char infile[INPUTWIDTH], outfile[INPUTWIDTH];

/* Allocate memory for class: */
if ((class = (Signature *) malloc(MXCLAS* sizeof(Signature))) == NULL)
  { printf("\nNot enough memory to allocate classes!\n");
    return(1); }

/* First take care of the input parameters: */
commandline( argc, argv, infile, outfile, &type);

/* Read in data from infile: */
if (type == SIG)
  {
    strcpy(sigfile, infile);
    strcpy(sbdfile, outfile);
    if ( readsig(sigfile, &head, class) == FAILURE )
      { printf("Quitting due to sig read errors... \n");
        return(1); }
    if ( writesbd(sbdfile, &head, class) == FAILURE )
      { printf("Quitting due to sbd write errors... \n");
        return(1); }
  }

if (type == SBD)
  {
    strcpy(sbdfile, infile);
    strcpy(sigfile, outfile);
    if ( readsbd(sbdfile, &head, class) == FAILURE )
      { printf("Quitting due to sbd read errors... \n");
        return(1); }
    if ( writesig(sigfile, &head, class) == FAILURE )
      { printf("Quitting due to sig write errors... \n");
        return(1); }
  }

printf("\n...Ending... \n");
free(class);
return(0);

/*-----------------------------------------------*/

Function Commandline
This routine handles the command line inputs.

A note on command line parameters:
The command line parameters are passed in when the user calls up the
program. Thus, the 1st parameter is the name of the program name.
The variables for command line parameters are:
  argc The # of command line parameters read.
  argv[argc] The char array of argc parameters

/*-----------------------------------------------*/
void commandline(int argc, char *argv[], char *infile,
                 char *outfile, int *type)
{
  char c[INPUTWIDTH], c2[INPUTWIDTH], namfile[INPUTWIDTH];
  char *trplptr;
  int check, j, band;
  FILE *fpin, *fput;

  /* If the user types in a ? after the program name, give him (or her)
a list of the command line parameters:
   */
  if(((argc > 1) || (*argv[1] == '?') || argc > 2)
```c
printf("\nUse these easy command line parameters: \n");
printf("<sbdsg> <Infile> <Outfile> ");
exit(0);
}

/* Copy the command line parameters that were given into their respective variables. Also clear any variables that were not given as command line parameters: */
if(argc >= 2) {strcpy(infile,argv[1]);}
else {strcpy(infile,":0");}

/* In this case, the user gave all the parameters initially. */
if(argc >= 3) {strcpy(outfile,argv[2]);}
else {strcpy(outfile,":0");}

if(argc > 3) { printf("Maximum 3 input parameters!");
printf(" Taking the first 3"); }

clrscr();
textbackground(0);
textcolor(6);
window(1,1,80,18);
gotoxy(10,2); /*
printf("\nSignature File Transform\n----------------------------------\n");

while((fpin=fopen(infile,"r")) == NULL)
{ /*
window(1,20,80,25);
gotoxy(5,20); /*
printf("\nEnter the input file here");
printf("\n(Data should be .SIG or .SBD file)");
/*
window(1,1,80,18);
gotoxy(1,3);
cleol(); /*
printf("\nData File name: ");
gets(infile);
}
fclose(fpin);

/* Find whether .SBD or .SIG file: */
strupr(infile);
strcpy(namfile,infile);
if ((trlptr=strstr(namfile,".SBD")) != NULL)
{ /*type = SBD; */
else
{ if ((trlptr=strstr(namfile,".SIG")) != NULL)
{ /*type = SIG; */
else
{ printf("\nNamefile type not specified - assuming SIG");
 /*type = SIG; */

 /* Check output file is viable: */
strcpy(namfile,"\0");
while((fpout=fopen(outfile,"w")) == NULL)
{ strcpy(namfile,infile);
if ((trlptr=strstr(namfile,".")) != NULL)
{ if (*type==SIG) strcpy(trlptr,".SBD");
if (*type==SBD) strcpy(trlptr,".SIG");
 /*
window(1,20,80,25);
gotoxy(5,20); /*
if (*type==SIG) printf("\nEnter the output .SBD file here");
if (*type==SBD) printf("\nEnter the output .SIG file here");
printf("\n(Default name is %s ",namfile);
/*
window(1,1,80,18);
gotoxy(1,3);
cleol(); /*
printf("\nOutput File name: ");
ggets(outfile);
```
if (strncpy(outfile, "\0") == 0) strcpy(outfile, namfile);
}
/*
  window(1,20,80,25);
crsr();
window(1,1,80,18);
crsr();
gotoxy(1,10); */
fclose(fpout);
return;

*******************************************************************************

Function Readsig

This routine reads data from McIDAS .SIG file.
The arrays are all indexed from 0 to n-1.

Read in Data:
1) name of sig file
2) number of classes
3) number of bands
4) blank (says "--Max data values--")
5) max-B1 max-B2 ... max-BN
6) blank (says "--BAND MEANS--")
7) blank (says "Class Counts " then bands)
8) class# counts mean-B1 mean-B2 ... mean-BN
   C+1 if C of these...
   8+C1) blank (says "Class covariance matrices")
   8+C2) class#
   8+C3) COVAR(0,0) COVAR(0,1) ... COVAR(0,N-1)
   N of these...
   C of these...
*******************************************************************************

int readsig(char *infile, Sbdheader *head, Signature class[2])
#define COV 32*MXCLAS
{
  int i,j,k;
  int clas, band, classes, nbands, eof;
  int max[MAXSBD], min[MAXSBD], smean[MAXSBD][MXCLAS];
  long count[MXCLAS];
  float fcount, fmean, cova[32];
  /*
  float *covarm;
  FILE *fpin;
  char *trilptr, nameline[32];
  char dataline[80];
  char firstline[100];
  char moreline[80];
  */
  
  /* Allocate memory for covarm: */
  if ((covarm = (float *) malloc(COV*sizeof(float))) == NULL)
    { printf("\nNot enough memory to allocate covariance!\n");
      return(1);
    }*/

  if((fpin=fopen(infile,"rt")) == NULL)
    { printf("\nError opening infile\n");
      return FAILURE;
    }
  fseek(fpin,0L,SEEK_SET);

  i = 0;
  eof = NO;
  /* File search: */
  while (eof == NO)
    {
      /* Header information: */
      /* First line */
for (i=0; i< 100 ; i++)  firstline[i] = 0;
for (i=0; i< 100 ; i++)  (*head).lastname[i] = 0;
if (fgets(firstline,100,fpin) == NULL)  eof = YES;  /* 1 */
printf("\nHeader of data file says: ");
puts(firstline);
printf("\n");
if (! (trilptr=strstr(firstline,".")) )= NULL)
{  strcpy(trilptr,"~ ");  }
strcpy(*head).lastname, firstline);

/* Classes */
for (i=0; i< 80 ; i++)  dataline[i] = 0;
strcpy(dataline,"\0");
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 2 */
j = sscanf(dataline, "%d ",&classes);
if (j < 1 || j == EOF)  eof = YES;
(*head).classes = classes;

/* Bands */
for (i=0; i< 80 ; i++)  dataline[i] = 0;
strcpy(dataline,"\0");
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 3 */
j = sscanf(dataline, "%d ",&nbands);
if (j < 1 || j == EOF)  eof = YES;
(*head).bands = nbands;

for (i=0; i< 24 ; i++)  (*head).blank[i] = 0;

/* Develop structures for each signature: */

/* Get Maximum Values */
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 4 */
for (i=0; i< 80 ; i++)  dataline[i] = 0;
strcpy(dataline,"\0");
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 5 */
for (i=0; i< MXBNDS ; i++)  max[i] = 0;
j = sscanf(dataline, "%d %d %d %d %d ",\&max[0], &max[1],
 &max[2], &max[3], &max[4], &max[5], &max[6]);
if (j < 1 || j == EOF)  eof = YES;

/* 2 blank lines */
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 6 */
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 7 */

/* Get Mean Values */
for (clas=0; clas<= classes ; clas++)
{
  for (band=0; band< MXBNDS ; band++)  smean[band][clas] = 0;
  for (i=0; i< 80 ; i++)  dataline[i] = 0;
  if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 8>8+C */
j = sscanf(dataline, "%d %f %d %d %d %d ",\&i,
 &fcount,&smean[0][clas],&smean[1][clas],
 &smean[2][clas],&smean[3][clas],&smean[4][clas],
 &smean[5][clas],&smean[6][clas]);
count[clas] = (long)fcount;
if (j < 1 || j == EOF)  eof = YES;
}

/* 1 blank line */
if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 8+C+1 */

/* Get Covariances */
for (clas=0; clas<= classes ; clas++)
{
  /* 1 blank line */
  if (fgets(dataline,80,fpin) == NULL)  eof = YES;  /* 8+C+2 */
  k = 0;
}
for (band=0; band< nbands ; band++)
{
    for (i=0; i< 80 ; i++) moreline[i] = 0;
    strcpy(moreline, "\0");
    if (fgets(moreline,80,fpin) == NULL) eof = YES; /* 8+C+3 */
    j = sscanf(moreline, "%f %f %f %f %f %f %f", &cova[0],
               &cova[1],&cova[2],&cova[3],&cova[4],&cova[5],&cova[6]);
    if (j < 1 || j == EOF) eof = YES;
    if (k > 32)
    { printf("\nError in covar computation!\n");
      return FAILURE;
    }
    for (i=band; i< nbands ; i++)
    { /* covarm[k+clas*32] = cova[i]; */
      class[clas].covar[k] = cova[i];
      k++;
    }
}

eof = YES;
}

for (clas=0; clas< clases ; clas++)
{
    /* Name */
    for (i=0; i< 32 ; i++) nameline[i] = 0;
    for (i=0; i< 32 ; i++) class[clas].name[i] = 0;
    sprintf(nameline, "CLASS %d=", clas);
    strcpy(class[clas].name,nameline);

    /* Blank */
    for (i=0; i< 92 ; i++) class[clas].blank[i] = 0;

    /* Minimum Values */
    for (i=0; i< 16 ; i++) class[clas].minvector[i] = 0;

    /* Maximum Values */
    for (i=0; i< nbands; i++) class[clas].maxvector[i] = max[i];
    for (i=nbands; i< 16 ; i++) class[clas].maxvector[i] = 0;

    /* Mean Values */
    for (i=0; i< nbands; i++)
    { fmean = 1.0*smean[i][clas];
      class[clas].meanvector[i]=fmean;
    }
    for (i=nbands; i< 16 ; i++) class[clas].meanvector[i] = 0;

    /* Counts */
    class[clas].scount = count[clas];

    /* Covariances */
    for (i=0; i< 80 ; i++)
    { /*class[clas].covar[i] = covarm[i+clas*32];*/
      /*class[clas].covar[i] = 1.*/
    }

    /* histo */
    for (i=0; i< HISTBYTES; i++) class[clas].histo[i] = 0; /*

    fclose(fpin);
    return SUCCESS;
}
 Function Writesig */
/* This routine writes data to McIDAS .SIG file. */
/* The arrays are all indexed from 0 to n-1. */
/* Sig Data: */
/* 1) name of sig file */
/* 2) number of classes */
/* 3) number of bands */
/* 4) blank (says "--Max data values--") */
/* 5) max=B1 max=B2 ... max=BN */
/* 6) blank (says "---BAND MEANS---") */
/* 7) blank (says "Class Counts " then bands) */
/* 8) class# counts mean=B1 mean=B2 ... mean=BN */
/* C+1 of these... */
/* 8+C+1) blank (says "Class covariance matrices") */
/* 8+C+2) class# */
/* 8+C+3) COVAR(0,0) COVAR(0,1) ... COVAR(0,N-1) */
/* N of these... */
/* C of these... */
int writesig(char *outfile, Sbdheader *head, Signature class[]) {

#define cov 32*MXCLAS

int i,j,k;
int clas, band, classes, nbands, eof, maxes[MXCLAS];
int max[MAXSBD], min[MAXSBD], smean[MAXSBD][MXCLAS];
long count[MXCLAS];
float fcount, fmean, cova[32];
float covar[MXBNDS][MXBNDS];
float *covarm;
FILE *fpin;
char *trlptr, nameline[32];
char dataline[70];
char firstline[100];
char moreline[80];

if((fpin=fopen(outfile,"wt")) == NULL)
{
  printf("nError opening outfile
");
  return FAILURE;
}

fseek(fpin, 0L, SEEK_SET);

nbands = (*head).bands;
classes = (*head).classes;
i = 0;
eof = NO;
for (i=0; i< 70; i++) dataline[i] = 0;

/* File search: */
while (eof == NO)
{
  /* Header information: */
  /* First line */
  strcpy(dataline, (*head).lanname);
  if (fputs(dataline,fpin) == NULL) eof = YES; /* 1 */

  /* Classes */
  j = sprintf(dataline, "\n%d Classes", (*head).classes);
  if (fputs(dataline,fpin) == NULL) eof = YES; /* 2 */
  if (j < 1 || j == EOF) eof = YES;

  /* Bands */
  j = sprintf(dataline, "\n%d Bands", (*head).bands);
  if (fputs(dataline,fpin) == NULL) eof = YES; /* 3 */
  if (j < 1 || j == EOF) eof = YES;

  j = sprintf(dataline, "\n--Max data values--");
  if (fputs(dataline,fpin) == NULL) eof = YES; /* 4 */

  /* Develop structures for each signature: */
/* Get Maximum Values */

for (i=0; i<classes ; i++) maxes[i]=0;
for (i=0; i<classes ; i++)
{ for (j=0; j<nbands; j++)
   { if(maxes[j] < class[i].maxvector[j])
      maxes[j]=class[i].maxvector[j];
   }
 }

for (i=0; i< 70 ; i++) dataline[i] = 0;
j = sprintf(dataline,"\n");
for (i=0; i< nbands ; i++)
{ j=sprintf(dataline+1+i*6,"%5d ",maxes[i]);
}
if ( j < 1 || j == EOF) eof = YES;
if (fputs(dataline,fpin) == NULL) eof = YES;  /* 5 */

/* 2 info lines */
j=sprintf(dataline,"\n
---BAND MEANS---");
if (fputs(dataline,fpin) == NULL) eof = YES;  /* 6 */
j=sprintf(dataline,"\nClass Counts Band Means\n");
if (fputs(dataline,fpin) == NULL) eof = YES;  /* 7 */

/* Write Mean Values */

for (clas=0; clas< classes ; clas++)
{ fcount = (float)class[clas].scount;
j=sprintf(dataline,"%3d %7.1f ",clas,fcount);
for (i=0; i< nbands ; i++)
{ j=sprintf(dataline+12+i*6,"%5d ",
   (int)class[clas].meanvector[i]);
}
if ( j < 1 || j == EOF) eof = YES;
if (fputs(dataline,fpin) == NULL) eof = YES;  /* 8+8+C */
if (fputs("\n",fpin) == NULL) eof = YES;
}

/* Covariance line */

if (fputs(dataline,"Class covariance matrices"));
if (fputs(dataline,fpin) == NULL) eof = YES;  /* 8+C+1 */

/* Write Covariances */

for (clas=0; clas< classes ; clas++)
{ /* 1 Number line */
if (fputs("\n",fpin) == NULL) eof = YES;
strcpy(dataline,class[clas].signame);
if (fputs(dataline,fpin) == NULL) eof = YES;  /* 8+C+2 */

/* Unfold covariance */

k = 0;
for (band=0; band< nbands ; band++)
{ if (k > 32)
    { printf("Error in covar computation!");
      return FAILURE;
    }
for (i=band; i< nbands ; i++)
{ covar[band][i] = class[clas].covar[k];
  covar[i][band] = covar[band][i];
  k++;
}
for (band=0; band< nbands ; band++)
{ if (fputs("\n",fpin) == NULL) eof = YES;
  for (i=0; i< nbands ; i++)
   { 

j=sprintf(dataline+i*9,"%8.2f ",covar[band][i]);
}
if (j < 1 || j == EOF) eof = YES;
if (fputs(dataline,fpin) == NULL) eof = YES;
}

eof = YES;
}

fclose(fpin);
return SUCCESS;

/*---------------------------------------------*/
/* Function Readsbd */
/* This routine reads data from ERDAS .SBD file. */
/* The arrays are all indexed from 0 to n-1. */
/*---------------------------------------------*/
/* Read in Data: */

int readsbd(char *sdbfile, Sbdheader *head,
Signature class[])
{
    int i,j,k, clas, band, classes, nbands;
    unsigned char zero;
    FILE *fpsbd, *namout;
    char namfile[INPUTWIDTH],classname[32];
    char *trlp;

    /* Read in .SBD file: */
    if((fpsbd=fopen(sdbfile,"rb")) == NULL)
      { printf("Error opening sdbfile\n");
        return FAILURE;
      }
    fseek(fpsbd,0L,SEEK_SET);

    /* Read in Sbdheader: */
    j = fread(*head).laname,sizeof(char),100,fpsbd);
    j = fread(*head).classes,sizeof(int),1,fpsbd);
    j = fread(*head).bands,sizeof(int),1,fpsbd);
    j = fread(*head).blank,sizeof(char),24,fpsbd);

    nbands = (*head).bands;
    classes = (*head).classes;
    /* Read in Signatures: */
    zero = 0;
    if (nbands <= 0 || nbands > MXBNDS || classes > MXCLAS)
      { printf("Error reading signatures from sdb file\n");
        return FAILURE;
      }

    for (i=0; i<=classes; i++)
      {
        j = fread(class[i].signature,sizeof(char),32,fpsbd);
        j = fread(class[i].scount,sizeof(long),1,fpsbd);
        j = fread(class[i].blank,sizeof(char),92,fpsbd);
        j = fread(class[i].minvector[0],sizeof(int),MAXSBD,fpsbd);
        j = fread(class[i].maxvector[0],sizeof(int),MAXSBD,fpsbd);
        j = fread(class[i].meanvector[0],sizeof(float),MAXSBD,fpsbd);
        /* Handle ERDAS bugs: */
if (nbands == 4)
{
    k = fread(&class[i].covar[0], sizeof(float), 1, fpsbd);
    k = fread(&class[i].covar[1], sizeof(float), 1, fpsbd);
    k = fread(&class[i].covar[2], sizeof(float), 1, fpsbd);
    k = fread(&class[i].covar[3], sizeof(float), 1, fpsbd);
    for (j=10; j<32; j++)
        k = fread(&class[i].covar[j], sizeof(float), 1, fpsbd);
}
if (nbands == 3)
{
    k = fread(&class[i].covar[0], sizeof(float), 1, fpsbd);
    k = fread(&class[i].covar[1], sizeof(float), 1, fpsbd);
    k = fread(&class[i].covar[2], sizeof(float), 1, fpsbd);
    k = fread(&class[i].covar[3], sizeof(float), 1, fpsbd);
    for (j=6; j<32; j++)
        k = fread(&class[i].covar[j], sizeof(float), 1, fpsbd);
}
if (nbands != 3 && nbands != 4)
{
    for (j=0; j<32; j++)
        k = fread(&class[i].covar[j], sizeof(float), 1, fpsbd);
}
for (k=0; k<nbands*512; k++) j = fread(&zero, 1, 1, fpsbd);
fclose(fpsbd);
return SUCCESS;

/*============================================================================*/
/* Function Writesbd */
/*============================================================================*/
/* This routine writes data to ERDAS .SBD file. */
/*============================================================================*/
/* The arrays are all indexed from 0 to n-1. */
/*============================================================================*/
/* Read in Data: */
/*============================================================================*/
/* 1) name of sig file */
/*============================================================================*/
/* 2) number of classes */
/*============================================================================*/
/* 3) number of bands */
/*============================================================================*/
/* 4) blank (says "--Max data values--") */
/*============================================================================*/
/* 5) max-B1 max-B2 ... max-BN */
/*============================================================================*/
/* 6) blank (says "---BAND MEANS---") */
/*============================================================================*/
/* 7) blank (says "Class Counts " then bands) */
/*============================================================================*/
/* 8) class# mean-B1 mean-B2 ... mean-BN */
/*============================================================================*/
/* C+1 of these... */
/*============================================================================*/
/* 8+C+1) blank (says "Class covariance matrices") */
/*============================================================================*/
/* 8+C+2) class# */
/*============================================================================*/
/* 8+C+3) COVAR(0,0) COVAR(0,1) ... COVAR(0,N-1) */
/*============================================================================*/
/* N of these... */
/*============================================================================*/
/* C of these... */
/*============================================================================*/
/* int writesbd(char *outfile, Sbdheader *head, 
    Signature class[]) */

int i, j, k, clas, band, classes, nbands;
unsigned char zero;
FILE *fpout, *namout;
char outnamfile[INPUTWIDTH], classname[32];
char *triplptr;

nbands = (*head).bands;
classes = (*head).classes;

/* Open .NAM file and write class names: */
strcpy(outnamfile, outfile);
if ((triplptr=strstr(outnamfile, ".")) != NULL)
    { strcpy(triplptr, ".NAM"); }
if ((namout=fopen(outnamfile, "wb")) == NULL)
    { printf("Error opening namefile
");
        return FAILURE;
    }
seek(namout, 0L, SEEK_SET);
for (i=0; i<classes; i++)
    { strcpy(classname, class[i].signature);
        fprintf(namout, "%s\n", classname);
    }
fclose(namout);

/* Write out .SBD file: */
if ((fpout=fopen(outfile, "wb")) == NULL)
    { printf("Error opening outfile
");
        return FAILURE;
    }
seek(fpout, 0L, SEEK_SET);

/* Write out Sbdheader: */
j = fwrite(&(*head).lanname, sizeof(char), 100, fpout);
j = fwrite(&(*head).classes, sizeof(int), 1, fpout);
j = fwrite(&(*head).bands, sizeof(int), 1, fpout);
(*head).blank[2]=1; /* Histogram=yes */
j = fwrite(&(*head).blank, sizeof(char), 244, fpout);

/* Write out Signatures: */
zero = 0;
if (nbands <= 0 || nbands > MAXBNDS || classes > MXCLAS)
    { printf("Error writing signatures\n");
        return FAILURE;
    }
for (i=0; i<classes; i++)
    { j = fwrite(&class[i].signature, sizeof(char), 32, fpout);
        j = fwrite(&class[i].scount, sizeof(long), 1, fpout);
        class[i].blank[2]=1; /* Histogram=yes */
        j = fwrite(&class[i].blank, sizeof(char), 92, fpout);
        j = fwrite(&class[i].minvector[0], sizeof(int), MAXSBD, fpout);
        j = fwrite(&class[i].maxvector[0], sizeof(int), MAXSBD, fpout);
        j = fwrite(&class[i].meanvector[0], sizeof(float), MAXSBD, fpout);
        /* Handle ERDAS bugs: */
        if (nbands == 4)
            { k = fwrite(&class[i].covar[0], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[1], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[2], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[3], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[4], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[5], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[6], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[7], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[8], sizeof(float), 1, fpout);
                k = fwrite(&class[i].covar[9], sizeof(float), 1, fpout);
                for (j=10; j<32; j++)
                    { k = fwrite(&class[i].covar[j], sizeof(float), 1, fpout); }
            }
    }
else
{
  for (j=0; j<32; j++)
    { k = fwrite(&class[i].covar[j], sizeof(float), 1, fpout); }
}
for (k=0; k<nbands*512; k++) j = fwrite(&zero, 1, 1, fpout);
}
fclose(fpout);

return SUCCESS;
2. A COMPREHENSIVE UNSUPERVISED CLUSTERING TECHNIQUE FOR THE CLASSIFICATION OF REMOTELY SENSED DATA.
A COMPREHENSIVE UNSUPERVISED CLUSTERING TECHNIQUE
FOR THE CLASSIFICATION OF REMOTELY SENSED DATA

by

Gerard R. Peltzer

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

(Civil and Environmental Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

1992
APPROVED:

Frank L. Scarpace, Professor
Civil and Environmental Engineering and
Environmental Studies

Date: Aug 11, 1992
ABSTRACT

An original procedure to classify multispectral images into spectrally similar categories in an automatic, or unsupervised, manner has been developed at the Space Science and Engineering Center of the University of Wisconsin - Madison. This procedure has been integrated into an existing image processing package which provides an accurate, adjustable, and easy to use process for the extraction of thematic information from multispectral data.

The package presented consists of routines for image classification and analysis of classification output. It also includes routines which aid in comparing output with that of other classification methods.

The classification algorithm is recursive and processes an image repeatedly, using a changing set of spectral clusters which evolve to represent the statistical classes present in the data. Throughout the classification, oversized clusters are split using a method analogous to analyzing mass moments of inertia. Poorly separated clusters are merged using Transformed Divergence as a decision function for class separability. The final number of classes produced is a function of the degree of class separation and the class shape desired by the user.

Tests of the classifier show results to be better defined than for a commercially available package and a popular unsupervised routine. In addition, outputs of the classification have good accuracy in test images with ground truth. Additional tests of the classifier show many possible future applications.
ACKNOWLEDGEMENTS

I wish to express my thanks and gratitude to Professor Frank L. Scarpace, my Advisor, whose insight and support are greatly appreciated. I would like to thank Dr. Sanjay S. Limaye of the Space Science and Engineering Center, whose support, patience, and exceptional scientific knowledge are also greatly appreciated. In addition, sincere thanks are extended to the other members of my committee, Professors Thomas M. Lillesand and Ralph W. Kiefer.

This work was supported in part through NASA Contract NAS5-31347: "A Planetary Version of PC-McIDAS." McIDAS is the host package for the algorithms developed for this thesis.

I would like to thank the people at SSEC for their assistance and for a job well done in the development of the McIDAS-X system. David Santek provided McIDAS system programming assistance. Patrick Fry provided programming assistance and X-Windows graphing routines which were very effective and easy to use.

I would like to thank Philip Polzer for providing his data and image classifications for use in testing. Thanks also to Professor Roland Chin, whose courses in Digital Image Processing and Pattern Recognition provided me with a solid background.

I especially wish to express my gratitude to my family and friends for their encouragement and support. I wish to thank my father, Robert G. Peltzer, for providing me with much of the information and software I needed to write this document. I would like to thank Christine A. Wu for her help in editing and formatting, and for the all the support she has given me.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF IMAGES</td>
<td>vi</td>
</tr>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Hypothesis</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Format of Document</td>
<td>3</td>
</tr>
<tr>
<td>2.0 BACKGROUND AND LITERATURE REVIEW</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Symbolic Definitions</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Decision Functions</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Distance Functions</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Clustering</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Criterion Functions</td>
<td>9</td>
</tr>
<tr>
<td>2.5.1 Minimum Variance Criterion</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Decision Directed Approaches</td>
<td>10</td>
</tr>
<tr>
<td>2.7 The ISODATA Classifier</td>
<td>11</td>
</tr>
<tr>
<td>2.7.1 Splitting Clusters</td>
<td>13</td>
</tr>
<tr>
<td>2.7.2 Merging Clusters</td>
<td>14</td>
</tr>
<tr>
<td>2.7.3 ISODATA Illustrated</td>
<td>14</td>
</tr>
<tr>
<td>3.0 METHODOLOGY</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Problems with ISODATA</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Splitting Clusters</td>
<td>18</td>
</tr>
<tr>
<td>3.2.1 Standard Splitting Problems</td>
<td>18</td>
</tr>
<tr>
<td>3.2.2 Mass Moments of Inertia</td>
<td>20</td>
</tr>
<tr>
<td>3.2.3 An Improved Splitting Procedure</td>
<td>22</td>
</tr>
<tr>
<td>3.3 Merging Clusters</td>
<td>23</td>
</tr>
</tbody>
</table>
3.3.1 Standard Merging Problems .............................................24
3.3.2 Divergence .................................................................24
3.3.3 An Improved Merging Procedure ......................................26
3.4 An Improved Classifier .....................................................26

4.0 IMPLEMENTATION ....................................................................28
4.1 Software/Hardware Packages Used .........................................28
4.1.1 McIDAS ..............................................................................28
4.1.2 ERDAS ...............................................................................29
4.1.3 File Translation .................................................................30
4.2 The McIDAS Classifier ..........................................................31
4.2.1 USCLAS .............................................................................32
4.2.2 Cluster Visualization .........................................................37
4.3 Performing a Classification ......................................................39

5.0 APPLICATIONS AND TESTS .....................................................41
5.1 A Challenging AVHRR Scene ..................................................42
5.2 Landsat Scene of Central Wisconsin ........................................44
5.3 Voyager Planetary Image ........................................................46

6.0 SUMMARY .............................................................................49

BIBLIOGRAPHY ..............................................................................50

APPENDIX A - IMAGES ....................................................................A1

APPENDIX B - SOURCE CODE .........................................................B1

B.1 USCLAS Classifier .....................................................................
B.2 ISO Standard ISODATA Splitting and Merging Routines ............
B.3 ELLIPS Cluster Viewing Utility ................................................
B.4 ATOERD File Translation Utility ..............................................
B.5 ERDTOA File Translation Utility ..............................................
B.6 SBDSIG File Translation Utility ..............................................
LIST OF IMAGES

1. The McIDAS system and USCLAS ................................................. A1
2. Kuwait AVHRR Scene, Visible Band ........................................... A2
3. Kuwait AVHRR Scene, Mid IR Band ........................................... A3
4. Kuwait AVHRR Scene, USCLAS Classification ................................. A4
5. Kuwait AVHRR Scene, ERDAS ISODATA Classification ........................ A5
6. Kuwait AVHRR Scene, USCLAS Color Enhanced ............................. A6
7. Kuwait AVHRR Scene, ERDAS ISODATA Color Enhanced ................. A7
8. Wisconsin TM Scene, USCLAS Classification .................................. A8
9. Wisconsin TM Scene, ERDAS Classification .................................. A9
10. Wisconsin TM Scene, Supervised Classification ............................ A10
11. Jupiter Mosaic, USCLAS Classification with 26 classes ..................... A11
12. Jupiter Mosaic, USCLAS Classification with 26 classes ..................... A12
13. Jupiter Mosaic, USCLAS Classification with 38 classes ..................... A13
1.0 INTRODUCTION

In recent years, the concurrent development of digital imaging systems and powerful, inexpensive computers has resulted in the development of the field of digital image classification. Image classification is a form of pattern recognition where the ultimate objective is to be able to represent an image by a set of classes or themes which describe all of the features present. Classifications are thus, in essence, maps of image features, and can contain spatial, spectral, or temporal information about data in image form.

The majority of image classifications are spectrally based, and are performed on multispectral images (Lillesand and Kiefer, 1987). These images contain recorded intensities of different wavelengths of light at each image element or pixel. The rationale behind spectral classification is that feature types exhibit unique spectral reflectance combinations, or signatures, which are contained in their data values in multispectral images.

The classification methods discussed in this paper are based on this spectral concept, and make use of multispectral data. This does not preclude the use of other types of data as long as they can be presented in the form of multiple data values, or bands, for each image element. For example, a temporal classification could be performed with bands containing data obtained at different times. Also, a spatial element could be introduced with the addition of bands containing ancillary geographic data (Northcutt, 1991). Once again, the rationale is that different feature types will show unique signatures in their multi-band data values.

Image classification can be performed in a supervised or unsupervised manner. In a supervised procedure, sample areas or "training sets", of known feature type are developed by the user and used to describe the spectral signatures of features. Each image element is then classified to the class it most
nearly matches. In unsupervised, or automatic classification, clusters are found which are statistical groupings of similar signatures, and then each element is classified to the most similar cluster. The main advantage of this method over supervised classification methods is that no training sets of known data need to be taken, giving the user a much easier procedure to follow. However, in an unsupervised process the user is left to define the relationship between clusters and feature types after classification. Unsupervised classification is also very useful in gaining insight into the nature or structure of the data, especially in the discovery of distinct new subclasses and departures from expected characteristics (Duda and Hart, 1973).

A wide variety of unsupervised clustering algorithms have been developed for applications in the field of statistical pattern recognition, many of which include heuristic procedures developed as a result of experience gained through experimentation (for some descriptions see Tou and Gonzalez, 1974, Duda and Hart, 1973). One of the most widely used of these procedures in image classification is the "Iterative Self-Organizing Data Analysis Technique" or ISODATA (Tou and Gonzalez, 1974). The advantages of ISODATA are that it is not spatially biased to any edge of the data file, and that it produces better results than other methods for data that are not normally distributed (ERDAS, 1990). The disadvantages include the narrow range of input parameters which lead to convergence, a simplistic, data dependent criteria for cluster splitting and merging, and a lack of control over the distribution and shape of output clusters.

1.1 Hypothesis

The thesis hypothesis is that an unsupervised classifier can be developed which is accurate, easy to use, and gives the user control of the distribution and shape of output clusters according to an absolute index of separability. This
classifier would be based on the ISODATA method, with an improved cluster
splitting criterion analogous to mass moments of inertia, and a merging criterion
using Transformed Divergence as a decision function to improve accuracy and
ease of use.

1.2 Format of Document
In this thesis, the development and testing of an unsupervised
classification procedure is illustrated. In Chapter Two, methodologies for
multispectral image classification are reviewed, and the types of variables used
and terms describing them are defined. The ISODATA method is developed in
detail at the end of Chapter Two. In Chapter Three, the strengths and
weaknesses of ISODATA and other classifiers are discussed, and procedures
for a new classifier are derived in detail. In Chapter Four, the implementation of
this new classifier is presented, including additional routines which aid in
analysis of the classification output. In Chapter Five, classifiers are applied to
several test images, and the accuracy and usefulness of the new classifier is
assessed.
2.0 BACKGROUND AND LITERATURE REVIEW

Multispectral classification can be described as using pattern recognition to transform an image into a map of information classes, so any discussion of the subject needs to start with a description of the field of statistical pattern recognition. In pattern recognition, the question to be dealt with is: how does one specify a procedure for partitioning data? Whether separating apples from oranges or specifying land cover classes, the principal function of a pattern recognition system is to provide decisions to the user concerning the class membership of the patterns in the data encountered (Tou and Gonzalez, 1974).

2.1 Symbolic Definitions

Before moving further, a mathematical notation will be developed which can be used to describe the subjects covered in this paper. In general, bold face characters are used to denote vectors, and capitals are used to denote matrices. The matrix transpose is denoted by $T$, and vectors used will often be presented in their transposed format for convenience.

The mathematics of pattern recognition often involves use of a multi-dimensional geometrical pattern space. This pattern space is used to describe multispectral data, and is a geometrical construct consisting of a separate dimension for each type of data measurement. For example, in an $n$ band multispectral image classification there will be $n$ dimensions, one for each spectral band of data. For visualization, data is often presented in a scatter plot format, with multiple 2-dimensional scatter plots shown for data with more than two dimensions in pattern space.

Multispectral data values can be described as pattern vectors: $x = (x_1 \ x_2 \ x_3 \ ...)^T$. Here $x$ represents all the data for one image picture element; each $x_i$ is the value of the $i$th band of data.
The mean vector of a class \( k \) is described as: \( \mathbf{m}_k = (m_1 \ m_2 \ m_3 \ldots)^T \). Here \( \mathbf{m} \) represents the vector of means for a class of data; each \( m_i \) is the value of the mean of the class in the \( i \)th band of data. The terms 'class' and 'cluster' will often be used together, and have essentially the same meaning here.

It is often reasonable to assume that classes are sets of data containing multivariate Gaussian distributions. In this case, a measure of the joint variation of two variables about their common mean can be developed (Jensen, 1986). This measure is called the covariance and is defined as: \( c_{ij} = E[(x-m_i)(x-m_j)] \). Here \( E \) denotes the expected value of a function and \( c_{ij} \) represents the covariance between dimensions \( i \) and \( j \). Computationally, the covariance is usually represented as a matrix for a given set of data. The covariance matrix of a class \( k \) is represented as:

\[
K_k = \begin{bmatrix}
c_{11} & c_{12} & \ldots & c_{1n} \\
c_{12} & c_{22} & \ldots & c_{2n} \\
c_{1n} & c_{2n} & \ldots & c_{nn}
\end{bmatrix}
\]

Note that the covariance matrix is symmetrical; \( c_{ij} = c_{ji} \) always.

The diagonal elements of the covariance matrix, \( c_{ii} \), are actually the variances of the data set in each band. The square root of these variances gives the standard deviation vector for the class, which is defined as: \( \sigma_k = (\sigma_1 \ \sigma_2 \ \sigma_3 \ldots)^T \).

The covariance matrix contains information about the similarity and relationships between different bands in the data, but at first glance is difficult to interpret. Another way to interpret this information is with the correlation matrix, which is easily derived from the covariance. The correlation matrix contains elements which give a correlation for data values in one band versus another; each element is computed as: \( c_{ij} / (\sigma_i \sigma_j) \) (Jensen, 1986). For example, the
correlation between bands 1 and 2 is $c_{12}/(\sigma_1\sigma_2)$. Correlations run from -1 to 1, with 1 indicating band i's data varies in exactly the same way as band j, -1 being a perfect negative relationship between the two, and 0 being no correlation whatsoever.

2.2 Decision Functions

The question of how to partition data in a classification actually involves two separate issues: how should one measure the similarity between samples, and how should one evaluate a partitioning of a set of samples into clusters. (Duda and Hart, 1973) The traditional approach to these problems is to use a distance function to develop a set of decision functions in a multi-dimensional pattern space. Decision functions are mathematical boundaries separating one class of data from another, and a set of decision functions for all possible class pairs is all that is needed to classify a data set. As an example, in an image with two dimensions of data, a decision function between two classes would simply be a line or curve delineating the border between classes (see Figure 1). For data of three dimensions, a decision function becomes a plane or surface, and in more dimensions decision functions are referred to as hyperplanes or hyperspace surfaces (Tou and Gonzalez, 1974).

![Fig. 1. Decision function for two classes:](image-url)
2.3 Distance Functions

An important approach to finding the similarity between samples is the use of distance functions in pattern space. Distance functions can be expressed as decision functions, and are a measure of a data point's similarity to a class. The relationship is essentially the smaller the distance, the greater the similarity. As an example, in Figure 1, an unknown data point could be classified as either class 1 or class 2 based on the shortest Euclidean distance to the mean vector of each class. This type of distance function is called Minimum Distance to Mean, or MDM, and can be described as \( d = ||x - m|| \). Here \( ||x - m|| \) represents the Euclidean distance \( \sqrt{(x1 - m1)^2 + (x2 - m2)^2 + ... + (xn - mn)^2} \). Considering all possible values of \( x \), the MDM function produces a linear decision function between two classes. In two dimensions, the decision boundary between class 1 and class 2 is a line normal to the vector from the first class mean to the second.

There are other meaningful distance functions which are sometimes useful in classification. The Mahalanobis distance, for example, is a useful measure of similarity when statistical properties of clusters are known. This distance is given as \( d = (x - m)^T K^{-1} (x - m) \), and is derived from the probability density function of a multivariate Gaussian distribution (Tou and Gonzalez, 1974). The Mahalanobis distance function produces a parabolic decision function between two classes where the focus in each dimension is the class mean with the smallest variance.

2.4 Clustering

Once a certain form of distance function is decided upon, the problem of classification becomes one of determining the best parameters. Theoretically,
applying standard parametric classifiers such as the maximum likelihood or Bayesian approach would be useful, but unfortunately these approaches do not yield analytically simple results (Duda and Hart, 1973). With these approaches, exact solutions have computational requirements that grow exponentially with the number of samples, making all but the simplest examples unmanageable. There is another set of classifiers, however, which are not subject to this limitation. These classifiers use a process called clustering, and there are many varieties of clustering routines. Duda and Hart (1973) present an excellent literature review of the history of the methods used in unsupervised clustering. Some of the earliest work in clustering was done in the field of numerical taxonomy, and Sokal and Sneath (1963) present references to this work. The remainder of this chapter will focus on the methodologies leading to the development of the ISODATA algorithm, which was first presented by Ball and Hall (1967) and described in detail in Tou and Gonzalez (1974).

To pursue the problem of clustering, some assumptions must first be made about the nature of image data. From a geometrical perspective, multispectral data can be viewed as clouds of points in an n-dimensional space. By assuming these clouds are normal distributions, a compact statistical description of the data can be developed. But what if the actual information classes in the image are not normally distributed and their probability distributions are unknown? In this case the normal assumption is still useful because the clustering procedure can approximate complex density functions with a mixture of normal distributions (Duda and Hart, 1973). The algorithm is not concerned with which combinations of normal distributions produce information classes; that task is left to the operator.
2.5 Criterion Functions

One way to determine parameters and develop a well-defined clustering routine is to define a criterion function that measures the clustering quality of any partition of the data (Duda and Hart, 1973). Criterion functions are designed to be minimized when the criterion for the best possible clustering according to the function is met. Some examples of criterion functions include the average squared distances between samples in each cluster, the scatter matrix criterion, and the trace and determinant criteria. Numerous other performance indices used in pattern recognition are useful in clustering.

2.5.1 Minimum Variance Criterion

One of the most useful and widely used criterion functions is the sum of squared errors function. This function is also known as the minimum variance criterion (Murtagh, 1990), and is defined as:

$$ J = \sum_{i=1}^{N_c} \sum_{x \in C_i} \| x - m_i \|^2. $$

Here, $J$ is the sum of squared error criterion, $N_c$ is the number of clusters, and $C_i$ is cluster $i$. The sum of squared error is the sum of the single class error criteria over all the clusters being analyzed. Each single class criterion is the sum of the distance from the mean squared for every data vector contained in the cluster. What $J$ really represents is the total squared error incurred in representing the $n$ samples ($x$) by the $N_c$ cluster means ($m_i$) (Duda and Hart, 1973). This criterion is similar to a least mean square error (LMSE) algorithm, where the summation of errors to be minimized is proportional to an average or mean value (Tou and Gonzalez, 1974).

As with all criterion functions, the minimum variance criterion has its advantages and disadvantages. When the data forms clusters of compact, well-separated clouds it performs well, but, as can be expected, difficulties arise with
clusters that are not clearly defined. A less obvious problem arises when there are large differences in the number of samples in adjacent clusters (Duda and Hart, 1973). In this case, the larger cluster is partitioned poorly because a small reduction in the squared distance is multiplied by a large number of data points. This is shown in Figure 2, where the sum of the squared error is lower for the poorly defined partition of (b). This problem can be dealt with by the addition of other secondary criterion functions, which will be described later in this paper.

Fig. 2. The problem of classifying large clusters:

![Diagram showing the problem of classifying large clusters](image)

The sum of the squared error is larger for (a) than for (b).
(From Duda and Hart, 1973).

2.6 Decision Directed Approaches

Now the question at hand is how to implement the above criterion, decision, and distance functions into a useful unsupervised clustering routine. Duda and Hart (1973) concisely describe the best way to proceed as the decision directed approach, which is "to use the a priori information to design a classifier and to use the decisions of this classifier to label the samples." What this infers is that the classifier should use the knowledge accumulated from processed samples to alter the decision functions of the classifier. If these alterations are made to optimize a criterion function, a fairly robust classification scheme can be developed. An algorithm can follow this procedure by
processing the data set in a one-pass manner, or it can be designed to repetitively process the entire data set in a process of iterative optimization. The main advantage of one-pass methods is their speed, but they cannot compete with fully repetitive methods in thorough processing of data. Some of the more popular variations of one-pass methods include sequential clustering (ERDAS, 1990), and maximum-minimum distance algorithms (Tou and Gonzalez, 1974).

There are many varieties of iterative optimization procedures which follow the decision directed approach and make decision function alterations according to a criterion function. One of the most powerful classifiers developed recently is the AUTOCLASS algorithm, developed at the NASA Ames Research Center for analysis of Infrared Astronomical Satellite data (Denning, 1989). The AUTOCLASS program applies Bayesian inference to determine the most probable classification of the data, assuming normal distributions for classes. Construction of a set of all possible hypothesis would far exceed the capacity of any existing supercomputer, so AUTOCLASS constructs a locally most-likely hypothesis iteratively, using posterior probability as a criterion (Denning, 1989). Although very effective, AUTOCLASS is computationally intensive, requiring approximately 36 hours on a Symbolics mainframe to classify a star field of 5425 objects with 94 spectral bands of data each (Denning, 1989).

2.7 The ISODATA Classifier

The minimum variance criterion can also be used in an iterative optimization approach to image classification. This class of algorithm is generally named after the ISODATA method, an acronym for Iterative Self-Organizing Data Analysis Technique. Several versions of the ISODATA algorithm have been developed, starting with the basic iterative optimization and moving on to include secondary criteria and heuristic procedures.
The basic procedure for the ISODATA classifier is as follows. Initially choose a set of \( k \) cluster means. For the first iteration, classify each sample to the nearest cluster mean: \( x \in C_i \) if \( \| x - m_i(1) \| < \| x - m_j(1) \| \), for all classes \( j \).

Next, find the new cluster centers which minimize the sum of squared distances from all points in each cluster \( C_i \) to a new mean for iteration 2, \( m_i(2) \). The beauty of this criterion is that the new set of \( m \) vectors which fulfill this requirement is simply the sample mean of all the elements classed in \( C_i \) (Tou and Gonzales, 1974, Duda and Hart, 1973). So each cluster \( i \) is assigned a new mean given by \( m_i(2) = \frac{1}{n_i} \sum_{x \in C_i} x \). The data is then re-classified using the new cluster means, and the means are once again updated to minimize the sum of squared distance. The process is repeated until the cluster means cease to vary significantly from one iteration to the next. When this convergence is reached, the user is presented with a set of classes which describe the locations of the local maxima in the data. This basic ISODATA routine is also known as the K-means algorithm for its function of allowing the analyst to specify the number of clusters (k) to be found in the data (Lillesand and Kiefer, 1987).

The basic ISODATA routine can be expected to yield good results when the data are arranged in clusters which are relatively far apart, and when the number of clusters in the data is known. For most practical applications, experimenting with the number of clusters and the initial cluster placement is required to obtain a good classification of data. This can be a time-consuming process considering that the image is processed repetitively for each classification, and multiple classifications must be performed to get good results.

To improve on the theme of the basic ISODATA, various secondary criteria can be added to the iterative optimization. Some of these procedures
are heuristic in nature, having been developed through experimentation with what 'looks good'. A definitive example of the standard full ISODATA routine is described in Tou and Gonzalez (1974); its routines for cluster splitting and merging will be discussed here.

One improvement is to optimize the initial placement of cluster means in the pattern space of the data set. A technique shown experimentally to be a good method is to spread the initial cluster means linearly and evenly from the minimum data value in each band to the maximum value in each band (ERDAS, 1990). In addition, deleting clusters which contain few or no pixels after each iteration can be used to rid the output of classes with no useful information.

2.7.1 Splitting Clusters

The two most important secondary criteria used in the full ISODATA routine are cluster splitting and merging. Cluster splitting involves breaking up clusters which are physically too large, dimensionally too elongated, or whose distribution is too far from a normal distribution. In the standard ISODATA, a splitting criterion is applied to all clusters on every even iteration. This criterion is applied by taking the standard deviation vector for each cluster \( j \), \( \sigma_j = (\sigma_{1j}, \sigma_{2j}, \ldots, \sigma_{nj}) \), and finding the maximum component \( \sigma_{j_{max}} \). In addition, the average distance of samples from the cluster mean vector in each class \( j \) is computed:

\[
\overline{D}_j = \frac{1}{N_j} \sum_{x \in C_j} ||x - m_j||, \quad \text{with } N_j \text{ being the number of samples in cluster } j.
\]

From this, an overall average distance is obtained by:

\[
\overline{D} = \frac{1}{N} \sum_{C_j} N_j \overline{D}_j,
\]

where \( N \) is the total number of data elements in the image. Finally, the criterion is: if, for a class \( j \), \( \overline{D}_j > \overline{D} \) and \( \sigma_{j_{max}} > \Phi \), then split cluster \( j \) into two new clusters. \( \Phi \) is the split factor, input by the user. Thus the cluster is split if its
greatest variance in any band is larger than a maximum allowable value and if it is of larger than average size. The two new clusters are assigned the mean vector of the original cluster, then an amount equal to a fraction of $\sigma_{j_{\text{max}}}$ is added to the mean vector of the first and subtracted from the mean vector of the second. This amount should be large enough to make a detectable difference, yet not so large as to cause the new means to interfere with other clusters.

2.7.2 Merging Clusters

Cluster merging in the ISODATA routine involves combining clusters whose characteristics are too similar according to some criterion. In the standard ISODATA, clusters are merged simply if their means are too close together. Functionally, cluster merging takes place on even iterations, or when no cluster splitting occurred. The pairwise distances between cluster centers are first computed: $D_{ij} = \| m_i - m_j \|$, for every combination of classes $i$ and $j$. The $L$ smallest pairs with distances that are less than a distance of $\Theta$ apart are merged. Here $L$ is the maximum number of mergings and $\Theta$ is the merging factor; both are input by the user. The new merged cluster mean is a weighted average of the previous means: $m^* = \frac{1}{N_1 + N_2} (N_1 m_1 + N_2 m_2)$, where $N_1$ and $N_2$ are the number of pixels in the two original clusters.

2.7.3 ISODATA Illustrated

The standard ISODATA procedure is illustrated in Figure 3. In this two-dimensional example, a scatterplot of a hypothetical data set is shown. Initially, the cluster means are distributed equally from the minimum to maximum values of the data in both bands, and are shown as 'x's. At the end of the first iteration, the means migrate to new positions (shown as '+'s), with the two clusters at the extreme ends being removed as empty clusters. Note that the clusters are moving toward the maxima of the data set, with the exception of the cluster
located between two maxima in the upper right. This cluster is split as its variance in band 2 is quite large relative to the other clusters. By the third iteration (shown as "*"s), the split has produced two clusters which are much closer to peaks in the data. In the upper right corner, there are two clusters which are relatively close to one another. These two would be merged if the merging parameter is greater than their pairwise distance.

Fig 3. ISODATA Procedure:

2-Dimensional Example showing movement of cluster means.
3.0 METHODOLOGY

The standard ISODATA routine is an effective means for classifying images in an unsupervised manner. However, there are several major drawbacks to the routine which make it difficult to apply and limit its effectiveness. The goal of this chapter is to describe these limitations and how they can best be overcome to produce a better classifier.

Before going into the specific strengths and weaknesses of the ISODATA routine, one central weakness of all iterative optimization routines needs to be addressed. Ideally, any clustering routine applied to an image would either find all the classes inherent in the data, or would have misclassifications if it were not comprehensive enough. Unfortunately, in the real world there is a significant problem with this, for in an unsupervised setting, there is no definitive right answer; the true number of classes may be impossible to define in a realistic scene with gradually changing features. To compound this problem, the nature of iterative optimization routines is that different initial conditions will lead to different results. Iterative routines such as ISODATA can be unstable with respect to initial conditions, and a small change in input parameters can result in a large change in output results. This is because the convergence criterion has no memory and only works to minimize its error from the current conditions. It is extremely difficult to produce an unsupervised classifier which will always converge to a single 'right' answer, but it is possible to design a classifier which responds to input parameter changes in a predictable manner. Such a classifier would be very useful, providing a highly adjustable process which could be customized for different uses.
3.1 Problems with ISODATA

Although a good unsupervised classifier, the standard ISODATA routine has some significant weaknesses. The most significant problem with ISODATA involves the input parameters given by the user. As mentioned previously, ISODATA can be unstable with respect to changes in initial values. To compound the problem, the routine requires the user to provide values for these numerous initial parameters, some of which are data dependent. This has the result of forcing the operator to perform classifications repeatedly in a search for good results. In the standard ISODATA, there are essentially 5 input parameters: the initial number of classes, the separate splitting and merging factors, a maximum number of pairs to be merged, and a convergence threshold. Anderberg (1973) describes a version of the ISODATA method which requires 7 initial parameters.

One solution to this problem of too many parameters is to fix some of the variables to values which generally provide useful results for a range of input images. This can be done by defining each parameter to be a certain fraction of a maximum value encountered in the data set. Although certainly making the routine easier to use, this approach reduces its adaptability. A general rule in unsupervised clustering is that, as in the field of high-performance aircraft, more maneuverability means less stability and the need for a more highly qualified operator.

The best solution to this dilemma of clustering parameters however, is to reduce the number of required parameters and define them in terms of an absolute index. In the best possible classifier, all of the new parameters would be data independent, would have reliable default settings, and would produce output changes predictably. To build such a classifier, the ISODATA routine must be stripped down and rebuilt using a better set of secondary criteria.
3.2 Splitting Clusters

The first secondary criterion to deal with is the splitting criterion. The effect of cluster splitting on the overall classification should be relatively small, yet tests of the standard ISODATA implemented by the author have shown it to destabilize the routine, forcing more iterations before convergence. In addition to occasionally leading to a lack of convergence, this tends to make the routine's output unstable with respect to the splitting parameter $\Phi$.

3.2.1 Standard Splitting Problems

The standard ISODATA routine splits clusters when the greatest variation of sample data in one band is large. Here the user defines what is considered large by selecting the value of $\Phi$. As a parameter, $\Phi$ can be input as a straight value, a percentage of the largest variance in the data, or a ratio of the largest variance over the smallest variance present in the data. All of these methods have been implemented in routines written by the author, and the ratio method has been found to produce the most reasonable working values. Regardless of the method used however, the splitting criterion still remains a simplistic method for splitting clusters.

There are three fundamental weaknesses with splitting routines that lead to poor judgment in cluster splitting and output instability when changing values of $\Phi$. The first is that splitting is not determined (especially in the percentage approach) solely on one cluster's attributes, but from a relative index which changes with every iteration. The second problem is that there is only one splitting factor which must work for all image bands used in the classification. This can cause problems because if the variances are consistently larger in one band of the data, then that band will always determine cluster splitting. This common problem occurs when classifying data from scanners such as
Landsat Thematic Mapper, where the thermal infra-red channel is statistically quite different from the other channels.

A third important weakness of the ISODATA splitting criterion (especially in the ratio approach) is that it analyzes the separability only along the axes of the bands in the pattern space of the classification. Splitting may not occur where it should because the data may have its maximum variance along axes not aligned along spectral channels. Figure 4 illustrates this condition by showing two clusters that are equally likely to be split according to the standard splitting criterion. Of the two, it is obvious that (b) should be the cluster to be split, but the variances in each band do not provide enough information to come to that conclusion.

Fig. 4. The problem of the ISODATA splitting criterion.

So how can a better criterion for cluster splitting decisions be developed? To find the answer, it is necessary to discuss the factors leading to a decision to split a cluster into separate classes. The first thing to keep in mind is the MDM nature of the classification. Because of this, clusters with a normal distribution of similar proportions in all dimensions are to be desired. In looking at (b) of Figure 4, two things seem important in determining that it is the preferable
cluster to split. First, the ratio of the longest dimension of the cluster to the
shortest dimension is a good indicator of how far the cluster is from a uniform
normal distribution. Second, the distribution of samples (e.g. in clusters or
smoothly distributed) is an indication of whether the cluster data is distributed
normally or bimodally.

3.2.2 Mass Moments of Inertia

There is a way to incorporate these observations into a useful splitting
criterion. This criterion uses a method which is based on the concept of mass
moments of inertia from the field of mechanics. A moment of inertia is
essentially a measure of the resistance of a body to angular acceleration about
an axis of rotation (Hibbeler, 1986). What the method of moments of inertia can
provide in terms of cluster splitting is an indication of the mass distribution of the
contents of a cluster, and a means of measuring this distribution along the
longest axis of the cluster.

The moment of inertia of an object about any axis is given by the parallel
axis theorem: \( I = I_{obj} + md^2 \), where \( I_{obj} \) is the moment of inertia of the object,
\( m \) is the mass, and \( d \) is the normal distance to the axis. For a point mass, \( I_{obj} \) is
zero and \( m \) is equal to 1. For any normal three-dimensional object, there are
three different moments of inertia and three products of inertia which describe
the object’s inertia about all possible axes. For an object filling a volume \( v \) with
uniform density, the moments of inertia about the \( x \), \( y \), and \( z \) orthogonal axes are
respectively:

\[
I_{xx} = \int_{v} (y^2 + z^2)dv, \quad I_{yy} = \int_{v} (x^2 + z^2)dv, \quad I_{zz} = \int_{v} (x^2 + y^2)dv.
\]

The products of inertia are respectively:
\[ I_{xy} = -\int v(x y) dv, \quad I_{xz} = -\int v(x z) dv, \quad I_{yz} = -\int v(y z) dv. \]

These expressions can be derived discretely for an object consisting of separate point masses (such as a cluster in the ISODATA routine). In this case, the integral is replaced by a summation. \( I_{i j} = \sum_{x} \sum_{i \neq j} (x_i - m_i)^2 \) represents the moment about an axis \( j \). The mean vectors have been included here to find the moment for an origin at the object's mean vector while using the general pattern space coordinate system. The product of inertia can be expressed as

\[ I_{i j} = -\sum_{x} (x_i - m_i)(x_j - m_j) \] for the product involving axes \( i \) and \( j \).

The inertial and product moments can be combined into a matrix called the inertial matrix (Greenwood, 1988). This 3 by 3 matrix contains all the information needed to know an object's mass distribution about any axis, and is given by

\[
I = \begin{bmatrix}
I_{xx} & I_{xy} & I_{xz} \\
I_{xy} & I_{yy} & I_{yz} \\
I_{xz} & I_{yz} & I_{zz}
\end{bmatrix}.
\]

To better develop a feeling for an object's mass distribution about an axis, the concept of a radius of gyration can be defined. It can be seen that a moment of inertia about an axis is the integral of all the mass elements making up the object, multiplied by the square of the distance from that axis. The effective (or root mean square) value of this distance for an object is known as the radius of gyration for the given axis (Greenwood, 1988). This radius is defined as
\[ k_i = \sqrt{\frac{l_{ii}}{m}} \] for an object mass \( m \) and axis \( i \) in the pattern space of the classification. The radius of gyration is an extremely useful concept in cluster splitting, giving an indication of how far, on average, samples are located from any axis passing through the mean vector of the cluster.

Another useful concept related to the inertial matrix is that of principal axes. For any given object, there always exists a set of axes for which the products of inertia are all equal to zero, and the inertial matrix is diagonal. In this special case, the axes are called the principal axes, and the corresponding moments are the principal moments of inertia. Specifically, the principal moments are found from the eigenvalues of the inertial matrix; the corresponding eigenvectors give the principal axes. The key to the principal moments of inertia is that they, as eigenvalues, contain the largest and smallest possible values for the system. This translates into knowing the largest and smallest radii of gyration, and the longest and shortest axes of an object.

3.2.3 An Improved Splitting Procedure

At this point, a new criterion for cluster splitting can be developed from the above concepts - a criterion where each cluster is judged individually and there is no comparison between clusters. The procedure starts by finding the inertial matrix for the cluster to be considered. The principal axes and moments of inertia are found, and the ratio of the longest to shortest dimensions of the cluster in terms of radius of inertia is computed. This length to width value is then compared to the splitting parameter, which now represents an absolute measure of the 'roundness' of the cluster. For example, a split factor of three would split any cluster whose longest dimension is greater than three times its smallest. If the cluster is split, the two new means are moved apart by one radius of gyration in the direction of the longest axis of the cluster. This criterion
is not prevented from using more than three bands of data, as the mathematics remain valid for multidimensional inertial matrix operations.

After some consideration, it may seem that the actions of the inertial and covariance matrices are similar, and that this splitting procedure could be related to the covariance matrix of the data. This is indeed the case, and the principal axes can be found through a principal components analysis of the covariance matrix. Alternatively, a simple relationship between the inertial and covariance matrices can be derived. Each diagonal element of the covariance matrix (relating to the variance) can be expressed as: 

$$C_{ii} = \frac{1}{(n-1)} \sum_{x}^{data} (x_i - m_i)^2,$$

where \(n\) is the data count of the cluster. The diagonal elements of the inertial matrix can then be expressed as: 

$$I_{ii} = (n-1) \sum_{j \neq i}^{bands} C_{jj}.$$ 

Similarly, the non-diagonal elements, 

$$C_{ij} = \frac{1}{(n-1)} \sum_{x}^{data} (x_i - m_i)(x_j - m_j),$$

can be used to make 

$$I_{ij} = -(n-1)C_{ij}.$$ 

The inertial matrix can now be written from components of the covariance matrix as:

$$I = \begin{bmatrix} I_{12} & I_{12} & I_{1n} \\ I_{12} & I_{22} & I_{2n} \\ I_{1n} & I_{2n} & I_{nn} \end{bmatrix} = (n-1) \begin{bmatrix} (c22+...cnn) & -c12 & ...c1n \\ -c12 & (c11+ cnn) & ...-c2n \\ -c1n & -c2n... & (c11+c22...) \end{bmatrix}$$

3.3 Merging Clusters

In the ISODATA routine, the most important secondary criterion is that of cluster merging. In most applications of the standard ISODATA, cluster splitting and merging compete to produce a final number of output classes. The merging
parameter must walk a fine line, merging similar clusters without producing an elongated cluster that will be split on the next iteration (leading to an endless cycle with no convergence).

3.3.1 Standard Merging Problems

The standard merging criterion has problems because it works solely with cluster mean vectors and not with information about cluster distributions. It also suffers from the same problem of the standard splitting routine: it uses a relative merging parameter $\Theta$ which changes every iteration. It should be noted here that in the standard merging routine, $\Theta$ is best input as a fraction of the maximum distance found between cluster pairs. Any other method, such as direct distance, has been found in tests by the author to be too unwieldy for normal use.

3.3.2 Divergence

It now becomes obvious that to design a better merging routine, a criterion needs to be developed which can quantify the similarity between two clusters while taking into account their sample distributions. In the field of pattern recognition, there exists just such a measure, known as divergence. Divergence is a measure of the statistical dissimilarity between two classes, and can be used to evaluate the effectiveness of classification decisions.

The concept of divergence can be developed from the laws of conditional probability (Tou and Gonzalez, 1974). If the probability of having a data sample $x$, given that it belongs to cluster $w_i$ is $P(x|w_i)$, then a likelihood ratio can be developed between the two classes $w_i$ and $w_j$ as $U_{ij} = \ln \frac{P(x|w_i)}{P(x|w_j)}$. This ratio can be better understood through Figure 5, where it is apparent that the ratio is a measure of the likelihood of finding $x$ in class $w_i$ versus finding it in class $w_j$. 
Fig. 5. Conditional probabilities:

The expected value of the likelihood ratio for the class \( w_i \) is

\[
E\{U_{ij} \mid w_i\} = \int P(x \mid w_i) \ln \frac{P(x \mid w_i)}{P(x \mid w_j)} \, dx.
\]

There is also the expected value of the class \( w_j \)'s likelihood ratio, which is

\[
E\{U_{ji} \mid w_j\} = \int P(x \mid w_j) \ln \frac{P(x \mid w_j)}{P(x \mid w_i)} \, dx.
\]

\( U_{ij} \) is equal to \(-U_{ji} \), so this can be expressed as

\[
E\{U_{ji} \mid w_j\} = \int P(x \mid w_j) \ln \frac{P(x \mid w_j)}{P(x \mid w_j)} \, dx.
\]

The divergence between \( w_i \) and \( w_j \) is given as

\[
Div_{ij} = E\{U_{ij} \mid w_j\} + E\{U_{ji} \mid w_i\},
\]

and can be re-written as

\[
Div_{ij} = \int \left[ P(x \mid w_i) - P(x \mid w_j) \right] \ln \frac{P(x \mid w_i)}{P(x \mid w_j)} \, dx.
\]

What this essentially means is that the divergence is proportional to the shaded areas of Figure 5 showing dissimilarity between classes. As the two classes become more distinct, this area increases along with the divergence.

An equation for the divergence in multidimensional situations can be derived as long as the conditional probability \( P(x \mid w_i) \) is known. For multivariate normal distributions, the conditional probability can be defined as

\[
P(x \mid w_i) = \frac{1}{\sqrt{2\pi}^{n/2} |K_i|^{1/2}} \exp\left[ -\frac{1}{2} (x - m_i)^T K_i^{-1} (x - m_i) \right],
\]

where \(|K_i|\) is the determinant of the covariance matrix \( K_i \). The divergence can then be derived
\[
\text{Div}_{ij} = \frac{1}{2} \text{tr}[(K_i - K_j)(K_i^{-1} - K_j^{-1})] + \frac{1}{2} \text{tr}[(K_i^{-1} + K_j^{-1})(m_i - m_j)(m_i - m_j)^T]
\]

, where \( \text{tr} \) is the trace, or sum of the diagonal elements, of a matrix (Tou and Gonzalez, 1974).

When the divergence between two clusters is calculated, the resulting value can be difficult to interpret. To convert to a scale which is more easily understandable, the concept of transformed divergence has been developed (Kumar and Silva, 1977). The transformed divergence gives an exponentially decreasing value for the separability of a pair of classes and is expressed as

\[
T\text{Div}_{ij} = 2000(1 - \exp \left(-\frac{\text{Div}_{ij}}{8}\right)).
\]

A transformed divergence of 2000 indicates total separation of the classes while 1700 or less is poor according to Jensen (1986).

3.3.3 An Improved Merging Procedure

A good procedure for cluster merging can now be put together with the use of a criterion based on divergence. This criterion ensures that an absolute index of class separability is used and that merging should not interfere with cluster splitting when parameters are set to reasonable values. In this procedure, the transformed divergence is first calculated for every combination of cluster pairs. Next, any clusters with a transformed divergence less than the merging parameter \( \Theta \) are merged. \( \Theta \) is now a transformed divergence threshold; the output classification is guaranteed to have clusters with a transformed divergence greater than the minimum selected by the user.

3.4 An Improved Classifier

The ISODATA routine can now be rebuilt with these improved procedures for cluster splitting and merging. Theoretically, the process is straightforward,
but producing a working classification system is challenging and requires a whole host of additional procedures. In the practical development of the algorithm, there are more ways in which the standard ISODATA can be improved upon. The next chapter will describe a working example of a complete unsupervised classification system based on the improved ISODATA classifier.
4.0 IMPLEMENTATION

The development of this thesis came about partly as a result of a perceived need for a good unsupervised classification system to augment available image processing packages. The basic requirements for a useful classification system would consist of one or more classification routines, a means of display and printout, and a way to edit output classifications and easily examine their class statistics.

4.1 Software and Hardware Packages Used

Currently, two major software packages are available for users of remotely sensed data at the University of Wisconsin - Madison. The first package is the Environmental Remote Sensing Data Analysis System, or ERDAS, and is used at the University's Environmental Remote Sensing Center. The second package is the Man-Computer Interactive Data Access System, or McIDAS, developed by the University of Wisconsin Space Science and Engineering Center (SSEC). The idea to develop an unsupervised classification system for McIDAS came about because there was no available classification routine for the McIDAS system, and the ERDAS ISODATA routine previously produced such poor results as to be practically useless. The ERDAS corporation has recently rewritten its version of the ISODATA routine, which now produces good results and provides a benchmark for testing classification routines.

4.1.1 McIDAS

McIDAS is a very powerful data analysis system used in meteorology, remote sensing, and planetary science. It is unique in that it has the capability to ingest real-time data from geostationary and polar orbiting satellites (McIDAS Applications Guide, 1985). It also has the ability to process large amounts of data rapidly and display time sequences of images in a motion-picture fashion.
McIDAS is an open system and has the advantage of allowing the user to integrate additional routines into the main body of code through a custom compiler. It is currently available on several types of computers, including IBM PS/2 compatibles and IBM RISC workstations. The classification system was developed on the UNIX version of McIDAS, McIDAS-X, because of the speed of the workstation and its UNIX operating system which allows multitasking and integration of FORTRAN and C languages. In the time span of this project, McIDAS-X has gone from beta test to full release versions, and has proven to be an excellent host system for image classification.

McIDAS-X runs in an X-Windows environment and consists of a window for command input, an image window, and one or more output windows. An example of how McIDAS and its classification system appear to the user is shown in Image 1. The image window contains a number of frames, each of which may contain an image or graph. Only one of these frames may be displayed at a time, but the user can switch the display between frames rapidly. This feature can be very useful in viewing subtle changes in similar images or image classifications. Commands in McIDAS-X are entered through the command window, where they are processed in a background mode. The advantage of this is that the user is able to work on other projects while large, time consuming processes are running. Because of the mainframe origin of McIDAS, programs generally are not run in an interactive manner and all the information the program needs is given in advance.

4.1.2 ERDAS

The PC-ERDAS software package is a large conglomeration of image processing programs running on IBM PC compatible computers. PC-ERDAS contains several routines for multispectral image classification, including a version of the ISODATA routine, two one-pass methods for image classification,
and several routines for performing supervised image classification. The exact nature of the ERDAS ISODATA classifier is not described in the manuals, but can be inferred by watching it at work.

The ERDAS ISODATA routine requires initial parameters for the number of classes, the maximum number of iterations, and a skip factor. The skip factor works to speed up the routine by analyzing data only at pixels every \( n \)th row and column apart, where \( n \) is the skip factor. This can drastically reduce the time required, as a skip factor of four can make the classifier finish in one-sixteenth the time of a full classification. Increasing the skip factor could be expected to reduce the final number of output classes because the actual amount of data analyzed decreases. This is the case for the standard ISODATA classifier, but changing the skip factor in the ERDAS ISODATA classifier causes unpredictable changes in the classification output. This is because of a key characteristic of the ERDAS routine; the ERDAS ISODATA is a k-means classifier whose output number of classes is the same as that input. As with all k-means methods, this has the advantage of producing exactly the number of classes desired, but is insensitive to the actual number of clusters in the data. It is unknown whether ERDAS incorporates any cluster splitting or merging with its k-means approach. However, plots of cluster signatures suggest that a length-to-width means for cluster splitting is not used.

4.1.3 File Translation

To transfer files between ERDAS and McIDAS, a set of file translation programs were written early in the project. These programs allow the user to move images between systems and compare classification results in one system versus the other. Specifically, they translate image files between the ERDAS .LAN image format and the McIDAS area format, and translate classifications between the ERDAS .GIS format and McIDAS classification areas.
image classifications are composed of three files: .GIS files, .TRL files, and .SBD files. The .TRL files contain enhancement tables relating the display color to the class number, while the .SBD files contain statistical signature information about spectral classes. In McIDAS, the equivalent to the enhancement table file is the standard .ET file and the equivalent signature file is the .SIG file, a format developed for this project. There are 3 main programs which are listed in Appendix B: ATOERD.C, ERDTOA.C, and SBDSIG.C. ATOERD converts from McIDAS to ERDAS while ERDTOA converts from ERDAS to McIDAS and SBDSIG converts between .SIG and .SBD formats.

4.2 The McIDAS Classifier

The McIDAS classification system developed by the author consists of a set of original routines and existing functions that allow the user to classify and analyze multispectral images within the McIDAS system. In this new system, there are essentially three classification routines and a routine for visualizing the output classes of a classification. In addition, the McIDAS system provides routines for displaying classifications and changing the output enhancement table in order to highlight or merge classes.

The first two classification routines in the system consist of an implementation of the standard ISODATA and an MDM classification routine. The MDM routine reads in class signatures from .SIG files and is designed to classify additional images from clusters derived in an ERDAS ISODATA or McIDAS classification. The standard ISODATA routine uses a ratio based splitting parameter which is the largest allowable ratio of maximum to minimum standard deviations. The standard ISODATA subroutines for cluster splitting and merging are included in ISO.FOR in Appendix B.

The most important routine of the package is the improved ISODATA classifier, named USCLAS. USCLAS embodies all of the criteria proposed in
this thesis, including cluster splitting using moments of inertia and cluster merging using transformed divergence. The full USCLAS program is included in Appendix B.

4.2.1 USCLAS

USCLAS is run as a command in the McIDAS system, with parameters entered through the command line. The only required parameters are for the input and output areas, though a number of adjustable parameters may be entered. The default parameter settings are given in the help menu for the program. When first viewing the USCLAS help menu, the assortment of parameters might seem confusing. This should not be disconcerting, as the program is designed for ease of use and should produce good classification results using defaults. The additional parameters, however, can be very useful in gaining added control over the output of the classifier. The program parameters will now be discussed with an emphasis on their relation to the performance of the classifier.

The first optional parameter is \textit{nbands}, the number of bands. This parameter allows the user to run a classification using either all of the bands of an image, or only selected bands. The maximum number of bands is currently set at seven.

The next parameter, \textit{classes}, allows the user to choose the initial number of classes for the classifier to work with. The initial number of classes has an important and predictable effect on the classification output. Increasing the number of initial classes in USCLAS generally produces a larger number of output classes that meet all the other requirements of the classification. This effect is one of diminishing returns, however, and increasing the initial number of classes beyond a certain threshold ceases to change the output classification. A high value of around 200 initial classes has been found to be most effective for
finding as many distinct classes as possible in the data; however, values greater than 100 generally work well.

The *skip* parameter sets the classifier to analyze data only at every *n*th row and column. This speeds up the clustering by a factor of *n*², but at a cost of a poorer classification. Unlike the ERDAS approach, the skip factor in USCLAS is used until convergence is reached, at which point the whole image (*skip* set at 1) is classified. A good approach to getting to a desired type of output quickly is to run USCLAS classifications with a large skip factor until the most useful program settings are found, then run USCLAS on the whole image to produce the final output. The effects of varying the skip factor on a test image are illustrated in Figure 6. In these classifications of an AVHRR scene with 4 bands, increasing the skip factor generally produces fewer output classes. (The AVHRR and the scene used are described in Chapter 5.)

![Fig. 6. Output classes as a function of skip factor.](image)

The parameter *iter*, the maximum number of iterations allowed, has a default of 35. This should be enough in most cases, except for data where the image bands are not registered well to each other. In general, changing the other parameters in USCLAS makes the classifier either more or less demanding, which is reflected in an increase or decrease in the required
iterations. If USCLAS goes to the maximum number of iterations, it terminates and classifies the image using the last set of classes.

The merge and split parameters work as described earlier in the text. For merging, a transformed divergence value is entered. This value is the minimum transformed divergence allowed between cluster pairs. Although values of 1700 or greater are considered good theoretically (Jensen, 1986), practical values of around 1400 seem to show better results. The ERDAS ISODATA seems to agree with this value, as its output classifications often contain cluster pairs with transformed divergences of less than 1500.

For cluster splitting, a value for the ratio of the largest to smallest cluster dimensions is entered. The default ratio of three allows clusters to be essentially up to three times as long as they are wide. For cluster splitting as well as merging, an input of zero disables the routine.

The null parameter tells the MDM classifier how to handle data in which there are areas of no data for some or all of the bands. In the case where some bands are zero, USCLAS is generally forced to develop an entirely new set of classes for that area which have a zero mean in the empty bands. As this can be good or bad depending on the application, the null parameter can be set to classify pixels as zero if any one of the bands contains drop-outs, or only if all bands are empty. The classifier reserves the special class of zero for data drop out.

The minpix parameter is the key in defining the minimum number of pixels allowed in each cluster. In USCLAS, a cluster is defined as empty and is removed if there are fewer than the minimum number of pixels in that cluster. As many images contain noise pixels, setting the minimum number of pixels to a value greater than zero reduces the presence of small noise classes. Setting
this parameter to a value too high can eliminate small classes of useful information, however.

The *stretch* parameter in USCLAS allows the user to change the output appearance of the classification. In USCLAS, the output classes are given colors (described in the classification's enhancement table file) which try to match the 'colors' of the class mean vectors in the first three bands of the data. To make a more pleasing display, the colors are obtained from a histogram stretch of the mean vectors. The *stretch* parameter defines the type of stretch used: either linear or histogram equalization may be used.

One measure of the success of USCLAS as the practical implementation of the improved ISODATA classifier is in its response to the modified parameters. When changing the merging parameter, the classifier responds in a stable manner, as shown in Figure 7. Increasing the minimum transformed divergence in USCLAS makes for a more demanding classifier which reduces the output number of classes and increases the iterations required. At the extreme values of the merging parameter (no merging or no allowed cluster overlap), the classifier tends to an output equilibrium. Because of the predictable nature of this criterion, adjusting the merging parameter is the best way to fine-tune the number and characteristics of the output classes desired.
Cluster splitting also responds predictably to changes in the splitting factor for most images, as is shown in Figure 8. Due to the nature of the criterion however, the splitting factor is not recommended as a means to adjust the number of output classes, but rather as a check to ensure that class sizes are well behaved.

The speed of USCLAS on an IBM RISC workstation is very fast relative to an ISODATA routine running on a PC-compatible computer, but still requires a significant amount of time. As an example of the time required, a USCLAS
classification of a 512 by 512 image containing six bands takes from twenty to forty minutes to classify, depending on the number of iterations required. The same scene processed through the ERDAS ISODATA on a 33MHz 80486 based PC-compatible computer can require over five hours to complete. The large difference in speed is due mainly to the differences in processing power and disk access time between the two computers, but there are also some speed enhancements incorporated into USCLAS. The standard ISODATA routine is slowed considerably by its need to access the disk repeatedly. For every iteration, it must read in the image, read in the old classification for comparison, then write out the new classification. This heavy disk access creates a scenario where the classifier works at the speed of the disk drive, no matter how fast the processor. In USCLAS, the output classification is kept entirely in memory until the final write, thus reducing the three disk operations to only one disk read each iteration.

4.2.2 Cluster Visualization

The USCLAS program produces four separate output files which contain the information from the classification. The display part of the output consists of a McIDAS area containing classified values for the image classified and an .ET enhancement table to provide colors for these classes. The other two files are .SIG and .LOG files which contain the statistics of the classification. The .LOG file is a log of the classification's operations through every iteration, including detailed descriptions of changing cluster means and statistics.

The .SIG file may be used to visualize the positions of the output classes in the pattern space of the classification through the use of the cluster visualization program named ELLIPS. ELLIPS is so named because it plots elliptical plots of the statistical distribution of the data in each cluster. This program also plots only the locations of the mean vectors of the data if
requested, but the elliptical plots give a much better indication of the spread and overlap of classes.

ELLIPS draws a series of elliptical plots, going through all the possible combinations of axes two at a time. An example of an ELLIPS plot can be seen in Image 1, where the ellipses of the displayed classification are drawn for bands two and three. To follow the process ELLIPS goes through, a two-dimensional covariance matrix can be analyzed:

$$K_k = \begin{bmatrix} c_{11} & c_{12} \\ c_{12} & c_{22} \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & c_{12} \\ c_{12} & \sigma_2^2 \end{bmatrix}.$$  

A plot of a class distribution for a value of two standard deviations out from the mean vector would appear similar to Figure 9.

![Elliptical distribution plots.](image)

In the ELLIPS program, a plot connecting the four points along the principal axes of the ellipse is performed for every class in the output. To obtain these points, the value of $\theta$ is first computed as $\theta = \tan^{-1} \left[ \frac{\sigma_2}{\sigma_1} \right]$. Coordinates for ellipse axes points are simply computed in the manner of
\[ x_I = m_I + 2\sigma I \cos \theta \] and \[ y_I = m_I + 2\sigma I \sin \theta. \] ELLIPS uses two standard deviations as a default value, but will also accept any other value as an input parameter.

4.3 Performing a Classification

Now that the components of the classification have been described in detail, the classification procedure will be described to help clarify the way the system works.

To begin an image classification, a multi-band McIDAS area must be first obtained. (USCLAS will classify a single-band image, but this amounts to performing a level slicing operation.) If the image is in the form of several single-band areas, then the user must first run the McIDAS COMBIN routine, which was developed by the author to make a multi-band image from single-band components.

The next step is to run USCLAS. A help menu for the program can be obtained through the HELP USCLAS command. When running this or any other classification routine, the best approach is to switch to a separate output window, start the classifier, then switch back to the first output window. The USCLAS output, which includes progress updates every iteration, can then be monitored separately while other McIDAS routines are run.

Once USCLAS has finished running, the user may display the output area in the image window, using the command EU REST nnnn to color the classification image. nnnn is the output area number; all the USCLAS output files are written as this number plus an extension. Class colors may be modified with the EU command. Classes may also be merged by making component clusters of an information class the same color.

To aid in the analysis of output classes, the ELLIPS program may be called to create a window containing a plot of statistical ellipses or class means.
Several ELLIPS plots may be run at once; the only limitation is with available screen area for viewing.

After processing, the classification output may also be used as input for other McIDAS programs. The MDCLAS program, for example, can classify later images in a time series with the classes obtained from the first image of the series.
5.0 APPLICATIONS AND TESTS

In the course of its development, the USCLAS classification program has been used to classify a wide variety of test images. Classifications have been run on images from the Landsat Thematic Mapper (TM), SPOT High Resolution Visible (HRV) scanner, NOAA Advanced Very High Resolution Radiometer (AVHRR), Voyager spacecraft, and digitized aerial photography. All of these classifications have produced results that are appealing and useful to the user, so it could be argued that USCLAS is successful at producing good unsupervised classifications on a wide variety of data. It is important, however, to define just how good the classifier is in comparison with other classification methods.

To investigate the usefulness of the USCLAS program, three test images have been selected to provide working comparisons. These images consist of an AVHRR image of Kuwait and the Persian Gulf during the Gulf War, a Landsat TM image of agriculture in central Wisconsin, and a planetary mosaic of Jupiter created from Voyager spacecraft images. The three images were chosen specifically to allow USCLAS to be compared to as many other methods as possible. The Kuwait image provides a difficult scene for unsupervised classifiers due to the presence of gradually changing smoke features. The Wisconsin image, on the other hand, provides an easily classified scene for which a supervised classification made with ground truth is available. The Jupiter image used for the third scene is useful for comparison with published results of an unsupervised classification on a similar image.
5.1 A Challenging AVHRR Scene

The first scene to be analyzed is the Kuwait image: a NOAA-10 AVHRR image obtained on February 23, 1991. The AVHRR scanner has a resolution of approximately 1km per pixel, with four spectral bands of data. There is one band each for red visible light at .58-.68μm, near-Infrared (IR) at .7-1.1μm, mid-IR at 3.5-3.9μm (the spectral region of equal solar reflectance and thermal emission), and thermal IR at 10.5-11.3μm.

The Kuwait image is a complex scene with many interesting features, as can be seen from the visible and mid-IR bands displayed in Images 2 and 3. The most significant feature is the oil-fire smoke that drifts over the desert and the Persian Gulf. Smoke presents a challenge in classification because of its gradually changing density and spectral signature. It affects the signatures of the features it covers by lowering their effective albedo, and is very hard to differentiate over water. Another image feature which complicates classification comes in the form of sun-glint over the southern Persian Gulf. This saturates the mid-IR and thermal IR channels and is difficult to distinguish from the oil fires present in the image.

With these difficulties in mind, a series of classifications were performed to attempt a mapping of image features and smoke densities. First, the ERDAS ISODATA classifier was used to produce two classifications, one with 26 output classes and one with 40. Initial parameters for the ERDAS classifier were a skip factor of 1 and the same convergence criteria (96% of pixels unchanged) as that used in USCLAS. Next, USCLAS classified the image with two different merging parameter settings (transformed divergences of 1400 and 1450) to produce classifications with 44 and 24 classes, respectively. The USCLAS classifications used initial parameters of 200 initial classes, a skip factor of 1, and a splitting factor of 3. The 44-class USCLAS classification is shown in Image 4 along with
ellipse plots of the least correlated bands. The 40-class ERDAS classification is printed out in a similar manner in Image 5.

There are several dissimilarities between the two classifications which show the strengths and weaknesses of each. The first thing to be noticed about the two classifications is the difference in class colors. This comes from the use of a histogram equalization stretch in USCLAS; which creates more vivid colors at the cost of a less "natural" looking output. It is also evident that the ERDAS classification contains more classes in the general area of smoke-free desert. Although these classes might be useful for analysis, their transformed divergences are small and their shapes are subject to change with slight differences in initial parameters. A test of USCLAS using a transformed divergence merging parameter of 1350 brought out more land classes to make the output look similar to that of the ERDAS classification.

The most significant difference in the two classifiers is in the classification of the actual oil fires. In Images 6 and 7, the two classifications are shown in closer detail with water and fire classes highlighted for visibility. Here the advantage of the USCLAS splitting criteria over that of the ERDAS ISODATA routine can be clearly seen. For in the ERDAS classification, the oil fires are not composed of separate classes as in the USCLAS output, but are misclassified into several water classes. Increasing the number of classes in the ERDAS classifier does not solve this problem, as it persists for a wide range of input class numbers. In contrast, USCLAS finds several unique smoke classes throughout a wide range of input parameters when a reasonable splitting parameter is used.

The differences in classifications can also be seen in the ellipse plots of Images 4 and 5. In the ERDAS classification, the area of pattern space corresponding to smoke-free desert is crowded with a host of classes which
overlap in all band combinations. In contrast, the same area of the USCLAS output contains fewer classes in a quilted pattern with little overlap. The USCLAS output also contains many more classes in the spectral region of the snow covered areas of Iran (in the northeast corner of the image). These classes are most visible in the upper-right region of the ellipse plot for bands 1 and 3, and contain significant information that has been missed in the ERDAS classification.

The results of classifying this image and other similar scenes of the Kuwaiti oil fires show that the USCLAS classifier is the better approach to the problem of mapping oil fires and smoke densities in multispectral images.

5.2 Landsat Scene of Central Wisconsin

A Landsat TM scene of agricultural areas around Rush Lake (near Oshkosh, Wisconsin) is the next scene to be analyzed. This image, dated May 7, 1990, is one of a series of scenes for which supervised classifications with ground truth have been performed (Polzer, 1992). In the supervised classification of this scene, 174 initial training sets were taken and classified using the IR bands 3, 4, 5, and 7 of the Thematic Mapper. The results were then merged to make a classification with 42 output classes and a final display classification with a further reduction to 24 information classes. The final display classification was made with a majority smoothing filter and was not used in this comparison.

To test the USCLAS and ERDAS ISODATA classifiers against the supervised classification, a 600 by 600 pixel subset of the scene containing a large density of training sets was selected. This subset provides a test image of manageable size for repeated classifications. Although not reflecting the true diversity of the full scene, the subset image still contains most of its spectral
variation. For this reason, it is not unfair to compare classifications derived from this image with the supervised full scene classification.

To obtain results comparable to the supervised classification, the scene subset was classified by USCLAS using a skip factor of 1, a splitting factor of 3, and a merging factor of 1375, with 200 initial classes. This produced an output classification containing 36 classes. The image was also classified with the ERDAS ISODATA routine with the number of classes set at 42. These two classifications were then compared with a subset of the supervised classification.

The results of the classifiers can be seen in Images 8, 9, and 10, where a portion of the subset has been displayed for the USCLAS, ERDAS ISODATA, and supervised classifications, respectively. The darkest classes in the USCLAS and ERDAS images have been enhanced for greater visibility, and are shown in shades of blue. Also included in Images 8 and 9 are ellipse plots of the two band combinations with the least correlation. In these plots, band 2 represents the .76-.90\(\mu\)m TM band 4, band 3 represents the 1.55-1.75\(\mu\)m TM band 5, and band 4 represents the 2.08-2.35\(\mu\)m TM band 7.

It can be seen that the USCLAS and ERDAS results are similar for this scene with only a few differences. As in the Kuwait image, the ellipse plots show that the USCLAS output classes are more distinct and better distributed than the ERDAS ISODATA classes. The USCLAS classification also contains several more water-edge classes, including classes containing only a few pixels at the edge of Rush Lake. These classes are statistically unique, and may or may not be useful to the operator depending on the application. The \textit{minpix} parameter of USCLAS can be very useful for this type of situation by allowing the user to eliminate these small classes if desired.
In comparison with the unsupervised classifications, the supervised output is much more difficult to interpret. Initially, the dissimilarity between color enhancement tables might lead the interpreter to consider the supervised results to be inferior, but a more careful analysis should be performed. Such an analysis requires an intimate knowledge of the scene and classification methods used, and is beyond the scope of this paper. However, some useful observations can still be made in comparing classifications. In general, the supervised classification appears less homogenous than the unsupervised results, and areas used as training sets often contain more than one class. In contrast, these training set areas in USCLAS are more homogenous and contain better separated classes. The accuracy of the USCLAS output based on training areas would likely be higher than that of the supervised classification because of this. A credit to the supervised classification comes in its separate class for roads (not shown in the image), which are only detectable as changes in class type in the unsupervised classifications.

5.3 Voyager Planetary Image

The final image to be analyzed is a mosaic of the planet Jupiter derived from images taken by the Voyager 1 spacecraft. It serves as an example of the application of unsupervised classification techniques to the analysis of cloud motions and cloud compositions. What makes this image useful as a test is that it may be compared with published work on a statistical clustering analysis performed on similar data (Thompson, 1990).

The image used is essentially a rectified map of the surface features of the planet Jupiter, and is one of a series of planetary mosaics made from images taken during the spacecraft's flyby of Jupiter in 1979. The images in this series cover 360 degrees of longitude with 9 pixels per degree for a total of 3240 image
elements. In latitude, they cover areas from 60 degrees north latitude to 60 degrees south latitude with 8 pixels per degree for a total of 960 image lines. Images using the .345\(\mu\)m ultra-violet (UV) filter, the .481\(\mu\)m blue filter, the .561\(\mu\)m green filter, and the .589\(\mu\)m orange filter were combined to make the four band image used for classification. Some problems with this type of image that make classification difficult include areas of data drop-out in one or more bands, areas of blurry imaging in the UV band, and poor geometrical registration between bands mainly in areas of high latitudes.

USCLAS was used to classify the Jupiter mosaic image with several different sets of input parameters in order to develop a feeling for how classifications of this type of image would turn out. Because of the registration problems, the USCLAS mosaic classifications exhibit a steep curve for the number of output classes as a function of the merging factor. Two classifications with good results were used for analysis: one in the middle range of output classes and one near the upper equilibrium number (where increasing the merging factor further ceases to affect the output). The first classification resulted in 26 output classes for parameters of 200 initial classes, split factor of 3, skip of 4, and merging factor of 1375. The second classification produced 38 classes with the merging factor at 1350. Image 11 shows the first classification with two accompanying ellipse plots of the least correlated bands of the data. The classes are seen to be well-behaved, except for classes 3, 6 and 14 which represent areas of poor (but not zero valued) data in band 4 (orange). For this classification the \textit{null} parameter was used in the \textit{null=any} setting to classify data as zero when any band contains a dropout. For the mosaic images, this helps to produce a neater classification.

The differences in output between the first and second classifications may be seen in Images 12 and 13. These cover a portion of the image in greater
detail, showing cloud structure around the Great Red Spot. The 38 class classification obviously contains more gradations in cloud types; whether this is an advantage or not is dependant on the application. In both cases, the classifications differentiate the separate cloud features well.

The USCLAS classifications of these Jovian mosaics have proven to be useful in current work on the cloud structure and mean zonal flows of Jupiter, but there remains the question of what the best number of output classes to work with is. Thompson, 1990, helps to answer this in his classification of similar composite images from Voyager 1 and Voyager 2. Thompson performed unsupervised classifications on Jovian mosaics using four bands composed of the green and orange filters along with the clear and narrowband Methane filters. Because the UV filter was not used, the images used were better in general than those classified by USCLAS and contained no drop-outs. In addition to the classification, a statistical package was used to determine the number of clusters which could be considered statistically significant. The results of this analysis showed there to be approximately 40 significant clusters, with 25 being of sufficient size to merit keeping. Thompson's results suggest that the USCLAS classifier is adept at finding the significant clusters in mosaic data, but the input merging parameter should be varied through a few classification runs in order to produce the best results for the application at hand.
6.0 SUMMARY

The goal of this thesis has been to present an unsupervised classification system which is accurate, easy to use, and more discriminating than other available methods. To reach this goal, a solid analysis of the most popular unsupervised classification methods was first presented, concentrating on the relative strengths and weaknesses of different methods. Next, the minimum variance criterion and the ISODATA method were shown to make a good basic framework for a new unsupervised classification routine. Additional criteria were then added to improve the classifier and provide more control over the nature of the output. Finally, a useful classification system was presented which integrates the new USCLAS classifier and a cluster visualization technique into an existing image processing package.

The new classification system was tested on a wide variety of images until the author was satisfied that its results are at least as useful as those from other available methods. In practice, the USCLAS classifier has performed as predicted, and has shown to be easily adjustable for optimizing the output to the application at hand. Additional comparisons of USCLAS to supervised classifiers in future projects would be useful in better defining the relative strengths and weaknesses of both approaches. The author encourages researchers to try this classification system along with traditional methods when working on detailed classification projects.
BIBLIOGRAPHY


